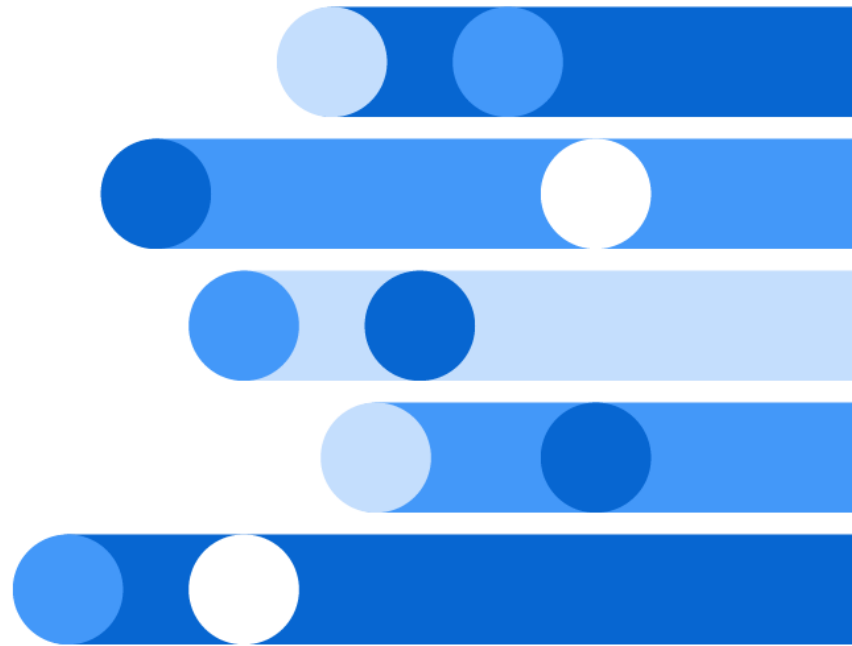




SAS[®] Econometrics Econometrics Procedures

2026.07*



* This document might apply to additional versions of the software. Open this document in SAS Help Center and click on the version in the banner to see all available versions.



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2026. *SAS® Econometrics: Econometrics Procedures*. Cary, NC: SAS Institute Inc.

SAS® Econometrics: Econometrics Procedures

Copyright © 2026, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

July 2026

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

SAS software may be provided with certain third-party software, including but not limited to open source software, which is licensed under its applicable third-party software license agreement. For license information about third-party software distributed with SAS software, refer to [Third-Party Software Reference | SAS Support](#).

Contents

Chapter 1.	Introduction	1
Chapter 2.	Introduction to Bayesian Analysis Procedures	5
Chapter 3.	Introduction to Packages for the TSMODEL Procedure	45
Chapter 4.	Shared Concepts	53
Chapter 5.	CARIMA Procedure	113
Chapter 6.	CAUSALDISCOVERY Procedure	135
Chapter 7.	CCDM Procedure	171
Chapter 8.	CCOPULA Procedure	285
Chapter 9.	CESM Procedure	337
Chapter 10.	CNTSELECT Procedure	375
Chapter 11.	CPANEL Procedure	461
Chapter 12.	CQLIM Procedure	559
Chapter 13.	CSPATIALREG Procedure	635
Chapter 14.	CSSM Procedure	717
Chapter 15.	DEEPCAUSAL Procedure	907
Chapter 16.	DEEPPRICE Procedure	953
Chapter 17.	DYNAMICLINEAR Procedure	1027
Chapter 18.	ECM Procedure	1097
Chapter 19.	FRONTIER Procedure	1145
Chapter 20.	HMM Procedure	1183
Chapter 21.	MKTATTRIBUTION Procedure	1415
Chapter 22.	SASEBEA Interface Engine	1437
Chapter 23.	SASEBLS Interface Engine	1595
Chapter 24.	SASEMOOD Interface Engine	1671
Chapter 25.	SASEUSCB Interface Engine	1697
Chapter 26.	SEVSELECT Procedure	1721
Chapter 27.	SMC Procedure	1901
Chapter 28.	TSINFO Procedure	2059
Chapter 29.	TSMODEL Procedure	2067
Chapter 30.	UCM Procedure	2111
Chapter 31.	Time Series Analysis Package	2241
Chapter 32.	Time Series Model Package	2339
Chapter 33.	Utility Package	2445

Chapter 1

Introduction

Contents

Overview of SAS Econometrics Procedures	1
About This Book	2
Chapter Organization	2
Typographical Conventions	3
Options Used in Examples	3
Where to Turn for More Information	4
Online Documentation	4
SAS Technical Support Services	4

Overview of SAS Econometrics Procedures

SAS Econometrics procedures include predictive modeling tools that have been specially developed to take advantage of the distributed environment that SAS Viya provides. Methods include time series, compound distribution modeling, copula fitting and simulation, count data, panel data, limited dependent variables, spatial regression, economic capital modeling, hidden Markov models and severity modeling. In addition, these procedures provide many diagnostic tests, variable and model selection methods, visualization and distribution fitting comparisons that can help you to find the best model for your analysis.

NOTE: When you license SAS Econometrics, you also have access to SAS/ETS software. For more information about SAS/ETS procedures, see *SAS/ETS User's Guide*.

About This Book

This book assumes that you are familiar with Base SAS software and with the books *SAS Programmers Guide: Essentials* and *Base SAS Procedures Guide*. It also assumes that you are familiar with basic SAS System concepts, such as using the DATA step to create SAS data sets and using Base SAS procedures (such as the PRINT and SORT procedures) to manipulate SAS data sets.

Chapter Organization

This book contains the following chapters:

- **Chapter 1**, this chapter, provides an overview of SAS Econometrics procedures and summarizes related information, products, and services.
- **Chapter 4** provides information about topics that are common to multiple procedures. Topics include how to use SAS Cloud Analytic Services (CAS) sessions and how to load a SAS data set onto a CAS server. This chapter also documents the following statements that are used across a number of procedures: CLASS, DISPLAY, DISPLAYOUT, EFFECT, and SELECTION.
- Next are chapters that describe the procedures; they appear in alphabetical order by procedure name and are organized as follows:
 - The “Overview” section briefly describes the analysis provided by the procedure.
 - The “Getting Started” section provides a quick introduction to the procedure through a simple example.
 - The “Syntax” section describes the SAS statements and options that control the procedure.
 - The “Details” section discusses methodology and other topics, such as ODS tables.
 - The “Examples” section contains examples that use the procedure.
 - The “References” section contains references for the methodology.
- The last chapters in the book describe packages that you can use with the TSMODEL procedure. For more information about using packages, see *SAS Visual Forecasting: Time Series Packages*.

Typographical Conventions

This book uses several type styles for presenting information. The following list explains the meaning of the typographical conventions used in this book:

<i>roman</i>	is the standard type style used for most text.
UPPERCASE ROMAN	is used for SAS statements, options, and other SAS language elements when they appear in text. However, you can enter these elements in your own SAS programs in lowercase, uppercase, or a mixture of the two.
UPPERCASE BOLD	is used in the “Syntax” sections’ initial lists of SAS statements and options.
<i>oblique</i>	is used in the syntax definitions and in text to represent arguments for which you supply a value.
VariableName	is used for the names of variables and data sets when they appear in text.
bold	is used for matrices and vectors.
<i>italic</i>	is used for terms that are defined in text, for emphasis, and for references to publications.
monospace bold	is used for example code. In most cases, this book uses lowercase type for SAS code.

Options Used in Examples

The HTMLBLUE style is used to create the graphs and the HTML tables that appear in the online documentation. The PEARLJ style is used to create the PDF tables that appear in the documentation. A style template controls stylistic elements such as colors, fonts, and presentation attributes. You can specify a style template for an HTML ODS destination as follows:

```
ods html style=HTMLBlue;
```

You can also specify a style template for a PDF ODS destination as follows:

```
ods pdf style=PearlJ;
```

Most of the PDF tables are produced by using the following SAS System option:

```
options papersize=(6.5in 9in);
```

If you run the examples, you might get slightly different output. This is a function of the SAS System options that are used and the precision that your computer uses for floating-point calculations.

Where to Turn for More Information

Online Documentation

You can access the documentation by going to <http://support.sas.com/documentation>.

SAS Technical Support Services

The SAS Technical Support staff is available to respond to problems and answer technical questions regarding the use of procedures in this book. Go to <http://support.sas.com/techsup> for more information.

Chapter 2

Introduction to Bayesian Analysis Procedures

Contents

Overview	5
Introduction	6
Background in Bayesian Statistics	8
Prior Distributions	8
Bayesian Inference	9
Bayesian Analysis: Advantages and Disadvantages	11
Standard Prior Distributions	12
Markov Chain Monte Carlo Method	16
Assessing Markov Chain Convergence	27
Summary Statistics	35
Using Multiple Chains	37
A Bayesian Reading List	39
Introductory Books	39
Intermediate-Level Books	39
Advanced Titles	40
Books Motivated by Statistical Applications and Data Analysis	41
References	41

Overview

SAS Econometrics software provides Bayesian capabilities in three procedures: CQLIM, CNTSELECT, and SMC. The CQLIM and CNTSELECT procedures provide Bayesian analyses in addition to the standard frequentist analyses that they have always performed. PROC CQLIM enables you to perform Bayesian analysis of limited dependent variable models, and PROC CNTSELECT enables you to perform Bayesian analysis of count regression models. PROC SMC enables you to perform Bayesian analysis of nonlinear time series models by using sequential Monte Carlo methods.

This chapter provides an overview of Bayesian statistics; describes specific sampling algorithms used in these procedures; and discusses posterior inference and convergence diagnostics computations. Sources that provide in-depth treatment of Bayesian statistics can be found at the end of this chapter, in the section “A Bayesian Reading List” on page 39. Additional chapters contain syntax, details, and examples for the individual procedures CQLIM (see Chapter 12, “CQLIM Procedure”), CNTSELECT (see Chapter 10, “CNTSELECT Procedure”), and SMC (see Chapter 27, “SMC Procedure”).

Introduction

The most frequently used statistical methods are known as *frequentist* (or *classical*) methods. These methods assume that unknown parameters are fixed constants, and they define probability by using limiting relative frequencies. It follows from these assumptions that probabilities are objective and that you cannot make probabilistic statements about parameters because they are fixed. Bayesian methods offer an alternative approach; they treat parameters as random variables and define probability as “degrees of belief” (that is, the probability of an event is the degree to which you believe the event is true). It follows from these postulates that probabilities are subjective and that you can make probability statements about parameters. The term “Bayesian” comes from the prevalent usage of Bayes’ theorem, which was named after the Reverend Thomas Bayes, an 18th-century Presbyterian minister. Bayes was interested in solving the question of inverse probability: after observing a collection of events, what is the probability of one event?

Suppose you are interested in estimating θ from data $\mathbf{y} = \{y_1, \dots, y_n\}$ by using a statistical model described by the density $\pi(\mathbf{y}|\theta)$. Bayesian philosophy states that θ cannot be determined exactly, and uncertainty about the parameter is expressed through probability statements and distributions. You can say that θ follows a normal distribution with mean 0 and variance 1, if it is believed that this distribution best describes the uncertainty associated with the parameter. The following steps describe the essential elements of Bayesian inference:

1. A probability distribution for θ is formulated as $\pi(\theta)$, which is known as the *prior* distribution, or just the prior. The prior distribution expresses your beliefs (for example, on the mean, the spread, the skewness, and so forth) about the parameter before you examine the data.
2. Given the observed data $\mathbf{y}$, you choose the statistical model $\pi(\mathbf{y}|\theta)$ to describe the distribution of $\mathbf{y}$ given θ .
3. You update your beliefs about θ by combining information from the prior distribution and the data through the calculation of the *posterior* distribution, $\pi(\theta|\mathbf{y})$.

The third step is carried out by using Bayes’ theorem, which enables you to combine the prior distribution and the model in the following way:

$$\pi(\theta|\mathbf{y}) = \frac{\pi(\theta, \mathbf{y})}{\pi(\mathbf{y})} = \frac{\pi(\mathbf{y}|\theta)\pi(\theta)}{\pi(\mathbf{y})} = \frac{\pi(\mathbf{y}|\theta)\pi(\theta)}{\int \pi(\mathbf{y}|\theta)\pi(\theta)d\theta}$$

The quantity

$$\pi(\mathbf{y}) = \int \pi(\mathbf{y}|\theta)\pi(\theta)d\theta$$

is the normalizing constant of the posterior distribution. The quantity $\pi(\mathbf{y})$ is also the marginal distribution of $\mathbf{y}$, and it is sometimes called the marginal distribution of the data. The likelihood function of θ is any function proportional to $\pi(\mathbf{y}|\theta)$; that is, $L(\theta) \propto \pi(\mathbf{y}|\theta)$. Another way of writing Bayes’ theorem is as follows:

$$\pi(\theta|\mathbf{y}) = \frac{L(\theta)\pi(\theta)}{\int L(\theta)\pi(\theta)d\theta}$$

The marginal distribution $\pi(\mathbf{y})$ is an integral. As long as the integral is finite, the particular value of the integral does not provide any additional information about the posterior distribution. Hence, $\pi(\theta|\mathbf{y})$ can be written up to an arbitrary constant, presented here in proportional form as

$$\pi(\theta|\mathbf{y}) \propto L(\theta)\pi(\theta)$$

Simply put, Bayes' theorem tells you how to update existing knowledge with new information. You begin with a prior belief $\pi(\theta)$, and after learning information from data $\mathbf{y}$, you change or update your belief about θ and obtain $\pi(\theta|\mathbf{y})$. These are the essential elements of the Bayesian approach to data analysis.

In theory, Bayesian methods offer simple alternatives to statistical inference—all inferences follow from the posterior distribution $\pi(\theta|\mathbf{y})$. In practice, however, you can obtain the posterior distribution with straightforward analytical solutions only in the most rudimentary problems. Most Bayesian analyses require sophisticated computations, including the use of simulation methods, such as the Markov chain Monte Carlo (MCMC) methods. You generate samples from the posterior distribution and use these samples to estimate the quantities of interest. All SAS Econometrics procedures use a self-tuning random walk Metropolis (RWM) algorithm to generate these samples (see the sections “[Metropolis and Metropolis-Hastings Algorithms](#)” on page 17 and “[Gibbs Sampler](#)” on page 18). An important aspect of any analysis is assessing the convergence of the Markov chains. Inferences that are based on nonconverged Markov chains can be both inaccurate and misleading.

Both Bayesian and classical methods have their advantages and disadvantages. From a practical point of view, your choice of method depends on what you want to accomplish with your data analysis. If you have prior information (either expert opinion or historical knowledge) that you want to incorporate into the analysis, then you should consider Bayesian methods. In addition, if you want to communicate your findings in terms of probability notions that can be more easily understood by nonstatisticians, Bayesian methods might be appropriate. The Bayesian paradigm can often provide a framework for answering specific scientific questions that a single point estimate cannot sufficiently address. Alternatively, if you are interested only in estimating parameters based on the likelihood, then numerical optimization methods, such as the Newton-Raphson method, can give you very precise estimates and there is no need to use a Bayesian analysis. For further discussions of the relative advantages and disadvantages of Bayesian analysis, see the section “[Bayesian Analysis: Advantages and Disadvantages](#)” on page 11.

Background in Bayesian Statistics

Prior Distributions

A prior distribution of a parameter is the probability distribution that represents your uncertainty about the parameter before the current data are examined. Multiplying the prior distribution and the likelihood function together leads to the posterior distribution of the parameter. You use the posterior distribution to carry out all inferences. You cannot carry out any Bayesian inference or perform any modeling without using a prior distribution.

Objective Priors versus Subjective Priors

Bayesian probability measures the degree of belief that you have in a random event. By this definition, probability is highly subjective. It follows that all priors are *subjective priors*. Not everyone agrees with this notion of subjectivity when it comes to specifying prior distributions. There has long been a desire to obtain results that are objectively valid. Within the Bayesian paradigm, this can be somewhat achieved by using prior distributions that are “objective” (that is, that have a minimal impact on the posterior distribution). Such distributions are called *objective* or *noninformative* priors (see the next section). However, while noninformative priors are very popular in some applications, they are not always easy to construct. See DeGroot and Schervish (2002, Section 1.2) and Press (2003, Section 2.2) for more information about interpretations of probability. See Berger (2006) and Goldstein (2006) for discussions about objective Bayesian versus subjective Bayesian analysis.

Noninformative Priors

Roughly speaking, a prior distribution is noninformative if the prior is “flat” relative to the likelihood function. Thus, a prior $\pi(\theta)$ is noninformative if it has minimal impact on the posterior distribution of θ . Other names for the noninformative prior are *vague*, *diffuse*, and *flat* prior. Many statisticians favor noninformative priors because they appear to be more objective. However, it is unrealistic to expect that noninformative priors represent total ignorance about the parameter of interest. In some cases, noninformative priors can lead to *improper posteriors* (nonintegrable posterior density). You cannot make inferences with improper posterior distributions. In addition, noninformative priors are often not invariant under transformation; that is, a prior might be noninformative in one parameterization but not necessarily noninformative if a transformation is applied.

See Box and Tiao (1973) for a more formal development of noninformative priors. See Kass and Wasserman (1996) for techniques for deriving noninformative priors.

Improper Priors

A prior $\pi(\theta)$ is said to be improper if

$$\int \pi(\theta) d\theta = \infty$$

For example, a uniform prior distribution on the real line, $\pi(\theta) \propto 1$, for $-\infty < \theta < \infty$, is an improper prior. Improper priors are often used in Bayesian inference since they usually yield noninformative priors

and proper posterior distributions. Improper prior distributions can lead to posterior impropriety (improper posterior distribution). To determine whether a posterior distribution is proper, you need to make sure that the normalizing constant $\int \pi(\mathbf{y}|\theta)\pi(\theta)d\theta$ is finite for all $\mathbf{y}$. If an improper prior distribution leads to an improper posterior distribution, inference based on the improper posterior distribution is invalid.

Both the CQLIM and CNTSELECT procedures allow the use of improper priors—that is, flat priors on the real line—for regression coefficients and other parameters. These improper priors can lead to improper posterior distributions, so you must ensure that they are proper.

Informative Priors

An informative prior is a prior that is not dominated by the likelihood and that has an impact on the posterior distribution. If a prior distribution dominates the likelihood, it is clearly an informative prior. These types of distributions must be specified with care in actual practice. On the other hand, the proper use of prior distributions illustrates the power of the Bayesian method: information gathered from the previous study, past experience, or expert opinion can be combined with current information in a natural way.

Conjugate Priors

A prior is said to be a conjugate prior for a family of distributions if the prior and posterior distributions are from the same family, which means that the form of the posterior has the same distributional form as the prior distribution. For example, if the likelihood is binomial, $y \sim \text{Bin}(n, \theta)$, a conjugate prior on θ is the beta distribution; it follows that the posterior distribution of θ is also a beta distribution. Other commonly used conjugate prior/likelihood combinations include the normal/normal, gamma/Poisson, gamma/gamma, and gamma/beta cases. The development of conjugate priors was partially driven by a desire for computational convenience—conjugacy provides a practical way to obtain the posterior distributions. The Bayesian procedures do not use conjugacy in posterior sampling.

Bayesian Inference

Bayesian inference about θ is primarily based on the posterior distribution of θ . There are various ways in which you can summarize this distribution. For example, you can report your findings through point estimates. You can also use the posterior distribution to construct hypothesis tests or probability statements.

Point Estimation and Estimation Error

Classical methods often report the maximum likelihood estimator (MLE) or the method of moments estimator (MOME) of a parameter. In contrast, Bayesian approaches often use the posterior mean. The definition of the posterior mean is given by

$$E(\theta|\mathbf{y}) = \int \theta \pi(\theta|\mathbf{y}) d\theta$$

Other commonly used posterior estimators include the posterior median, defined as

$$\theta: P(\theta \geq \text{median}|\mathbf{y}) = P(\theta \leq \text{median}|\mathbf{y}) = \frac{1}{2}$$

and the posterior mode, defined as the value of θ that maximizes $\pi(\theta|\mathbf{y})$.

The variance of the posterior density (simply referred to as the *posterior variance*) describes the uncertainty in the parameter, which is a random variable in the Bayesian paradigm. A Bayesian analysis typically uses the posterior variance, or the posterior standard deviation, to characterize the dispersion of the parameter. In multidimensional models, covariance or correlation matrices are used.

If you know the distributional form of the posterior density of interest, you can report the exact posterior point estimates. When models become too difficult to analyze analytically, you have to use simulation algorithms, such as the MCMC method to obtain posterior estimates (see the section “[Markov Chain Monte Carlo Method](#)” on page 16). All the Bayesian procedures rely on MCMC to obtain all posterior estimates. Using only a finite number of samples, simulations introduce an additional level of uncertainty to the accuracy of the estimates. *Monte Carlo standard error (MCSE)*, which is the standard error of the posterior mean estimate, measures the simulation accuracy. See the section “[Standard Error of the Mean Estimate](#)” on page 36 for more information.

The posterior standard deviation and the MCSE are two completely different concepts: the posterior standard deviation describes the uncertainty in the parameter, whereas the MCSE describes only the uncertainty in the parameter estimate as a result of MCMC simulation. The posterior standard deviation is a function of the sample size in the data set, and the MCSE is a function of the number of iterations in the simulation.

Hypothesis Testing

Suppose you have the following null and alternative hypotheses: H_0 is $\theta \in \Theta_0$ and H_1 is $\theta \in \Theta_0^c$, where Θ_0 is a subset of the parameter space and Θ_0^c is its complement. Using the posterior distribution $\pi(\theta|\mathbf{y})$, you can compute the posterior probabilities $P(\theta \in \Theta_0|\mathbf{y})$ and $P(\theta \in \Theta_0^c|\mathbf{y})$, or the probabilities that H_0 and H_1 are true, respectively. One way to perform a Bayesian hypothesis test is to accept the null hypothesis if $P(\theta \in \Theta_0|\mathbf{y}) \geq P(\theta \in \Theta_0^c|\mathbf{y})$ and vice versa, or to accept the null hypothesis if $P(\theta \in \Theta_0|\mathbf{y})$ is greater than a predefined threshold, such as 0.75, to guard against falsely accepted null distribution.

It is more difficult to carry out a point null hypothesis test in a Bayesian analysis. A point null hypothesis is a test of $H_0: \theta = \theta_0$ versus $H_1: \theta \neq \theta_0$. If the prior distribution $\pi(\theta)$ is a continuous density, then the posterior probability of the null hypothesis being true is 0, and there is no point in carrying out the test. One alternative is to restate the null to be a small interval hypothesis: $\theta \in \Theta_0 = (\theta_0 - a, \theta_0 + a)$, where a is a very small constant. The Bayesian paradigm can deal with an interval hypothesis more easily. Interval hypotheses can be tested in the CQLIM, CNTSELECT, and SMC procedures. For a more detailed treatment of Bayesian hypothesis testing, see Berger (1985).

Interval Estimation

The Bayesian set estimates are called *credible sets*, which are also known as *credible intervals*. This is analogous to the concept of confidence intervals used in classical statistics. Given the posterior distribution $\pi(\theta|\mathbf{y})$, A is a credible set for θ if

$$P(\theta \in A|\mathbf{y}) = \int_A \pi(\theta|\mathbf{y}) d\theta$$

For example, you can construct a 95% credible set for θ by finding an interval, A , over which $\int_A \pi(\theta|\mathbf{y}) = 0.95$.

You can construct credible sets that have equal tails. A $100(1 - \alpha)\%$ equal-tail interval corresponds to the $100(\alpha/2)$ th and $100(1 - \alpha/2)$ th percentiles of the posterior distribution. Some statisticians prefer this interval because it is invariant under transformations. Another frequently used Bayesian credible set is called the *highest posterior density* (HPD) interval.

A $100(1 - \alpha)\%$ HPD interval is a region that satisfies the following two conditions:

1. The posterior probability of that region is $100(1 - \alpha)\%$.
2. The minimum density of any point within that region is equal to or larger than the density of any point outside that region.

The HPD is an interval in which most of the distribution lies. Some statisticians prefer this interval because it is the smallest interval.

One major distinction between Bayesian and classical sets is their interpretation. The Bayesian probability reflects a person's subjective beliefs. Following this approach, a statistician can make the claim that θ is inside a credible interval with measurable probability. This property is appealing because it enables you to make a direct probability statement about parameters. Many people find this concept to be a more natural way of understanding a probability interval, which is also easier to explain to nonstatisticians. A confidence interval, on the other hand, enables you to make a claim that the interval covers the true parameter. The interpretation reflects the uncertainty in the sampling procedure; a confidence interval of $100(1 - \alpha)\%$ asserts that, in the long run, $100(1 - \alpha)\%$ of the realized confidence intervals cover the true parameter.

Bayesian Analysis: Advantages and Disadvantages

Bayesian methods and classical methods both have advantages and disadvantages, and there are some similarities. When the sample size is large, Bayesian inference often provides results for parametric models that are very similar to the results produced by frequentist methods. Advantages of using Bayesian analysis include the following:

- It provides a natural and principled way of combining prior information with data, within a solid decision theoretical framework. You can incorporate past information about a parameter and form a prior distribution for future analysis. When new observations become available, the previous posterior distribution can be used as a prior. All inferences logically follow from Bayes' theorem.
- It provides inferences that are conditional on the data and are exact, without reliance on asymptotic approximation. Small sample inference proceeds in the same manner as if one had a large sample. Bayesian analysis also can estimate any functions of parameters directly, without using the "plug-in" method (a way to estimate functionals by plugging the estimated parameters in the functionals).
- It obeys the likelihood principle. If two distinct sampling designs yield proportional likelihood functions for θ , then all inferences about θ should be identical from these two designs. Classical inference does not in general obey the likelihood principle.
- It provides interpretable answers, such as "the true parameter θ has a probability of 0.95 of falling in a 95% credible interval."

- It provides a convenient setting for a wide range of models, such as hierarchical models and missing data problems. MCMC, along with other numerical methods, makes computations tractable for virtually all parametric models.

There are also disadvantages to using Bayesian analysis:

- It does not tell you how to select a prior. There is no correct way to choose a prior. Bayesian inferences require skills to translate subjective prior beliefs into a mathematically formulated prior. If you do not proceed with caution, you can generate misleading results.
- It can produce posterior distributions that are heavily influenced by the priors. From a practical point of view, it might sometimes be difficult to convince subject matter experts who do not agree with the validity of the chosen prior.
- It often comes with a high computational cost, especially in models with a large number of parameters. In addition, simulations provide slightly different answers unless the same random seed is used. Note that slight variations in simulation results do not contradict the early claim that Bayesian inferences are exact. The posterior distribution of a parameter is exact, given the likelihood function and the priors, while simulation-based estimates of posterior quantities can vary due to the random number generator used in the procedures.

For more in-depth treatments of the pros and cons of Bayesian analysis, see Berger (1985, Sections 4.1 and 4.12), Berger and Wolpert (1988), Bernardo and Smith (1994, with a new edition coming out), Carlin and Louis (2000, Section 1.4), Robert (2001, Chapter 11), and Wasserman (2004, Section 11.9).

The following sections provide detailed information about the Bayesian methods provided in SAS.

Standard Prior Distributions

Table 2.1 through Table 2.10 show the density functions for all the distributions that you can use as priors in SAS Econometrics procedures. Some procedures support only a subset of these distributions. You can typically specify these distributions as priors for model parameters in a PRIOR statement.

Table 2.1 Beta Distribution

Density	$f(\theta a, b) = \frac{\theta^{a-1}(1-\theta)^{b-1}}{B(a, b)}$
Parameter restriction	$a > 0, \quad b > 0$
Range	$\begin{cases} \theta \in [0, 1] & \text{when } a = 1, b = 1 \\ \theta \in [0, 1] & \text{when } a = 1, b \neq 1 \\ \theta \in (0, 1] & \text{when } a \neq 1, b = 1 \\ \theta \in (0, 1) & \text{otherwise} \end{cases}$
Mean	$\frac{a}{a+b}$
Variance	$\frac{ab}{(a+b)^2(a+b+1)}$

Table 2.1 *continued*

Mode	$\left\{ \begin{array}{ll} \frac{a-1}{a+b-2} & a > 1, b > 1 \\ 0 \text{ and } 1 & a < 1, b < 1 \\ 0 & \left\{ \begin{array}{l} a < 1, b \geq 1 \\ a = 1, b > 1 \end{array} \right. \\ 1 & \left\{ \begin{array}{l} a \geq 1, b < 1 \\ a > 1, b = 1 \end{array} \right. \\ \text{not unique} & a = b = 1 \end{array} \right.$
------	--

Table 2.2 Cauchy Distribution

Density	$f(\theta \mu, \gamma) = \frac{1}{\pi\gamma \left[1 + \left(\frac{\theta-\mu}{\gamma}\right)^2\right]}$
Parameter restriction	$\gamma > 0$
Range	$\theta \in (-\infty, \infty)$
Mean	Not defined
Variance	Not defined
Mode	μ

Table 2.3 Gamma Distribution

Density	$f(\theta a, b) = \frac{1}{b^a \Gamma(a)} \theta^{a-1} e^{-\theta/b}$
Parameter restriction	$a > 0, b > 0$
Range	$\theta \in [0, \infty)$
Mean	ab
Variance	ab^2
Mode	$(a-1)b$

Table 2.4 Inverse Gamma Distribution

Density	$f(\theta a, b) = \frac{b^a}{\Gamma(a)} \theta^{-(a+1)} e^{-b/\theta}$
Parameter restriction	$a > 0, b > 0$
Range	$\theta \in [0, \infty)$
Mean	$\frac{b}{a-1}, \quad a > 1$

Table 2.4 *continued*

Variance	$\frac{b^2}{(a-1)^2(a-2)}, \quad a > 2$
Mode	$\frac{b}{a+1}$

Table 2.5 Lognormal Distribution

Density	$f(\theta \mu, \sigma) = \frac{1}{\theta\sigma\sqrt{2\pi}} \exp\left(-\frac{(\log \theta - \mu)^2}{2\sigma^2}\right)$
Parameter restriction	$\sigma > 0$
Range	$\theta \in (0, \infty)$
Mean	$\exp(\mu + \sigma^2/2)$
Variance	$\exp(2\mu + 2\sigma^2) - \exp(2\mu + \sigma^2)$
Mode	$\exp(\mu - \sigma^2)$

Table 2.6 Normal Distribution

Density	$f(\theta \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\theta - \mu)^2}{2\sigma^2}\right)$
Parameter restriction	$\sigma > 0$
Range	$\theta \in (-\infty, \infty)$
Mean	μ
Variance	σ^2
Mode	μ

Table 2.7 Square Root Gamma Distribution

Density	$f(\theta a, b) = \frac{2}{b^a\Gamma(a)} \theta^{2a-1} e^{-\theta^2/b}$
Parameter restriction	$a > 0, b > 0$
Range	$\theta \in [0, \infty)$
Mean	$\frac{\Gamma(a+\frac{1}{2})}{\Gamma(a)} \sqrt{b}$
Variance	$\left\{ a - \left[\frac{\Gamma(a+\frac{1}{2})}{\Gamma(a)} \right]^2 \right\} b$
Mode	$\sqrt{(a - \frac{1}{2})b}, \quad a \geq \frac{1}{2}$

This is also known as the Nakagami distribution. For more information, see Stacy (1962).

Table 2.8 Square Root Inverse Gamma Distribution

Density	$f(\theta a, b) = \frac{2b^a}{\Gamma(a)} \theta^{-(2a+1)} e^{-b/\theta^2}$
Parameter restriction	$a > 0, b > 0$
Range	$\theta \in [0, \infty)$
Mean	$\frac{\Gamma(a-\frac{1}{2})}{\Gamma(a)} \sqrt{b}, \quad a > \frac{1}{2}$
Variance	$\left\{ \frac{1}{a-1} - \left[\frac{\Gamma(a-\frac{1}{2})}{\Gamma(a)} \right]^2 \right\} b, \quad a > 1$
Mode	$\sqrt{\frac{b}{a+\frac{1}{2}}}$

This is also known as the inverse Nakagami distribution. For more information, see Stacy (1962).

Table 2.9 t Distribution

Density	$f(\theta \mu, \sigma, \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu}\sigma} \left[1 + \frac{1}{\nu} \frac{(\theta-\mu)^2}{\sigma^2} \right]^{-\frac{\nu+1}{2}}$
Parameter restriction	$\sigma > 0, \nu > 0$
Range	$\theta \in (-\infty, \infty)$
Mean	$\mu, \text{ for } \nu > 1$
Variance	$\frac{\nu}{\nu-2} \sigma^2, \text{ for } \nu > 2$
Mode	μ

Table 2.10 Uniform Distribution

Density	$f(\theta m, M) = \frac{1}{M-m}$
Parameter restriction	$-\infty < m < M < \infty$
Range	$\theta \in [m, M]$
Mean	$\frac{m+M}{2}$
Variance	$\frac{(M-m)^2}{12}$
Mode	Not unique

Some procedures allow for improper uniform priors, where $m = -\infty$ or $M = \infty$. For more information, see the section “[Improper Priors](#)” on page 8.

Markov Chain Monte Carlo Method

The Markov chain Monte Carlo (MCMC) method is a general simulation method for sampling from posterior distributions and computing posterior quantities of interest. MCMC methods sample successively from a target distribution. Each sample depends on the previous one, hence the notion of the Markov chain. A Markov chain is a sequence of random variables, $\theta^1, \theta^2, \dots$, for which the random variable θ^t depends on all previous θ s only through its immediate predecessor θ^{t-1} . You can think of a Markov chain applied to sampling as a mechanism that traverses randomly through a target distribution without having any memory of where it has been. Where it moves next is entirely dependent on where it is now.

Monte Carlo, as in Monte Carlo integration, is mainly used to approximate an expectation by using the Markov chain samples. In the simplest version

$$\int_S g(\theta) p(\theta) d\theta \cong \frac{1}{n} \sum_{t=1}^n g(\theta^t)$$

where $g(\cdot)$ is a function of interest and θ^t are samples from $p(\theta)$ on its support S . This approximates the expected value of $g(\theta)$. The earliest reference to MCMC simulation occurs in the physics literature. Metropolis and Ulam (1949) and Metropolis et al. (1953) describe what is known as the Metropolis algorithm (see the section “[Metropolis and Metropolis-Hastings Algorithms](#)” on page 17). The algorithm can be used to generate sequences of samples from the joint distribution of multiple variables, and it is the foundation of MCMC. Hastings (1970) generalized their work, resulting in the Metropolis-Hastings algorithm. Geman and Geman (1984) analyzed image data by using what is now called Gibbs sampling (see the section “[Gibbs Sampler](#)” on page 18). These MCMC methods first appeared in the mainstream statistical literature in Tanner and Wong (1987).

The Markov chain method has been quite successful in modern Bayesian computing. Only in the simplest Bayesian models can you recognize the analytical forms of the posterior distributions and summarize inferences directly. In moderately complex models, posterior densities are too difficult to work with directly. With the MCMC method, it is possible to generate samples from an arbitrary posterior density, $\pi(\theta|\mathbf{y})$, and to use these samples to approximate expectations of quantities of interest. Several other aspects of the Markov chain method also contributed to its success. Most importantly, if the simulation algorithm is implemented correctly, the Markov chain is guaranteed to converge to the target distribution $\pi(\theta|\mathbf{y})$ under rather broad conditions, regardless of where the chain was initialized. In other words, a Markov chain is able to improve its approximation to the true distribution at each step in the simulation. Furthermore, if the chain is run for a very long time (often required), you can recover $\pi(\theta|\mathbf{y})$ to any precision. Also, the simulation algorithm is easily extensible to models with a large number of parameters or high complexity, although the “curse of dimensionality” often causes problems in practice.

Properties of Markov chains are discussed in Feller (1968), Breiman (1968), and Meyn and Tweedie (1993). Ross (1997) and Karlin and Taylor (1975) give a non-measure-theoretic treatment of stochastic processes, including Markov chains. For conditions that govern Markov chain convergence and rates of convergence, see Amit (1991), Applegate, Kannan, and Polson (1990), Chan (1993), Geman and Geman (1984), Liu, Wong, and Kong (1991a, b), Rosenthal (1991a, b), Tierney (1994), and Schervish and Carlin (1992). Besag (1974) describes conditions under which a set of conditional distributions gives a unique joint distribution. Tanner (1993), Gilks, Richardson, and Spiegelhalter (1996), Chen, Shao, and Ibrahim (2000), Liu (2001), Gelman et al. (2004), Robert and Casella (2004), and Congdon (2001, 2003, 2005) provide both theoretical

and applied treatments of MCMC methods. You can also see the section “A Bayesian Reading List” on page 39 for a list of books with varying levels of difficulty of treatment of the subject and its application to Bayesian statistics.

Metropolis and Metropolis-Hastings Algorithms

The Metropolis algorithm is named after its inventor, the American physicist and computer scientist Nicholas C. Metropolis. The algorithm is simple but practical, and it can be used to obtain random samples from any arbitrarily complicated target distribution of any dimension that is known up to a normalizing constant.

Suppose you want to obtain T samples from a univariate distribution with the probability density function $\pi(\theta|y)$. Suppose θ^t is the t th sample from f . To use the Metropolis algorithm, you need to have an initial value θ^0 and a symmetric *proposal* density $q(\theta^{\text{prop}}|\theta^t)$. For the $(t + 1)$ iteration, the algorithm generates a proposal sample θ^{prop} from $q(\cdot|\cdot)$ that is based on the current sample θ^t , and it makes a decision to either accept or reject the new sample. If the new sample is accepted, the algorithm repeats itself by starting at the new sample. If the new sample is rejected, the algorithm starts at the current point and repeats. The algorithm is self-repeating, so it can be carried out as long as required. In practice, you have to decide the total number of samples needed in advance and stop the sampler after that many iterations have been completed.

Suppose that $q(\theta^{\text{prop}}|\theta^t)$ is a symmetric distribution. The proposal distribution should be an easy distribution from which to sample, and it must be such that $q(\theta^{\text{prop}}|\theta^t) = q(\theta^t|\theta^{\text{prop}})$, meaning that the likelihood of jumping to θ^{prop} from θ^t is the same as the likelihood of jumping back to θ^t from θ^{prop} . The most common choice of the proposal distribution is the normal distribution $N(\theta^t, \sigma)$ with a fixed σ . This is called a random walk Metropolis algorithm (RWM) and can be summarized as follows:

1. Set $t = 0$. Choose a starting point θ^0 . This can be an arbitrary point as long as $\pi(\theta^0|y) > 0$.
2. Generate a new sample, θ^{prop} , by using the proposal distribution $q(\cdot|\theta^t)$.
3. Calculate the following quantity:

$$r = \min \left\{ \frac{\pi(\theta^{\text{prop}}|y)}{\pi(\theta^t|y)}, 1 \right\}$$

4. Sample u from the uniform distribution $U(0, 1)$.
5. Set $\theta^{t+1} = \theta^{\text{prop}}$ if $u < r$; otherwise set $\theta^{t+1} = \theta^t$.
6. Set $t = t + 1$. If $t < n$, the number of desired samples, return to step 2. Otherwise, stop.

Note that the iteration number keeps increasing regardless of whether a proposed sample is accepted.

This algorithm defines a chain of random variates whose distribution will converge to the desired distribution, $\pi(\theta|y)$, and so from some point forward, the chain of samples is a sample from the distribution of interest. In Markov chain terminology, this distribution is called the *stationary distribution* of the chain, and in Bayesian statistics, it is the posterior distribution of the model parameters. The reason that the Metropolis algorithm works is beyond the scope of this documentation, but you can find more detailed descriptions and proofs in many standard textbooks, including Roberts (1996) and Liu (2001). The random walk Metropolis algorithm is used by all SAS Econometrics procedures that provide Bayesian inference.

You are not limited to a symmetric random walk proposal distribution in establishing a valid sampling algorithm. A more general form, the Metropolis-Hastings (MH) algorithm, was proposed by Hastings (1970). The MH algorithm uses an asymmetric proposal distribution: $q(\theta^{\text{prop}}|\theta^t) \neq q(\theta^t|\theta^{\text{prop}})$. The difference in its implementation comes in calculating the ratio of densities:

$$r = \min \left\{ \frac{\pi(\theta^{\text{prop}}|\mathbf{y})q(\theta^t|\theta^{\text{prop}})}{\pi(\theta^t|\mathbf{y})q(\theta^{\text{prop}}|\theta^t)}, 1 \right\}$$

Other steps remain the same.

The extension of the Metropolis algorithm to a higher-dimensional θ is straightforward. Suppose that $\theta = (\theta_1, \theta_2, \dots, \theta_k)'$ is the parameter vector. To start the Metropolis algorithm, select an initial value for each θ_k and use a multivariate version of proposal distribution $q(\cdot|\cdot)$, such as a multivariate normal distribution, to select a k -dimensional new parameter. Other steps remain the same as those previously described, and this Markov chain eventually converges to the target distribution of $\pi(\theta|\mathbf{y})$. Chib and Greenberg (1995) provide a useful tutorial on the algorithm.

Gibbs Sampler

The Gibbs sampler, named by Geman and Geman (1984) after the American physicist Josiah W. Gibbs, is a special case of the “[Metropolis and Metropolis-Hastings Algorithms](#)” on page 17 in which the proposal distributions exactly match the posterior conditional distributions and proposals are accepted 100% of the time. Gibbs sampling requires you to decompose the joint posterior distribution into full conditional distributions for each parameter in the model and then sample from them. The sampler can be efficient when the parameters are not highly dependent on each other and the full conditional distributions are easy to sample from. Some researchers favor this algorithm because it does not require an instrumental proposal distribution as Metropolis methods do. However, while deriving the conditional distributions can be relatively easy, it is not always possible to find an efficient way to sample from these conditional distributions.

Suppose that $\theta = (\theta_1, \dots, \theta_k)'$ is the parameter vector, $\pi(\mathbf{y}|\theta)$ is the likelihood, and $\pi(\theta)$ is the prior distribution. The full posterior conditional distribution of $\pi(\theta_i|\theta_j, i \neq j, \mathbf{y})$ is proportional to the joint posterior density; that is,

$$\pi(\theta_i|\theta_j, i \neq j, \mathbf{y}) \propto \pi(\mathbf{y}|\theta)\pi(\theta)$$

For instance, the one-dimensional conditional distribution of θ_1 given $\theta_j = \theta_j^*, 2 \leq j \leq k$, is computed as the following:

$$\pi(\theta_1|\theta_j = \theta_j^*, 2 \leq j \leq k, \mathbf{y}) = \pi(\mathbf{y}|\theta = (\theta_1, \theta_2^*, \dots, \theta_k^*)')\pi(\theta = (\theta_1, \theta_2^*, \dots, \theta_k^*)')$$

The Gibbs sampler works as follows:

1. Set $t = 0$, and choose an arbitrary initial value of $\theta^0 = \{\theta_1^0, \dots, \theta_k^0\}$.
2. Generate each component of θ as follows:

- Draw θ_1^{t+1} from $\pi(\theta_1|\theta_2^t, \dots, \theta_k^t, \mathbf{y})$
 - Draw θ_2^{t+1} from $\pi(\theta_2|\theta_1^{t+1}, \theta_3^t, \dots, \theta_k^t, \mathbf{y})$
 - $\vdots$
 - Draw θ_k^{t+1} from $\pi(\theta_k|\theta_1^{t+1}, \dots, \theta_{k-1}^{t+1}, \mathbf{y})$
3. Set $t = t + 1$. If $t < T$, the number of desired samples, return to step 2. Otherwise, stop.

The Gibbs sampler can draw more than one parameter simultaneously in each step. The grouped parameters are called blocks, and each parameter must belong to exactly one block. A block that contains only one parameter is called a scalar block. The blocks are drawn one at a time, but parameters in the same block are drawn from their joint full conditional distribution. For example, if θ_1 and θ_2 are in the same block, then they are drawn from $\pi(\theta_1, \theta_2|\theta_3^t, \dots, \theta_k^t, \mathbf{y})$. To maximize the Gibbs sampler's efficiency, parameters with high posterior correlation should be put in the same block.

Often not all the full conditional distributions in a Gibbs sampler are tractable, or easy to draw a random sample from. A common strategy for overcoming this problem is to use a Metropolis step to draw from the troublesome full conditionals, applying the ideas of the section “[Metropolis and Metropolis-Hastings Algorithms](#)” on page 17 to a conditional posterior. This is known as Metropolis within Gibbs, and it is valid even if every step of the Gibbs sampler is a Metropolis step.

The name “Gibbs” was introduced by Geman and Geman (1984). Gelfand et al. (1990) first used Gibbs sampling to solve problems in Bayesian inference. See Casella and George (1992) for a tutorial on the sampler.

Tuning Random Walk Metropolis Algorithms

The efficiency of random walk Metropolis algorithms depends crucially on the proposal distribution. Finding a good proposal distribution for each block of parameters is called tuning. The proposal distribution must be tuned during a tuning phase before MCMC iterations are collected for posterior inference so that the tuning process does not bias posterior inference. Collectively, the tuning and burn-in phases are called the warmup phase. This section describes the tuning algorithm that is used for all RWM samplers employed in SAS Econometrics procedures.

A good proposal distribution should resemble the actual posterior distribution of the parameters. Large-sample theory states that the posterior distribution of the parameters approaches a multivariate normal distribution with a covariance matrix equal to the inverse of the negative Hessian of the log posterior at the posterior mode (Gelman et al. 2004, Appendix B; Schervish 1995, Section 7.4). That is why a normal proposal distribution often works well in practice. The proposal distribution for random walk Metropolis in SAS Econometrics procedures is always the normal distribution. It is generated in the following fashion:

$$\theta^{\text{prop}} \sim N(\theta^{\text{cur}}, c^{\text{cur}} \text{COV}^{\text{cur}})$$

where θ^{prop} is the proposed parameter vector, θ^{cur} is the previous value of the parameters in the Markov chain, c^{cur} is a scale parameter that is tuned in the tuning phase of the algorithm, and COV^{cur} is an approximation of the posterior covariance that is also tuned in the tuning phase of the algorithm. When there are multiple parameter blocks—that is, in a [Metropolis within Gibbs](#) scheme—each block is tuned separately, each with its own c^{cur} and COV^{cur} . In particular for a scalar block, COV^{cur} is also a scalar—an approximation to the posterior variance of the only parameter in that block.

The tuning phase consists of n_{stage} tuning stages, each n_{tune} iterations long. Between tuning stages, two sorts of adjustments to the proposal distribution are made: adjustments to the scale parameter c^{cur} and adjustments to the proposal covariance matrix COV^{cur} .

Scale Tuning

The acceptance rate of a Metropolis chain is closely related to its sampling efficiency. For a random walk Metropolis algorithm, a high acceptance rate means that most new samples occur right around the current parameter value. Their frequent acceptance means that the Markov chain is moving rather slowly and not fully exploring the parameter space. A low acceptance rate means that the proposed samples are often rejected; hence the chain is not moving much. An efficient random walk Metropolis sampler has an acceptance rate that is neither too high nor too low. The scale parameter c^{cur} in the proposal distribution effectively controls this acceptance probability. Roberts, Gelman, and Gilks (1997) show that if both the target density and the proposal density are normal, the optimal acceptance probability for the Markov chain should be around 0.45 in a one-dimensional problem and should asymptotically approach 0.234 in higher-dimensional problems. The corresponding optimal scale is 2.38.

Because of the nature of stochastic simulations, it is impossible to fine-tune a set of variables so that the Metropolis chain has exactly the target acceptance rate that you want. In addition, Roberts and Rosenthal (2001) empirically demonstrate that an acceptance rate between 0.15 and 0.5 is at least 80% efficient, so there is really no need to fine-tune the algorithms to reach an acceptance probability that is within a small tolerance of the optimal values. SAS Econometrics procedures tune the scale parameter so that the observed acceptance rate falls within some tolerance of the target acceptance rate. Let p^{obs} denote the observed acceptance rate in a given tuning stage, let δ denote the target acceptance rate, and let ϵ denote the tolerance. Then if $p^{\text{obs}} < \delta - \epsilon$, the value of c^{cur} is decreased. On the other hand, if $p^{\text{obs}} > \delta + \epsilon$, then the value of c^{cur} is increased. Typically $\epsilon = 0.075$ is a good choice, $\delta = 0.45$ is a good choice for blocks of parameters larger than three or four dimensions, and $\delta = 0.234$ is a good choice for scalar blocks.

When c^{cur} is adjusted between tuning stages, it is done using the following scheme:¹

$$c^{\text{new}} = \frac{c^{\text{cur}} \cdot \Phi^{-1}(\delta/2)}{\Phi^{-1}(p^{\text{obs}}/2)}$$

where c^{cur} is the scale from the previous tuning stage; c^{new} is the new scale for the next stage or any further sampling; p^{obs} is the observed acceptance rate in the previous tuning stage; and δ is the target acceptance rate. A good choice for the initial value of c^{cur} is 2.38 because it is optimal for a normal model with a normal proposal.

Covariance Tuning

When the proposal scale parameter is adjusted between stages, the proposal covariance matrix is also adjusted. To do this, SAS Econometrics procedures take a weighted average of the previously used covariance matrix and the observed covariance matrix in the last tuning stage. The formula to update the covariance matrix is

$$\text{COV}^{\text{new}} = w \text{COV}^{\text{obs}} + (1 - w) \text{COV}^{\text{cur}}$$

¹ Roberts, Gelman, and Gilks (1997) and Roberts and Rosenthal (2001) demonstrate that the relationship between acceptance probability and scale in a random walk Metropolis scheme is $p = 2\Phi(-\sqrt{I}c/2)$, where c is the scale; p is the acceptance rate; Φ is the cumulative distribution function of a standard normal distribution; and $I \equiv E_f[(f'(x)/f(x))^2]$, where $f(x)$ is the density function of samples. This relationship determines the updating scheme, with I replaced by the identity matrix to simplify calculation.

where w is the tuning weight, between 0 and 1; COV^{cur} is the covariance matrix that was used in the previous tuning stage; COV^{obs} is the covariance matrix that was observed in the previous tuning stage; and COV^{new} is the new covariance matrix that will be used for the next tuning stage or any further sampling. Larger weights cause the sampler to adapt the proposal covariance matrix faster during the tuning process, but if this value is too large, the sampler might never settle into a good choice. Typically, a good tuning weight is $w = 0.75$. If the initial value of COV^{cur} is very different from the posterior covariance matrix, many tuning stages might be required before the proposal distribution becomes efficient, because the Markov chain needs to spend time learning about the posterior covariance structure. If the proposal distribution is too different from the posterior, then the sampler can reject 100% of the time and tuning will not work. A good default choice for the initial value of the proposal covariance matrix, COV^{cur} , is the covariance matrix of the maximum likelihood estimates of the parameters, or the negative inverse Hessian of the log posterior density, if either is available.

Hamiltonian Monte Carlo Sampler

The Hamiltonian Monte Carlo (HMC) sampler, also known as the hybrid Monte Carlo sampler, is a version of the Metropolis algorithm that uses gradient information and auxiliary momentum variables to construct a Markov chain that converges to the posterior distribution (Neal 2011; Hoffman and Gelman 2014; Betancourt 2018). The algorithm uses Hamiltonian dynamics to construct distant Metropolis proposals, allowing it to converge quickly and to be efficient in many scenarios. HMC is applicable only to continuous unconstrained parameters, although constrained parameters can be transformed prior to sampling.

Suppose that $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots, \theta_k)$ is the unknown parameter vector with the posterior distribution $\pi(\boldsymbol{\theta} | \mathbf{y})$, also known as the position vector. To define Hamiltonian dynamics, first define an auxiliary momentum vector $\mathbf{r} = (r_1, r_2, \dots, r_k)$ with probability density $q(\mathbf{r})$. Assume that $\mathbf{r} \sim N(\mathbf{0}, \mathbf{M})$, where $\mathbf{M}$ is also known as the mass matrix. Next, define potential energy as $V(\boldsymbol{\theta}) = -\log \pi(\boldsymbol{\theta} | \mathbf{y})$; kinetic energy as $T(\mathbf{r}) = -\log \pi(\mathbf{r}) = \text{constant} + \frac{1}{2}\mathbf{r}'\mathbf{M}^{-1}\mathbf{r}$; and total energy, also known as the Hamiltonian, as $H(\boldsymbol{\theta}, \mathbf{r}) = V(\boldsymbol{\theta}) + T(\mathbf{r})$. Finally, Hamilton's equations define the evolution of position and momentum in time as a physical system, given by

$$\frac{d\mathbf{r}}{d\tau} = -\frac{\partial H}{\partial \boldsymbol{\theta}} \quad \text{and} \quad \frac{d\boldsymbol{\theta}}{d\tau} = \frac{\partial H}{\partial \mathbf{r}}$$

where τ represents time in the system. A solution to these equations is called a Hamiltonian flow, or a Hamiltonian trajectory.

An idealized version of HMC samples $\boldsymbol{\theta}^{t+1}$ conditional on $\boldsymbol{\theta}^t$ as follows:

1. Sample an initial momentum, $\mathbf{r}' \sim N(\mathbf{0}, \mathbf{M})$.
2. Run the Hamiltonian flow from the initial position-momentum pair $(\boldsymbol{\theta}^0, \mathbf{r}^0) = (\boldsymbol{\theta}^t, \mathbf{r}')$, until some integration time $\mathcal{T}$ is reached, to obtain $(\boldsymbol{\theta}^{\mathcal{T}}, \mathbf{r}^{\mathcal{T}})$.
3. Set the next iteration of the Markov chain to the endpoint of the Hamiltonian flow, $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{\mathcal{T}}$.

Hamiltonian flow conserves both total energy and volume of the system, and as a result, this Markov chain converges to the posterior distribution.

In practice, this algorithm cannot be used, because Hamilton's equations cannot be solved analytically, and instead they must be numerically integrated using leapfrog integration. Define a step size $\epsilon > 0$; let $l = 1, 2, \dots, L$ index leapfrog steps; and let $(\boldsymbol{\theta}^l, \mathbf{r}^l)$ denote the position-momentum pair at leapfrog step l ,

with the initial position-momentum pair $(\boldsymbol{\theta}^0, \mathbf{r}^0)$. Then a leapfrog step obtains $(\boldsymbol{\theta}^{l+1}, \mathbf{r}^{l+1})$ from $(\boldsymbol{\theta}^l, \mathbf{r}^l)$ via the following three steps:

$$\begin{aligned}\mathbf{r}^{l+1/2} &= \mathbf{r}^l - \frac{\epsilon}{2} \nabla V(\boldsymbol{\theta}^l) \\ \boldsymbol{\theta}^{l+1} &= \boldsymbol{\theta}^l + \epsilon \mathbf{M}^{-1} \mathbf{r}^{l+1/2} \\ \mathbf{r}^{l+1} &= \mathbf{r}^{l+1/2} - \frac{\epsilon}{2} \nabla V(\boldsymbol{\theta}^{l+1})\end{aligned}$$

where $\nabla V(\boldsymbol{\theta})$ is the gradient of the potential energy with respect to $\boldsymbol{\theta}$. The number of leapfrog steps is chosen on the basis of the desired integration time via $L = \lfloor \mathcal{T}/\epsilon \rfloor$, where $\lfloor x \rfloor$ rounds x down to the nearest integer.

Leapfrog integration exactly conserves volume but only approximately conserves energy, so in practice HMC uses a Metropolis correction to account for the error. To ensure that the Markov transition is reversible, you must reverse the sign of the momentum vector so that running the Hamiltonian flow from the endpoint returns it to the starting point. Accounting for all of this yields the HMC algorithm, which obtains iteration $\boldsymbol{\theta}^{t+1}$ from iteration $\boldsymbol{\theta}^t$ as follows:

1. Sample an initial momentum, $\mathbf{r}' \sim \mathcal{N}(\mathbf{0}, \mathbf{M})$.
2. Take $L = \lfloor \mathcal{T}/\epsilon \rfloor$ leapfrog steps from the initial position-momentum pair $(\boldsymbol{\theta}^0, \mathbf{r}^0) = (\boldsymbol{\theta}^t, \mathbf{r}')$ to obtain $(\boldsymbol{\theta}^L, \mathbf{r}^L)$.
3. Set $\boldsymbol{\theta}^{\text{prop}} = \boldsymbol{\theta}^L$ and $\mathbf{r}^{\text{prop}} = -\mathbf{r}^L$ and compute the Metropolis acceptance probability:

$$\begin{aligned}r &= \min \left\{ \exp \left[-H(\boldsymbol{\theta}^{\text{prop}}, \mathbf{r}^{\text{prop}}) + H(\boldsymbol{\theta}^0, \mathbf{r}^0) \right], 1 \right\} \\ &= \min \left\{ \exp \left[-H(\boldsymbol{\theta}^L, -\mathbf{r}^L) + H(\boldsymbol{\theta}^0, \mathbf{r}^0) \right], 1 \right\}\end{aligned}$$

4. With probability, r set $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{\text{prop}}$; otherwise set $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t$.

No-U-Turn Sampler

HMC can rapidly converge to the posterior distribution and can provide nearly iid draws, but it relies heavily on the careful choice of tuning parameters: the mass matrix $\mathbf{M}$, the step size $\epsilon > 0$, and the integration time $\mathcal{T} > 0$. The No-U-Turn Sampler (NUTS; Hoffman and Gelman 2014) solves part of this problem.

NUTS chooses the integration time dynamically—that is, based on the current position-momentum pair—and this is where the name comes from. To do this, it builds two Hamiltonian trajectories: a forward trajectory that moves forward from the initial position-momentum pair, and a backward trajectory that moves backward from the initial position-momentum pair. NUTS builds these trajectories recursively by using a binary tree as follows. Let $j = 0, 1, \dots, J_{\max}$ index the current height of the tree, where $J_{\max} > 0$ is the maximum height allowed.

1. Start at $j = 0$.
2. Flip a coin. If the result is heads, add 2^j leapfrog steps to the end of the forward trajectory. If the result is tails, add 2^j leapfrog steps to the end of the backward trajectory.
3. Set $j = j + 1$.

Then NUTS chooses to stop leapfrog integration when the trajectories start to make a “U-turn”—that is, when the endpoints of the two trajectories begin to move closer together.

Additionally, NUTS proposes all points in both the forward and backward trajectories simultaneously in the Metropolis step. If the step accepts, then NUTS uses a slice sampler to choose a single point from the combined trajectories. See Hoffman and Gelman (2014) for the details of how this works.

As the height of the tree increases, the computational cost of the iteration increases dramatically. As a result, NUTS caps the height of the tree at a maximum height of $J_{\max} > 0$. When an iteration reaches the maximum tree height, NUTS prematurely stops integration and proposes the current available forward and backward trajectories even if it has not made a U-turn. When NUTS reaches the maximum tree height for multiple iterations, this increases autocorrelation in the chain and leads to worse mixing. This is mainly a problem during sampling and not during warmup (tuning and burn-in). You can increase the maximum tree height to reduce these problems at the cost of increased computation per iteration. You can also force NUTS to cancel sampling if the maximum tree height is reached too many times.

The CQLIM and CNTSELECT procedures give you the option to use an implementation of the NUTS algorithm to draw the posterior samples.

Tuning for NUTS

The final two tuning parameters in HMC are the leapfrog integrator step size and the mass matrix. These can both be tuned during a tuning phase. NUTS tunes the step size throughout this tuning phase to achieve a target Metropolis acceptance rate of $0 < \delta < 1$ (usually set to 0.8), using a dual averaging optimization scheme. Smaller step sizes yield higher acceptance rates and a more robust algorithm with a higher computational cost per Markov chain iteration, whereas the opposite occurs for larger step sizes. So the target acceptance rate is chosen to balance computational cost and robustness, with larger values resulting in a more robust and more computationally costly NUTS algorithm.

NUTS does not tune the mass matrix, although it can be combined with a variety of mass matrix tuning algorithms. Typically the mass matrix is constant; this is known as Euclidean HMC. The alternative is to allow the mass matrix to depend on θ ; this is known as Riemannian HMC. Theory suggests that the ideal mass matrix in Euclidean HMC is the inverse posterior covariance matrix (Betancourt 2018). A diagonal mass matrix often yields a more efficient HMC or NUTS algorithm, but at the cost of robustness. In that case, the ideal mass matrix is the diagonal matrix of inverse posterior variances. In general, the posterior variances are unknown prior to sampling, so they must be estimated on the fly during an initial tuning phase.

The CQLIM and CNTSELECT procedures use a constant diagonal mass matrix for NUTS, and they have the ability to use a flexible class of tuning algorithms, known as expanding-window tuning algorithms, to learn the posterior variances. Let $n_{\text{tune}} \geq 0$ denote the number of tuning iterations available in the tuning phase. Next, divide the tuning phase into K windows. Let $k = 1, 2, \dots, K$ index windows, and let $n_k \geq 0$ denote the number of iterations that are apportioned to window k . Then $n_{\text{tune}} = \sum_{k=1}^K n_k$. The step size is tuned through each of these tuning windows, with no change to how it is tuned between windows, but the windows determine how to tune the mass matrix.

In the first tuning window, *only* the step size is tuned, and the initial mass matrix is used for sampling throughout. This allows the Markov chain to converge to the posterior distribution before the tuning algorithm attempts to estimate the posterior variances. Similarly, only the step size is tuned in the final window, and the mass matrix that is arrived at after tuning window n_{K-1} is used for sampling in the final tuning window, as well as after the tuning phase. This allows the step size to be adapted to the final mass matrix. Both the initial window size, n_1 , and the final window size, n_K , can be adjusted, including setting them both to 0 to tune the mass matrix throughout the tuning phase. Typically, both are set to about 50 or 100.

How the tuning algorithm behaves in the middle tuning windows, and the number and sizes of those windows, depends on the particular settings of the tuning algorithm. There are two more parameters that control this behavior: n_2 , the size of the first middle tuning window; and $\gamma > 0$, which controls how quickly the size of the windows expands. In particular, $n_3 = \lfloor \gamma n_2 \rfloor$, $n_4 = \lfloor \gamma n_3 \rfloor$, and so on, where $\lfloor x \rfloor$ rounds x to the nearest integer from below. The number of tuning windows is not directly controllable; instead the total number of tuning iterations n_{tune} indirectly controls the number of tuning windows. After the initial and final windows, there are $n_{\text{tune}} - n_1 - n_K$ tuning iterations to distribute among the middle tuning windows. The total number of tuning windows, K , is maximized subject to the constraint that the sum of the tuning window sizes be no larger than the total number of tuning iterations; that is,

$$n_1 + n_K + \sum_{k=2}^{K-1} \lfloor \gamma^{k-2} n_2 \rfloor \leq n_{\text{tune}}$$

This procedure does not quite use all available tuning iterations, so any remaining iterations are used in the last middle tuning window, with the result that

$$n_{K-1} = n_{\text{tune}} - n_1 - n_K - \sum_{k=2}^{K-2} \lfloor \gamma^{k-2} n_2 \rfloor$$

For example, if $n_{\text{tune}} = 2,000$, $n_1 = 75$, $n_K = 50$, $n_2 = 25$, and $\gamma = 2$, so that the size of each middle window is double that of the previous middle window, then you have $K = 8$ with the naive tuning window sizes $n_3 = 50$, $n_4 = 100$, $n_5 = 200$, $n_6 = 400$, and $n_7 = 800$. This accounts for 1,700 iterations because

$$75 + 25 + 50 + 100 + 200 + 400 + 800 + 50 = 1,700$$

Then the remaining 300 iterations are used by the final window, so that instead $n_7 = 1,100$.

When the windows are defined, the mass matrix is tuned as follows. Let $\mathbf{M}_k$ denote the mass matrix that is used for sampling in tuning window k . $\mathbf{M}_1$ is the initial mass matrix, and $\mathbf{M}_2 = \mathbf{M}_1$ because no mass matrix tuning is performed in window 1. Then $\mathbf{M}_3$ is the diagonal matrix of inverse sample variances from window 2. Similarly, $\mathbf{M}_4$ is the diagonal matrix of inverse sample variances from window 3, and so on, until $\mathbf{M}_K$ is obtained. Then $\mathbf{M}_K$ is the mass matrix that is used during the final tuning stage, as well as for the rest of posterior sampling.

More formally, let m_{ii}^{k+1} denote the i th diagonal element of $\mathbf{M}_{k+1}$, let t index iterations in the Markov chain, and let θ_i^t denote the t th iteration of element i of $\boldsymbol{\theta}$ in the Markov chain. Then for $k = 2, 4, \dots, K-1$,

$$\frac{1}{m_{ii}^{k+1}} = \frac{1}{n_k} \sum_{t=s_{k-1}+1}^{s_k} \left(\theta_i^t - \bar{\theta}_i^k \right)^2$$

where

$$\bar{\theta}_i^k = \frac{1}{n_k} \sum_{t=s_{k-1}+1}^{s_k} \theta_i^t$$

and $s_k = \sum_{j=1}^k n_j$.

Additionally, expanding window mass matrix tuning can use the previous *two* windows to define the mass matrix for the current window, making it an overlapping expanding-window mass matrix tuning algorithm. In that case, the preceding formula still applies for $k = 2$, but for $k = 3, 4, \dots, K-1$ you have

$$\frac{1}{m_{ii}^{k+1}} = \frac{1}{n_{k-1} + n_k} \sum_{t=s_{k-2}+1}^{s_k} \left(\theta_i^t - \bar{\theta}_i^k \right)^2$$

where

$$\bar{\theta}_i^k = \frac{1}{n_{k-1} + n_k} \sum_{t=s_{k-2}+1}^{s_k} \theta_i^t$$

The expanding-window mass matrix tuning algorithm is quite flexible and can accommodate a wide variety of behaviors through the control of its tuning parameters: the window expansion rate, $\gamma \geq 1$; the initial window size, $n_1 \geq 0$; the final window size, $n_K \geq 0$; the second window size, $n_2 \geq 2$; the choice to use overlapping or nonoverlapping windows; and the total number of tuning iterations, $n_{\text{tune}} \geq 0$. For example, if $n_1 = n_K = 0$, then the mass matrix is tuned throughout the entire tuning phase, and if $\gamma = 1$, then each window is the same size. Despite this flexibility, the performance of NUTS can be dramatically affected by the settings of these parameters. With the exception of n_{tune} , it is generally difficult to choose good values for these parameters, so you should leave them at their default settings if you do not have experience with mass matrix tuning algorithms. In difficult problems where NUTS is not performing well, increasing the value of n_{tune} enables the tuning procedure to obtain better estimates of the posterior variances to use in the mass matrix, and this often improves performance.

Parameter Transformation

Often several parameters of a model are constrained. For example, a standard deviation parameter must be positive, and a correlation parameter must be between -1 and 1 . In these cases, it can be more efficient to transform the constrained parameter to an unconstrained space for sampling, depending on the sampling algorithm that you use. Then, after sampling, you must transform the parameters back to the original, constrained space. There is an additional computational cost associated with transforming and back-transforming, so this technique is typically most advantageous when the posterior mass is concentrated near the boundary. Let the original posterior distribution be denoted by

$$\pi(\theta|\mathbf{y})$$

where $\theta = (\theta_1, \dots, \theta_k)'$ is the parameter vector associated with the model and $\mathbf{y}$ represents the data. Let $\mathbf{u} = T(\theta)$ denote the transformation, where T is an invertible and differentiable function, and $\mathbf{u} = (u_1, \dots, u_k)'$ is the transformed parameter vector. Then the posterior distribution of $\mathbf{u}$ given $\mathbf{y}$ is given by

$$\pi(\mathbf{u}|\mathbf{y}) = \pi(T^{-1}(\mathbf{u})|\mathbf{y})|\mathbf{J}|$$

where $\theta = T^{-1}(\mathbf{u})$ is the inverse transformation and $|\mathbf{J}|$ is the absolute value of the determinant of the Jacobian matrix given by

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \theta_1}{\partial u_1} & \cdots & \frac{\partial \theta_1}{\partial u_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial \theta_k}{\partial u_1} & \cdots & \frac{\partial \theta_k}{\partial u_k} \end{bmatrix}$$

When a subset of the parameters are transformed independently of the other parameters, their Jacobians can be computed separately and their determinants can be multiplied because the Jacobian of transformation of the full parameter vector is block diagonal. In general, if there are m independent transformations, $\mathbf{u}_1 = T_1(\theta_1), \dots, \mathbf{u}_m = T_m(\theta_m)$, where $\theta_1, \dots, \theta_m$ are m distinct subsets of the parameter vector θ , then the Jacobian of $\mathbf{u} = T(\theta) = (T_1(\theta_1), \dots, T_m(\theta_m))$, denoted by $\mathbf{J}$, is given by

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 & & \\ & \ddots & \\ & & \mathbf{J}_m \end{bmatrix}$$

where $\mathbf{J}_i$ is the Jacobian of the transformation T_i and the absolute determinant of $\mathbf{J}$ is given by

$$|\mathbf{J}| = \prod_{i=1}^m |\mathbf{J}_i|$$

In the extreme case in which each parameter is transformed independently of all the other parameters, the determinant of the Jacobian has the form

$$|\mathbf{J}| = \left| \prod_{i=1}^k \frac{\partial \theta_i}{\partial u_i} \right| = \left| \prod_{i=1}^k \frac{\partial T_i^{-1}}{\partial u_i} \right|$$

The gradient of the log posterior is additionally needed for HMC and NUTS:

$$\frac{\partial \log \pi(\mathbf{u}|\mathbf{y})}{\partial \mathbf{u}} = \frac{\partial \log \pi(\boldsymbol{\theta}|\mathbf{y})}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{\theta}}{\partial \mathbf{u}} + \frac{\partial \log |\mathbf{J}|}{\partial \mathbf{u}}$$

In the extreme case where each parameter is transformed independently of all the other parameters, these derivatives can be written as

$$\frac{\partial \log \pi(\mathbf{u}|\mathbf{y})}{\partial u_i} = \frac{\partial \log \pi(\boldsymbol{\theta}|\mathbf{y})}{\partial \theta_i} \frac{\partial \theta_i}{\partial u_i} + \frac{\partial}{\partial u_i} \log \left| \frac{\partial \theta_i}{\partial u_i} \right|$$

The CQLIM and CNTSELECT procedures enable you to transform the parameters before using the random walk Metropolis sampler. Additionally, both procedures automatically transform all parameters as necessary before using the No-U-Turn Sampler. Table 2.11 displays each type of constraint that the parameters can have, as well as the associated transformations, inverse transformations, Jacobians, and derivatives of the log Jacobians.

Table 2.11 Parameter Transformations

Constraint	Transformation	Inverse Transformation	Jacobian	Derivative of Log Jacobian
	$u = T(\theta) =$	$\theta = T^{-1}(u) =$	$\frac{\partial \theta}{\partial u} =$	$\frac{\partial}{\partial u} \log \left \frac{\partial \theta}{\partial u} \right =$
$-\infty < \theta < \infty$	θ	u	1	0
$\min < \theta < \infty$	$\ln(\theta - \min)$	$\min + e^u$	e^u	1
$-\infty < \theta < \max$	$\ln(\max - \theta)$	$\max - e^u$	$-e^u$	1
$\min < \theta < \max$	$\ln \left(\frac{\theta - \min}{\max - \theta} \right)$	$\frac{\min + \max e^u}{1 + e^u}$	$\frac{(\max - \min)e^u}{(e^u + 1)^2}$	$1 - 2 \frac{e^u}{1 + e^u}$

Burn-In, Thinning, and Markov Chain Samples

Burn-in refers to the practice of discarding an initial portion of a Markov chain sample so that the effect of initial values on the posterior inference is minimized. For example, suppose the target distribution is $N(0, 1)$ and the Markov chain was started at the value 10^6 . The chain might quickly travel to regions around 0 in a few iterations. However, including samples around the value 10^6 in the posterior mean calculation can produce substantial bias in the mean estimate. In theory, if the Markov chain is run for an infinite amount of time, the effect of the initial values decreases to zero. In practice, you do not have the luxury of infinite samples. In practice, you assume that after t iterations, the chain has reached its target distribution and you can throw away the early portion and use the good samples for posterior inference. The value of t is the burn-in number.

With some models you might experience poor mixing (or slow convergence) of the Markov chain. This can happen, for example, when parameters are highly correlated with each other. Poor mixing means that the Markov chain slowly traverses the parameter space and the chain has high dependence. High sample autocorrelation can result in biased Monte Carlo standard errors. A common strategy is to *thin* the Markov chain in order to reduce sample autocorrelations. You thin a chain by keeping every k th simulated draw from each sequence. You can safely use a thinned Markov chain for posterior inference as long as the chain converges. It is important to note that thinning a Markov chain can be wasteful because you are throwing away a $\frac{k-1}{k}$ fraction of all the posterior samples generated. MacEachern and Berliner (1994) show that you always get more precise posterior estimates if the entire Markov chain is used. However, other factors, such as computer storage or plotting time, might prevent you from keeping all samples.

To use the CQLIM, CNTSELECT, and SMC procedures, you need to determine the total number of samples to keep ahead of time. This number is not obvious and often depends on the type of inference you want to make. Mean estimates do not require nearly as many samples as small-tail percentile estimates. In most applications, you might find that keeping a few thousand iterations is sufficient for reasonably accurate posterior inference. In all procedures, the relationship between the number of iterations requested, the number of iterations kept, and the amount of thinning is

$$\text{kept} = \left\lceil \frac{\text{requested}}{\text{thinning}} \right\rceil$$

where $\lceil \cdot \rceil$ is the rounding operator.

Assessing Markov Chain Convergence

Simulation-based Bayesian inference requires using simulated draws to summarize the posterior distribution or calculate any relevant quantities of interest. You need to treat the simulation draws with care. There are usually two issues. First, you have to decide whether the Markov chain has reached its stationary, or the desired posterior, distribution. Second, you have to determine the number of iterations to keep after the Markov chain has reached stationarity. Convergence diagnostics help to resolve these issues. Note that many diagnostic tools are designed to verify a necessary but not sufficient condition for convergence. There are no conclusive tests that can tell you when the Markov chain has converged to its stationary distribution. You should proceed with caution. Also, note that you should check the convergence of *all* parameters, and not just those of interest, before proceeding to make any inference. With some models, certain parameters can appear to have very good convergence behavior, but that could be misleading due to the slow convergence of

other parameters. If some of the parameters have bad mixing, you cannot get accurate posterior inference for parameters that appear to have good mixing. See Cowles and Carlin (1996) and Brooks and Roberts (1998) for discussions about convergence diagnostics.

Statistical Diagnostic Tests

The Bayesian procedures include several statistical diagnostic tests that can help you assess Markov chain convergence. For a detailed description of each of the diagnostic tests, see the following subsections. Table 2.12 provides a summary of the diagnostic tests and their interpretations. All these diagnostics are available in the CQLIM, CNTSELECT, and SMC procedures.

Table 2.12 Convergence Diagnostic Tests Available in the Bayesian Procedures

Name	Description	Interpretation of the Test
Geweke	Tests whether the mean estimates have converged by comparing means from the early and latter part of the Markov chain.	Two-sided test based on a z -score statistic. Large absolute z values indicate rejection.
Heidelberger-Welch (stationarity test)	Tests whether the Markov chain is a covariance (or weakly) stationary process. Failure could indicate that a longer Markov chain is needed.	One-sided test based on a Cramér-von Mises statistic. Small p -values indicate rejection.
Heidelberger-Welch (half-width test)	Reports whether the sample size is adequate to meet the required accuracy for the mean estimate. Failure could indicate that a longer Markov chain is needed.	If a relative half-width statistic is greater than a predetermined accuracy measure, this indicates rejection.
Raftery-Lewis	Evaluates the accuracy of the estimated (desired) percentiles by reporting the number of samples needed to reach the desired accuracy of the percentiles. Failure could indicate that a longer Markov chain is needed.	If the total samples needed are fewer than the Markov chain sample, this indicates rejection.
Autocorrelation	Measures dependency among Markov chain samples.	High correlations between long lags indicate poor mixing.
Effective sample size	Relates to autocorrelation; measures mixing of the Markov chain.	Large discrepancy between the effective sample size and the simulation sample size indicates poor mixing.

Geweke Diagnostics

The Geweke test (Geweke 1992) compares values in the early part of the Markov chain to those in the latter part of the chain in order to detect failure of convergence. If the chain has converged, early and later parts of the chain should look similar. The statistic is constructed as follows. Two subsequences of the Markov

chain $\{\theta^t\}$ are taken out, with $\{\theta_1^t : t = 1, \dots, n_1\}$ and $\{\theta_2^t : t = n_a, \dots, n\}$, where $1 < n_1 < n_a < n$. The first subsequence is called the “first window,” and the second sequence is called the “second window.” Let $n_2 = n - n_a + 1$ denote the length of the second window, and define the means of the two windows as

$$\bar{\theta}_1 = \frac{1}{n_1} \sum_{t=1}^{n_1} \theta^t \quad \text{and} \quad \bar{\theta}_2 = \frac{1}{n_2} \sum_{t=n_a}^n \theta^t$$

Let $\hat{s}_1(0)$ and $\hat{s}_2(0)$ denote consistent spectral density estimates at zero frequency (see the section “[Spectral Density Estimate at Zero Frequency](#)” on page 29 for estimation details) for each window. If $n_1 = f_1 n$ and $n_2 = f_2 n$, where the fractions $0 < f_1 < 1$ and $0 < f_2 < 1$ are fixed, $f_1 + f_2 < 1$, and the chain is stationary, then the following statistic converges to a standard normal distribution as $n \rightarrow \infty$:

$$Z_n = \frac{\bar{\theta}_1 - \bar{\theta}_2}{\sqrt{\frac{\hat{s}_1(0)}{n_1} + \frac{\hat{s}_2(0)}{n_2}}}$$

This is a two-sided test, and the p -value is computed as $P(X > |Z_n|)$, where X is a standard normal distribution. A p -value less than α indicates a rejection of the null hypothesis that the first window of the chain and the second window of the chain come from the same distribution. If the null hypothesis is rejected, the Markov chain has not yet converged.

Spectral Density Estimate at Zero Frequency

For one sequence of the Markov chain $\{\theta_t\}$, the relationship between the h -lag covariance sequence of a time series and the spectral density, f , is

$$s_h = \frac{1}{2\pi} \int_{-\pi}^{\pi} \exp(i\omega h) f(\omega) d\omega$$

where i indicates that ωh is the complex argument. Inverting this Fourier integral,

$$f(\omega) = \sum_{h=-\infty}^{\infty} s_h \exp(-i\omega h) = s_0 \left(1 + 2 \sum_{h=1}^{\infty} \rho_h \cos(\omega h) \right)$$

It follows that

$$f(0) = \sigma^2 \left(1 + 2 \sum_{h=1}^{\infty} \rho_h \right)$$

which gives an autocorrelation adjusted estimate of the variance. In this equation, σ^2 is the naive variance estimate of the sequence $\{\theta_t\}$ and ρ_h is the lag h autocorrelation. Due to obvious computational difficulties,

such as calculation of autocorrelation at infinity, you cannot effectively estimate $f(0)$ by using the preceding formula. The usual route is to first obtain the *periodogram* $p(\omega)$ of the sequence, and then estimate $f(0)$ by smoothing the estimated periodogram. The periodogram is defined to be

$$p(\omega) = \frac{1}{n} \left[\left(\sum_{t=1}^n \theta_t \sin(\omega t) \right)^2 + \left(\sum_{t=1}^n \theta_t \cos(\omega t) \right)^2 \right]$$

The procedures use the following way to estimate $\hat{f}(0)$ from p (Heidelberger and Welch 1981). In $p(\omega)$, let $\omega = \omega_k = 2\pi k/n$ and $k = 1, \dots, \lfloor \frac{n}{2} \rfloor$.² A smooth spectral density in the domain of $(0, \pi]$ is obtained by fitting a gamma model with the log link function, using $p(\omega_k)$ as response and $x_1(\omega_k) = \sqrt{3}(4\omega_k/(2\pi) - 1)$ as the only regressor. The predicted value $\hat{f}(0)$ is given by

$$\hat{f}(0) = \exp(\hat{\beta}_0 - \sqrt{3}\hat{\beta}_1)$$

where $\hat{\beta}_0$ and $\hat{\beta}_1$ are the estimates of the intercept and slope parameters, respectively.

Heidelberger-Welch Diagnostics

The Heidelberger-Welch test (Heidelberger and Welch 1981, 1983) consists of two parts: a stationary test and a half-width test. The stationarity test assesses the stationarity of a Markov chain by testing the hypothesis that the chain comes from a covariance stationary process. The half-width test checks whether the Markov chain sample size is adequate to estimate the mean values accurately.

Given $\{\theta^t\}$, set $S_0 = 0$, $S_n = \sum_{t=1}^n \theta^t$, and $\bar{\theta} = (1/n) \sum_{t=1}^n \theta^t$. You can construct the following sequence with s coordinates on values from $\frac{1}{n}, \frac{2}{n}, \dots, 1$:

$$B_n(s) = (S_{[ns]} - [ns]\bar{\theta}) / (n\hat{p}(0))^{1/2}$$

where $[]$ is the rounding operator and $\hat{p}(0)$ is an estimate of the spectral density at zero frequency that uses the second half of the sequence (see the section “[Spectral Density Estimate at Zero Frequency](#)” on page 29 for estimation details). For large n , B_n converges in distribution to a Brownian bridge (Billingsley 1986). So you can construct a test statistic by using B_n . The statistic that these procedures use is the Cramér–von Mises statistic;³ that is, $\int_0^1 B_n(s)^2 ds = \text{CVM}(B_n)$. As $n \rightarrow \infty$, the statistic converges in distribution to a standard Cramér–von Mises distribution. The integral $\int_0^1 B_n(s)^2 ds$ is numerically approximated using Simpson’s rule.

Let $y_i = B_n(s)^2$, where $s = 0, \frac{1}{n}, \dots, \frac{n-1}{n}, 1$ and $i = ns = 0, 1, \dots, n$. If n is even, let $m = n/2$; otherwise, let $m = (n-1)/2$. The Simpson’s approximation to the integral is

$$\int_0^1 B_n(s)^2 ds \approx \frac{1}{3n} [y_0 + 4(y_1 + \dots + y_{2m-1}) + 2(y_2 + \dots + y_{2m-2}) + y_{2m}]$$

²This is equivalent to the fast Fourier transformation of the original time series θ_t .

³ The von Mises distribution was first introduced by von Mises (1918). The density function is $\pi(\theta|\mu\kappa) \sim M(\mu, \kappa) = [2\pi I_0(\kappa)]^{-1} \exp(\kappa \cos(\theta - \mu))$ ($0 \leq \theta \leq 2\pi$), where the function $I_0(\kappa)$ is the modified Bessel function of the first kind and order zero, defined by $I_0(\kappa) = (2\pi)^{-1} \int_0^{2\pi} \exp(\kappa \cos(\theta - \mu)) d\theta$.

Note that Simpson's rule requires an even number of intervals. When n is odd, y_n is set to 0 and the value does not contribute to the approximation.

This is a one-sided test, and the p -value is computed as $P(X > \text{CVM}(B_n))$, where X has a standard Cramér–von Mises distribution. A p -value less than α indicates rejection of the null hypothesis that the chain is stationary. This test can be performed repeatedly on the same chain, and it helps you identify a time t when the chain has reached stationarity. The whole chain, $\{\theta^t\}$, is first used to construct the Cramér–von Mises statistic. If it passes the test, you can conclude that the entire chain is stationary. If it fails the test, you drop the initial 10% of the chain and redo the test by using the remaining 90%. This process is repeated until either a time t is selected or it reaches a point where there are not enough data remaining to construct a confidence interval (the cutoff proportion is set to 50%).

The part of the chain that is deemed stationary is put through a half-width test, which reports whether the sample size is adequate to meet certain accuracy requirements for the mean estimates. Running the simulation less than this length of time would not meet the requirement, whereas running it longer would not provide any additional information that is needed. The statistic that is calculated here is the *relative half-width* (RHW) of the confidence interval. The RHW for a confidence interval of level $1 - \alpha$ is

$$\text{RHW} = \frac{z_{(1-\alpha/2)} \cdot (\hat{s}_n/n)^{1/2}}{\hat{\theta}}$$

where $z_{(1-\alpha/2)}$ is the z -score of the $100(1 - \alpha/2)$ th percentile (for example, $z_{(1-\alpha/2)} = 1.96$ if $\alpha = 0.05$), $\hat{s}_n$ is the variance of the chain estimated using the spectral density method (see the explanation in the section “Spectral Density Estimate at Zero Frequency” on page 29), n is the length, and $\hat{\theta}$ is the estimated mean. The RHW quantifies accuracy of the $1 - \alpha$ level confidence interval of the mean estimate by measuring the ratio between the sample standard error of the mean and the mean itself. The test assumes that you can stop the Markov chain if the variability of the mean stabilizes with respect to the mean—in other words, if the RHW is small enough. An implicit assumption is that large means are often accompanied by large variances. If this assumption is not met, then this test can produce false rejections (such as a small mean around 0 and large standard deviation) or false acceptance (such as a very large mean with relative small variance). As with any other convergence diagnostics, you might want to exercise caution in interpreting the results.

To perform the stationarity test, you need to select an α level (the default is 0.05). To perform the half-width test, you need to select another α level (again, the default is 0.05) and a predetermined tolerance value ϵ (the default is 0.1). If the calculated RHW is greater than ϵ , you conclude that there are not enough data to accurately estimate the mean with $1 - \alpha$ confidence under a tolerance of ϵ .

Raftery-Lewis Diagnostics

If your interest lies in posterior percentiles, you want a diagnostic test that evaluates the accuracy of the estimated percentiles. The Raftery-Lewis test (Raftery and Lewis 1992, 1995) is designed for this purpose. The Raftery-Lewis test evaluates the accuracy of a quantile estimate $\hat{\theta}_q$ of order $q \in (0, 1)$. In particular, it determines the number of MCMC iterations, M , that need to be discarded (burn-ins) and the number of iterations that are needed, N , in order to achieve a desired level of precision in the estimate $\hat{\theta}_q$. Details follow. Notation and deductions here closely resemble those in Raftery and Lewis (1995).

The quantile θ_q is defined such that $P(\theta \leq \theta_q | \mathbf{y}) = q$, where q can be an arbitrary cumulative probability, such as 0.025. This θ_q can be empirically estimated by finding the $100nq$ th number of the sorted $\{\theta^t\}$. Let $\hat{\theta}_q$ denote the estimand, which corresponds to an estimated probability $P(\theta \leq \hat{\theta}_q) = \hat{P}_q$. Because the simulated

posterior distribution converges to the true distribution as the simulation sample size grows, $\hat{\theta}_q$ can achieve any degree of accuracy if the simulator is run for a very long time. However, running a simulation for too long can be wasteful. Alternatively, you can use coverage probability to measure accuracy and stop the chain when a certain accuracy is reached.

A stopping criterion is reached when the estimated probability is within $\pm r$ of the true cumulative probability q , with probability s , such as $P(\hat{P}_q \in (q - r, q + r)) = s$. For example, suppose you want the coverage probability s to be 0.95 and the amount of tolerance r to be 0.005. This corresponds to requiring that the estimate of the cumulative distribution function of the 2.5th percentile be estimated to within ± 0.5 percentage points with probability 0.95. In order to perform the diagnostic, a value ϵ must be chosen for the tolerance level of a stationary test, described in the following paragraphs. For example, $\epsilon = 0.001$.

The Raftery-Lewis diagnostic test finds the number of iterations, M , that need to be discarded (burn-ins) and the number of iterations that are needed, N , in order to achieve a desired precision. Given a predefined cumulative probability q , these procedures first find $\hat{\theta}_q$, and then they construct a binary 0 – 1 process $\{Z_t\}$ by setting $Z_t = 1$ if $\theta^t \leq \hat{\theta}_q$ and 0 otherwise for all t . The sequence $\{Z_t\}$ is itself not a Markov chain, but you can construct a subsequence of $\{Z_t\}$ that is approximately Markovian if it is sufficiently k -thinned. When k becomes reasonably large, $\{Z_t^{(k)}\}$ starts to behave like a Markov chain.

Next, the procedures find this thinning parameter k . The number k is estimated by comparing the Bayesian information criterion (BIC) between two Markov models: a first-order and a second-order Markov model. A j th-order Markov model is one in which the current value of $\{Z_t^{(k)}\}$ depends on the previous j values. For example, in a second-order Markov model,

$$\begin{aligned} P(Z_t^{(k)} = z_t | Z_{t-1}^{(k)} = z_{t-1}, Z_{t-2}^{(k)} = z_{t-2}, \dots, Z_0^{(k)} = z_0) \\ = P(Z_t^{(k)} = z_t | Z_{t-1}^{(k)} = z_{t-1}, Z_{t-2}^{(k)} = z_{t-2}) \end{aligned}$$

where $z_i = \{0, 1\}, i = 0, \dots, t$. Given $\{Z_t^{(k)}\}$, you can construct two transition count matrices for a second-order Markov model:

$$\begin{array}{cc|cc} & & z_t = 0 & & z_t = 1 \\ & & z_{t-1} = 0 & z_{t-1} = 1 & z_{t-1} = 0 & z_{t-1} = 1 \\ z_{t-2} = 0 & w_{000} & w_{010} & z_{t-2} = 0 & w_{001} & w_{011} \\ z_{t-2} = 1 & w_{100} & w_{110} & z_{t-2} = 1 & w_{101} & w_{111} \end{array}$$

For each k , the procedures calculate the BIC that compares the two Markov models. The BIC is based on a likelihood ratio test statistic that is defined as

$$G_k^2 = 2 \sum_{i=0}^1 \sum_{j=0}^1 \sum_{l=0}^1 w_{ijl} \log \frac{w_{ijl}}{\hat{w}_{ijl}}$$

where $\hat{w}_{ijl}$ is the expected cell count of w_{ijl} under the null model, the first-order Markov model, where the assumption $(Z_t^{(k)} \perp Z_{t-2}^{(k)}) | Z_{t-1}^{(k)}$ holds. The formula for the expected cell count is

$$\hat{w}_{ijl} = \frac{\sum_i w_{ijl} \cdot \sum_l w_{ijl}}{\sum_i \sum_l w_{ijl}}$$

The BIC is $G_k^2 - 2 \log(n_k - 2)$, where n_k is the k -thinned sample size (every k th sample, starting with the first), with the last two data points discarded because of the construction of the second-order Markov model. The thinning parameter k is the smallest k for which the BIC is negative. When k is found, you can estimate a transition probability matrix between state 0 and state 1 for $\{Z_t^{(k)}\}$:

$$Q = \begin{pmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{pmatrix}$$

Because $\{Z_t^{(k)}\}$ is a Markov chain, its equilibrium distribution exists and is estimated by

$$\pi = (\pi_0, \pi_1) = \frac{(\beta, \alpha)}{\alpha + \beta}$$

where $\pi_0 = P(\theta \leq \theta_q | \mathbf{y})$ and $\pi_1 = 1 - \pi_0$. The goal is to find an iteration number m such that after m steps, the estimated transition probability $P(Z_m^{(k)} = i | Z_0^{(k)} = j)$ is within ϵ of equilibrium π_i for $i, j = 0, 1$. Let $e_0 = (1, 0)$ and $e_1 = 1 - e_0$. The estimated transition probability after step m is

$$P(Z_m^{(k)} = i | Z_0^{(k)} = j) = e_j \left[\begin{pmatrix} \pi_0 & \pi_1 \\ \pi_0 & \pi_1 \end{pmatrix} + \frac{(1 - \alpha - \beta)^m}{\alpha + \beta} \begin{pmatrix} \alpha & -\alpha \\ -\beta & \beta \end{pmatrix} \right] e_j'$$

which holds when

$$m = \frac{\log \left(\frac{(\alpha + \beta)\epsilon}{\max(\alpha, \beta)} \right)}{\log(1 - \alpha - \beta)}$$

assuming $1 - \alpha - \beta > 0$.

Therefore, by time m , $\{Z_t^{(k)}\}$ is sufficiently close to its equilibrium distribution, and you know that a total size of $M = mk$ should be discarded as the burn-in.

Next, the procedures estimate N , the number of simulations that are needed in order to achieve the desired accuracy in percentile estimation. The estimate of $P(\theta \leq \theta_q | \mathbf{y})$ is $\bar{Z}_n^{(k)} = \frac{1}{n} \sum_{t=1}^n Z_t^{(k)}$. For large n , $\bar{Z}_n^{(k)}$ is normally distributed with mean q , the true cumulative probability, and variance

$$\frac{1}{n} \frac{(2 - \alpha - \beta)\alpha\beta}{(\alpha + \beta)^3}$$

$P(q - r \leq \bar{Z}_n^{(k)} \leq q + r) = s$ is satisfied if

$$n = \frac{(2 - \alpha - \beta)\alpha\beta}{(\alpha + \beta)^3} \left\{ \frac{\Phi^{-1}\left(\frac{s+1}{2}\right)}{r} \right\}^2$$

Therefore, $N = nk$.

By using similar reasoning, the procedures first calculate the minimal number of iterations that are needed in order to achieve the desired accuracy, assuming that the samples are independent:

$$N_{\min} = \left\{ \Phi^{-1}\left(\frac{s+1}{2}\right) \right\}^2 \frac{q(1-q)}{r^2}$$

If $\{\theta^t\}$ does not have that required sample size, the Raftery-Lewis test is not performed. If you still want to perform the test, increase the number of Markov chain iterations.

The ratio $N/N_{\min}$ is sometimes referred to as the *dependence factor*. It measures deviation from posterior sample independence: the closer it is to 1, the less correlated the samples are. There are a few things to keep in mind when you use this test. This diagnostic tool is specifically designed for the percentile of interest and does not provide information about convergence of the chain as a whole (Brooks and Roberts 1999). In addition, the test can be very sensitive to small changes. Both N and $N_{\min}$ are inversely proportional to r^2 , so you can expect to see large variations in these numbers with small changes to input variables, such as the desired coverage probability or the cumulative probability of interest. Finally, the time until convergence for a parameter can differ substantially for different cumulative probabilities.

Autocorrelations

The sample autocorrelation of lag h for a parameter θ is defined in terms of the sample autocovariance function:

$$\hat{\rho}_h(\theta) = \frac{\hat{\gamma}_h(\theta)}{\hat{\gamma}_0(\theta)}, \quad |h| < n$$

The sample autocovariance function of lag h of θ is defined by

$$\hat{\gamma}_h(\theta) = \frac{1}{n-h} \sum_{t=1}^{n-h} (\theta^{t+h} - \bar{\theta})(\theta^t - \bar{\theta}), \quad 0 \leq h < n$$

Effective Sample Size

You can use autocorrelation and trace plots to examine the mixing of a Markov chain. A closely related measure of mixing is the effective sample size (ESS) (Kass et al. 1998). ESS can be interpreted as the equivalent number of independent draws from the posterior distribution that the Markov chain represents.

ESS is defined as

$$\text{ESS} = \frac{n}{\tau} = \frac{n}{1 + 2 \sum_{k=1}^{\infty} \rho_k(\theta)}$$

where n is the total sample size and $\rho_k(\theta)$ is the autocorrelation of lag k for θ . The quantity τ is referred to as the autocorrelation time. To estimate τ , the Bayesian procedures first find a cutoff point k after which the autocorrelations are very close to zero, and then sum all the ρ_k up to that point. The cutoff point k is such that $|\rho_k| < \min\{0.01, 2s_k\}$, where s_k is the estimated standard deviation:

$$s_k = \sqrt{\left(\frac{1}{n} \left(1 + 2 \sum_{j=1}^{k-1} \hat{\rho}_j^2(\theta) \right) \right)}$$

ESS and τ are inversely proportional to each other, and low ESS or high τ indicates bad mixing of the Markov chain.

ESS is costly to compute when the Markov chain is highly autocorrelated because it requires so many autocorrelation lags to be computed first. To remedy this problem, you can specify the cutoff point to be no larger than some maximum lag. In the case where more than the maximum number of lags would be required, this reduces computational cost but results in poor-quality estimates of ESS and τ . However, when the maximum lag is set high enough, this problem occurs only when the Markov chain is mixing very poorly, which is precisely the issue that ESS and τ are meant to diagnose.

Summary Statistics

Let θ be a p -dimensional parameter vector of interest: $\theta = \{\theta_1, \dots, \theta_p\}$. For each $i \in \{1, \dots, p\}$, there are n observations: $\theta_i = \{\theta_i^t, t = 1, \dots, n\}$.

Mean

The posterior mean is calculated by using the following formula:

$$E(\theta_i | \mathbf{y}) \approx \bar{\theta}_i = \frac{1}{n} \sum_{t=1}^n \theta_i^t, \text{ for } i = 1, \dots, p$$

Standard Deviation

Sample standard deviation is computed as the square root of the sample variance, which is calculated by using the following formula:

$$\text{Var}(\theta_i | \mathbf{y}) \approx s_i^2 = \frac{1}{n-1} \sum_{t=1}^n (\theta_i^t - \bar{\theta}_i)^2$$

Standard Error of the Mean Estimate

Suppose that you have n iid samples, the mean estimate is $\bar{\theta}_i$, and the sample standard deviation is s_i . The standard error of the estimate is $\hat{\sigma}_i / \sqrt{n}$. However, positive autocorrelation (see the section “[Autocorrelations](#)” on page 34 for a definition) in the MCMC samples makes this an underestimate. To account for the autocorrelation, the Bayesian procedures correct the standard error by using effective sample size (see the section “[Effective Sample Size](#)” on page 34).

Given an effective sample size of m , the standard error for $\bar{\theta}_i$ is $\hat{\sigma}_i / \sqrt{m}$. The following formula for the variance of $\bar{\theta}_i$ is equivalent:

$$\widehat{\text{Var}}(\bar{\theta}_i) = \frac{1 + 2 \sum_{k=1}^{\infty} \rho_k(\theta_i)}{n} \cdot \frac{\sum_{t=1}^n (\theta_i^t - \bar{\theta}_i)^2}{(n-1)}$$

The standard error of the mean is also known as the Monte Carlo standard error (MCSE). The MCSE provides a measurement of the accuracy of the posterior estimates, and small values do not necessarily indicate that you have recovered the true posterior mean.

Percentiles

Sample percentiles are calculated using the following steps. Let τ denote a percentile; let $p = \frac{\tau}{100}$; let the τ th percentile of θ_i be θ_i^p ; and let $\theta_{i(t)}$ for $t = 1, 2, \dots, n$ denote the posterior sample of θ_i , sorted so that $\theta_{i(t)} \leq \theta_{i(t+1)}$ for all t . Next define $np = j + g$, where j is the integer part of np and g is the fractional part. Then the τ th percentile of the posterior distribution of θ_i is calculated as

$$\theta_i^p = \begin{cases} \frac{1}{2} (\theta_{i(j)} + \theta_{i(j+1)}) & \text{if } g = 0 \\ \theta_{i(j+1)} & \text{if } g > 0 \end{cases}$$

Correlation

Correlation between θ_i and θ_j is calculated as

$$r_{ij} = \frac{\sum_{t=1}^n (\theta_i^t - \bar{\theta}_i) (\theta_j^t - \bar{\theta}_j)}{\sqrt{\sum_t (\theta_i^t - \bar{\theta}_i)^2 \sum_t (\theta_j^t - \bar{\theta}_j)^2}}$$

Covariance

Covariance between θ_i and θ_j is calculated as

$$s_{ij} = \sum_{t=1}^n (\theta_i^t - \bar{\theta}_i) (\theta_j^t - \bar{\theta}_j) / (n-1)$$

Equal-Tail Credible Interval

Let $\Pi(\theta_i | \mathbf{y})$ denote the marginal posterior cumulative distribution function of θ_i . A 100 $(1 - \alpha)$ % Bayesian equal-tail credible interval for θ_i is $(\theta_i^{\alpha/2}, \theta_i^{1-\alpha/2})$, where $\Pi(\theta_i^{\alpha/2} | \mathbf{y}) = \frac{\alpha}{2}$ and $\Pi(\theta_i^{1-\alpha/2} | \mathbf{y}) = 1 - \frac{\alpha}{2}$. The interval is obtained using the empirical $\frac{\alpha}{2}$ th and $(1 - \frac{\alpha}{2})$ th percentiles of $\{\theta_i^t\}$.

Highest Posterior Density (HPD) Interval

For a definition of an HPD interval, see the section “[Interval Estimation](#)” on page 10. SAS Econometrics procedures use the Chen-Shao algorithm (Chen and Shao 1999; Chen, Shao, and Ibrahim 2000) to estimate an empirical HPD interval of θ_i :

1. Sort $\{\theta_i^t\}$ to obtain the ordered values:

$$\theta_{i(1)} \leq \theta_{i(2)} \leq \dots \leq \theta_{i(n)}$$

2. Compute the 100 $(1 - \alpha)$ % credible intervals:

$$R_j(n) = (\theta_{i(j)}, \theta_{i(j + [(1-\alpha)n])})$$

for $j = 1, 2, \dots, n - [(1 - \alpha)n]$.

3. The 100 $(1 - \alpha)$ % HPD interval, denoted by $R_{j^*}(n)$, is the one with the smallest interval width among all credible intervals.

Using Multiple Chains

An effective strategy to check for convergence is to run multiple chains and check whether they yield similar results. Then the chains must be combined appropriately to compute both multichain posterior summary statistics and multichain MCMC diagnostics. For multichain posterior summary statistics such as the posterior mean, posterior quantiles, posterior intervals, or the posterior covariance and correlation matrices, the chains can be combined into one long chain, and then the statistics can be computed as if there were only one chain. To compute the MCSE of multichain estimates, or to compute multichain convergence diagnostics, you must take more care to account for the different autocorrelation structures in each of the chains.

Multichain Standard Error of the Mean Estimate

Suppose there are m chains, each of n iterations. Then each chain yields an estimate of the posterior mean of the parameter θ of the form

$$\bar{\theta}_j = \frac{1}{n} \sum_{t=1}^n \theta^{jt}$$

where $j = 1, 2, \dots, m$ index chains. Based on the central limit theorem, these estimates are independent and have the following asymptotic normal distributions:

$$\bar{\theta}_j \sim N(\mu, \sigma^2 / \text{ESS}_j)$$

where $\mu = E(\theta|y)$ is the true posterior mean, $\sigma^2 = E((\theta - \mu)^2|y)$ is the true posterior variance, and ESS_j is the effective sample size of θ in the j th chain.

The multichain estimate of the posterior mean is the mean of all the iterations over all chains, which is equivalent to taking the mean of single-chain posterior mean estimates. Then the asymptotic distribution of the multichain estimate of the posterior mean is

$$\bar{\theta} = \frac{1}{m} \sum_{j=1}^m \bar{\theta}_j \sim N \left(\mu, \frac{\sigma^2}{m^2} \sum_{j=1}^m \frac{1}{ESS_j} \right)$$

Then σ^2 is estimated using the multichain estimate of the posterior variance, denoted by $\hat{\sigma}^2$, and obtained by computing the sample variance, using all chains as one large sample. Estimates of the effective sample sizes are obtained via $ESS_j = n/\tau_j$, where τ_j is the estimated autocorrelation time for θ in the j th chain. This yields

$$\widehat{\text{Var}}(\bar{\theta}) = \frac{\hat{\sigma}^2}{nm} \frac{\sum_{j=1}^m \tau_j}{m}$$

The multichain MCSE is the square root of this value.

Multichain Effective Sample Size

From the preceding formula for multichain MCSE, multichain estimates of effective sample size and autocorrelation time naturally arise. The multichain estimate of autocorrelation time is simply the mean of the single-chain autocorrelation times:

$$\tau = \frac{1}{m} \sum_{j=1}^m \tau_j$$

Given τ , the multichain estimate of ESS is

$$ESS = \frac{nm}{\tau}$$

Multichain Autocorrelations

The multichain estimate of autocorrelation time can be equivalently constructed by first computing the mean autocorrelations for each lag, then computing autocorrelation time using the mean autocorrelations. So the multichain estimate of the lag- h autocorrelation is the mean autocorrelation for that lag, given by

$$\rho_h = \frac{1}{m} \sum_{j=1}^m \rho_{jh}$$

where ρ_{jh} is the lag- h autocorrelation from the j th chain.

Other Diagnostics

The Geweke, Heidelberger-Welch, and Raftery-Lewis diagnostics are solely single-chain diagnostics and cannot be constructed for multiple chains. They can be used with multiple chains by constructing them for each chain and then by checking whether all chains pass the diagnostic. For example, the Geweke diagnostic and the stationarity test of the Heidelberger-Welch diagnostic can both detect whether any of the chains have not yet converged. However, the half-width test of the Heidelberger-Welch diagnostic and the Raftery-Lewis diagnostic both assume that you will use only a single chain to do posterior inference. This means that these tests will sometimes tell you that you need longer chains than necessary to achieve adequate precision in estimating posterior quantities.

A Bayesian Reading List

This section lists a number of Bayesian textbooks of varying levels.

Introductory Books

- Berry, D. A. (1996). *Statistics: A Bayesian Perspective*. Belmont, CA: Duxbury Press.
- Bolstad, W. M. (2016). *Introduction to Bayesian Statistics*. 3rd ed. Hoboken, NJ: John Wiley & Sons.
- DeGroot, M. H., and Schervish, M. J. (2011). *Probability and Statistics*. 4th ed. Reading, MA: Addison-Wesley.
- Gamerman, D., and Lopes, H. F. (2006). *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. 2nd ed. Boca Raton, FL: Chapman & Hall/CRC.
- Ghosh, J. K., Delampady, M., and Samanta, T. (2006). *An Introduction to Bayesian Analysis: Theory and Methods*. New York: Springer-Verlag.
- Lee, P. M. (2012). *Bayesian Statistics: An Introduction*. 4th ed. London: Edward Arnold.
- Sivia, D. S. (2006). *Data Analysis: A Bayesian Tutorial*. 2nd ed. Oxford: Oxford University Press.
- Stone, J. V. (2013). *Bayes' Rule: A Tutorial Introduction to Bayesian Analysis*. Sheffield, UK: Sebtel Press.

Intermediate-Level Books

- Box, G. E. P., and Tiao, G. C. (1992). *Bayesian Inference in Statistical Analysis*. New York: John Wiley & Sons.
- Chen, M.-H., Shao, Q.-M., and Ibrahim, J. G. (2000). *Monte Carlo Methods in Bayesian Computation*. New York: Springer-Verlag.
- Gelman, A., and Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge: Cambridge University Press.
- Goldstein, M., and Wooff, D. A. (2007). *Bayes Linear Statistics: Theory and Methods*. Chichester, UK: John Wiley & Sons.

- Harney, H. L. (2016). *Bayesian Inference: Data Evaluation and Decisions*. 2nd ed. Berlin: Springer-Verlag.
- Kadane, J. B. (2011). *Principles of Uncertainty*. London: Chapman & Hall/CRC.
- Leonard, T., and Hsu, J. S. J. (1999). *Bayesian Methods: An Analysis for Statisticians and Interdisciplinary Researchers*. Cambridge: Cambridge University Press.
- Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. New York: Springer-Verlag.
- Marin, J.-M., and Robert, C. P. (2007). *Bayesian Core: A Practical Approach to Computational Bayesian Statistics*. New York: Springer-Verlag.
- Press, S. J. (2002). *Subjective and Objective Bayesian Statistics: Principles, Models, and Applications*. 2nd ed. New York: Wiley-Interscience.
- Robert, C. P. (2001). *The Bayesian Choice*. 2nd ed. New York: Springer-Verlag.
- Robert, C. P., and Casella, G. (2004). *Monte Carlo Statistical Methods*. 2nd ed. New York: Springer-Verlag.
- Tanner, M. A. (1996). *Tools for Statistical Inference: Methods for the Exploration of Posterior Distributions and Likelihood Functions*. 3rd ed. New York: Springer-Verlag.

Advanced Titles

- Berger, J. O. (1993). *Statistical Decision Theory and Bayesian Analysis*. 2nd ed. New York: Springer-Verlag.
- Bernardo, J. M., and Smith, A. F. M. (2007). *Bayesian Theory*. 2nd ed. Chichester, UK: John Wiley & Sons.
- Damien, P., Dellaportas, P., Polson, N. G., and Stephens, D. A. (2013) *Bayesian Theory and Applications*. Oxford: Oxford University Press.
- De Finetti, B. (1992). *Theory of Probability: A Critical Introductory Treatment*. Chichester, UK: John Wiley & Sons.
- Jeffreys, H. (1998). *Theory of Probability*. 3rd ed. Oxford: Oxford University Press.
- Kadane, J. B. (2016). *Pragmatics of Uncertainty*. London: Chapman & Hall/CRC.
- O'Hagan, A., and Foster, J. J. (2004). *Bayesian Inference*. 2nd ed. Volume 2B of Kendall's Advanced Theory of Statistics. London: Edward Arnold.
- Savage, L. J. (1954). *The Foundations of Statistics*. New York: John Wiley & Sons.

Books Motivated by Statistical Applications and Data Analysis

- Carlin, B. P., and Louis, T. A. (2008). *Bayesian Methods for Data Analysis*. 3rd ed. London: Chapman & Hall.
- Congdon, P. (2014). *Applied Bayesian Modeling*. 2nd ed. New York: John Wiley & Sons.
- Congdon, P. (2005). *Bayesian Models for Categorical Data*. Chichester, UK: John Wiley & Sons.
- Congdon, P. (2006). *Bayesian Statistical Modeling*. 2nd ed. New York: John Wiley & Sons.
- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, D., and Rubin, D. B. (2013). *Bayesian Data Analysis*. 3rd ed. London: Chapman & Hall.
- Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (1996). *Markov Chain Monte Carlo in Practice*. London: Chapman & Hall.

References

- Amit, Y. (1991). “On Rates of Convergence of Stochastic Relaxation for Gaussian and Non-Gaussian Distributions.” *Journal of Multivariate Analysis* 38:82–99.
- Applegate, D. L., Kannan, R., and Polson, N. (1990). *Random Polynomial Time Algorithms for Sampling from Joint Distributions*. Technical report, School of Computer Science, Carnegie Mellon University.
- Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis*. 2nd ed. New York: Springer-Verlag.
- Berger, J. O. (2006). “The Case for Objective Bayesian Analysis.” *Bayesian Analysis* 3:385–402. <http://www.stat.cmu.edu/bayesworkshop/2005/berger.pdf>.
- Berger, J. O., and Wolpert, R. (1988). *The Likelihood Principle*. 2nd ed. Hayward, CA: Institute of Mathematical Statistics.
- Bernardo, J. M., and Smith, A. F. M. (1994). *Bayesian Theory*. New York: John Wiley & Sons.
- Besag, J. (1974). “Spatial Interaction and the Statistical Analysis of Lattice Systems.” *Journal of the Royal Statistical Society, Series B* 36:192–236.
- Betancourt, M. (2018). “A Conceptual Introduction to Hamiltonian Monte Carlo.” <https://arxiv.org/abs/1701.02434v2>.
- Billingsley, P. (1986). *Probability and Measure*. 2nd ed. New York: John Wiley & Sons.
- Box, G. E. P., and Tiao, G. C. (1973). *Bayesian Inference in Statistical Analysis*. New York: John Wiley & Sons.
- Breiman, L. (1968). *Probability*. Reading, MA: Addison-Wesley.
- Brooks, S. P., and Roberts, G. O. (1998). “Assessing Convergence of Markov Chain Monte Carlo Algorithms.” *Statistics and Computing* 8:319–335.

- Brooks, S. P., and Roberts, G. O. (1999). "On Quantile Estimation and Markov Chain Monte Carlo Convergence." *Biometrika* 86:710–717.
- Carlin, B. P., and Louis, T. A. (2000). *Bayes and Empirical Bayes Methods for Data Analysis*. 2nd ed. London: Chapman & Hall.
- Casella, G., and George, E. I. (1992). "Explaining the Gibbs Sampler." *American Statistician* 46:167–174.
- Chan, K. S. (1993). "Asymptotic Behavior of the Gibbs Sampler." *Journal of the American Statistical Association* 88:320–326.
- Chen, M.-H., and Shao, Q.-M. (1999). "Monte Carlo Estimation of Bayesian Credible and HPD Intervals." *Journal of Computational and Graphical Statistics* 8:69–92.
- Chen, M.-H., Shao, Q.-M., and Ibrahim, J. G. (2000). *Monte Carlo Methods in Bayesian Computation*. New York: Springer-Verlag.
- Chib, S., and Greenberg, E. (1995). "Understanding the Metropolis-Hastings Algorithm." *American Statistician* 49:327–335.
- Congdon, P. (2001). *Bayesian Statistical Modeling*. Chichester, UK: John Wiley & Sons.
- Congdon, P. (2003). *Applied Bayesian Modeling*. New York: John Wiley & Sons.
- Congdon, P. (2005). *Bayesian Models for Categorical Data*. Chichester, UK: John Wiley & Sons.
- Cowles, M. K., and Carlin, B. P. (1996). "Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review." *Journal of the American Statistical Association* 91:883–904.
- DeGroot, M. H., and Schervish, M. J. (2002). *Probability and Statistics*. 3rd ed. Reading, MA: Addison-Wesley.
- Feller, W. (1968). *An Introduction to Probability Theory and Its Applications*. 3rd ed. New York: John Wiley & Sons.
- Gelfand, A. E., Hills, S. E., Racine-Poon, A., and Smith, A. F. M. (1990). "Illustration of Bayesian Inference in Normal Data Models Using Gibbs Sampling." *Journal of the American Statistical Association* 85:972–985.
- Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis*. 2nd ed. London: Chapman & Hall.
- Geman, S., and Geman, D. (1984). "Stochastic Relaxation, Gibbs Distribution, and the Bayesian Restoration of Images." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6:721–741.
- Geweke, J. (1992). "Evaluating the Accuracy of Sampling-Based Approaches to Calculating Posterior Moments." In *Bayesian Statistics*, vol. 4, edited by J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, 169–193. Oxford: Clarendon Press.
- Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (1996). *Markov Chain Monte Carlo in Practice*. London: Chapman & Hall.
- Goldstein, M. (2006). "Subjective Bayesian Analysis: Principles and Practice." *Bayesian Analysis* 3:403–420. <http://www.stat.cmu.edu/bayesworkshop/2005/goldstein.pdf>.

- Hastings, W. K. (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika* 57:97–109.
- Heidelberger, P., and Welch, P. D. (1981). "A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations." *Communications of the ACM* 24:233–245.
- Heidelberger, P., and Welch, P. D. (1983). "Simulation Run Length Control in the Presence of an Initial Transient." *Operations Research* 31:1109–1144.
- Hoffman, M. D., and Gelman, A. (2014). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research* 15:1351–1381.
- Karlin, S., and Taylor, H. (1975). *A First Course in Stochastic Processes*. 2nd ed. Orlando, FL: Academic Press.
- Kass, R. E., Carlin, B. P., Gelman, A., and Neal, R. M. (1998). "Markov Chain Monte Carlo in Practice: A Roundtable Discussion." *American Statistician* 52:93–100.
- Kass, R. E., and Wasserman, L. (1996). "Formal Rules of Selecting Prior Distributions: A Review and Annotated Bibliography." *Journal of the American Statistical Association* 91:343–370.
- Liu, C., Wong, W. H., and Kong, A. (1991a). *Correlation Structure and Convergence Rate of the Gibbs Sampler (I): Application to the Comparison of Estimators and Augmentation Scheme*. Technical report, Department of Statistics, University of Chicago.
- Liu, C., Wong, W. H., and Kong, A. (1991b). *Correlation Structure and Convergence Rate of the Gibbs Sampler (II): Applications to Various Scans*. Technical report, Department of Statistics, University of Chicago.
- Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. New York: Springer-Verlag.
- MacEachern, S. N., and Berliner, L. M. (1994). "Subsampling the Gibbs Sampler." *American Statistician* 48:188–190.
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). "Equation of State Calculations by Fast Computing Machines." *Journal of Chemical Physics* 21:1087–1092.
- Metropolis, N., and Ulam, S. (1949). "The Monte Carlo Method." *Journal of the American Statistical Association* 44:335–341.
- Meyn, S. P., and Tweedie, R. L. (1993). *Markov Chains and Stochastic Stability*. Berlin: Springer-Verlag.
- Neal, R. M. (2011). "MCMC Using Hamiltonian Dynamics." In *Handbook of Markov Chain Monte Carlo*, edited by S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, 113–161. Boca Raton, FL: CRC Press.
- Press, S. J. (2003). *Subjective and Objective Bayesian Statistics*. New York: John Wiley & Sons.
- Raftery, A. E., and Lewis, S. M. (1992). "One Long Run with Diagnostics: Implementation Strategies for Markov Chain Monte Carlo." *Statistical Science* 7:493–497.
- Raftery, A. E., and Lewis, S. M. (1995). "The Number of Iterations, Convergence Diagnostics, and Generic Metropolis Algorithms." In *Markov Chain Monte Carlo in Practice*, edited by W. R. Gilks, D. J. Spiegelhalter, and S. Richardson, 115–130. London: Chapman & Hall.

- Robert, C. P. (2001). *The Bayesian Choice*. 2nd ed. New York: Springer-Verlag.
- Robert, C. P., and Casella, G. (2004). *Monte Carlo Statistical Methods*. 2nd ed. New York: Springer-Verlag.
- Roberts, G. O. (1996). “Markov Chain Concepts Related to Sampling Algorithms.” In *Markov Chain Monte Carlo in Practice*, edited by W. R. Gilks, D. J. Spiegelhalter, and S. Richardson, 45–58. London: Chapman & Hall.
- Roberts, G. O., Gelman, A., and Gilks, W. R. (1997). “Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms.” *Annals of Applied Probability* 7:110–120.
- Roberts, G. O., and Rosenthal, J. S. (2001). “Optimal Scaling for Various Metropolis-Hastings Algorithms.” *Statistical Science* 16:351–367.
- Rosenthal, J. S. (1991a). *Rates of Convergence for Data Augmentation on Finite Sample Spaces*. Technical report, Department of Mathematics, Harvard University.
- Rosenthal, J. S. (1991b). *Rates of Convergence for Gibbs Sampling for Variance Component Models*. Technical report, Department of Mathematics, Harvard University.
- Ross, S. M. (1997). *Simulation*. 2nd ed. Orlando, FL: Academic Press.
- Schervish, M. J. (1995). *Theory of Statistics*. New York: Springer-Verlag.
- Schervish, M. J., and Carlin, B. P. (1992). “On the Convergence of Successive Substitution Sampling.” *Journal of Computational and Graphical Statistics* 1:111–127.
- Stacy, E. W. (1962). “A Generalization of the Gamma Distribution.” *Annals of Mathematical Statistics* 33:1187–1192.
- Tanner, M. A. (1993). *Tools for Statistical Inference: Methods for the Exploration of Posterior Distributions and Likelihood Functions*. New York: Springer-Verlag.
- Tanner, M. A., and Wong, W. H. (1987). “The Calculation of Posterior Distributions by Data Augmentation.” *Journal of the American Statistical Association* 82:528–540.
- Tierney, L. (1994). “Markov Chains for Exploring Posterior Distributions.” *Annals of Statistics* 22:1701–1762.
- Von Mises, R. (1918). “Über die ‘Ganzzahligkeit’ der Atomgewichte und verwandte Fragen [About the ‘integer nature’ of atomic masses and related questions].” *Physikalische Zeitschrift* 19:490–500.
- Wasserman, L. (2004). *All of Statistics: A Concise Course in Statistical Inference*. New York: Springer-Verlag.

Chapter 3

Introduction to Packages for the TSMODEL Procedure

Contents	
Overview	45
What Is a Package?	46
What Are Objects, Classes, Instances, Functions, and Methods?	47
What Are Objects?	47
What Are Classes?	48
What Are Instances?	48
What Are Functions?	48
What Are Methods?	48
How Do You Use Packages?	49
The REQUIRE Statement	49
The DECLARE OBJECT Statement	49
Chapter Organization	50
Syntax Conventions	51
Object Names	51
Object Method Names	51
Return Codes	52

Overview

The chapter describes the use of packages in the TSMODEL procedure. It defines the main concepts of packages and describes the steps for using packages in the TSMODEL procedure, the package syntax conventions and terminology, and the structural organization of each package’s chapter.

NOTE: When you license SAS Visual Forecasting, you also have access to SAS/ETS software and SAS Forecast Server Procedures. For more information about SAS/ETS procedures, see *SAS/ETS User’s Guide*. For more information about SAS Forecast Server Procedures, see *SAS Forecast Server Procedures: User’s Guide*.

What Is a Package?

A package is a set of related specialized objects and functions that tackles a unique facet of the time series analysis problem. You can use these specialized objects and functions when you write your custom SAS code in order to gain access both to cutting-edge data analysis tools and to utilities that are designed to significantly speed up code development and improve the quality of the resulting code.

Table 3.1 shows the packages that you can use with the TSMODEL procedure. Each package has a unique abbreviation that contains up to five alphabetic characters.

Table 3.1 Packages for the TSMODEL Procedure

Package Name	Abbreviation	Description
Automatic time series analysis and forecasting	ATSM	Tools for automatic modeling and forecasting of time series using various model families such as exponential smoothing (ESM), ARIMA, intermittent demand (IDM), and unobserved component models (UCM)
External Languages	EXTLANG	Tools for enabling the seamless integration of external language programs into SAS environments
Time series motif discovery	MOTIF	Tools for the discovery of frequent patterns or repeated subsequences in time series
Multivariate singular spectrum analysis	MSSA	Tools for decomposing a multivariate time series into additive components and categorizing those components based on the magnitudes of their contributions
Simple forecast service	SFS	Tools for automatic forecasting of time series with a simple-to-use interface and using only ARIMA models
Singular spectrum analysis	SSA	Tools for decomposing a time series into additive components and categorizing those components based on the magnitudes of their contributions
Subspace tracking	SST	Tools for tracking the principal subspace of multiple time series
Time filters	TIMFIL	Tools for filtering and aggregation of time series
Time frequency analysis	TFA	Tools for efficient analysis of time series in both time and frequency domains
Time series analysis	TSA	Tools for efficient statistical analysis of time series (transformations, decompositions, statistical tests for intermittency, seasonality, stationarity, and forecast bias, and so on)
Time series dimension reduction	TDR	Tools for reducing the dimensionality of time series

Table 3.1 *continued*

Package Name	Abbreviation	Description
Time series distance measure	TSD	Tools for measuring the distance between two time series or among sequences in temporal data
Time series model	TSM	Tools for efficient time series modeling and forecasting
Utility	UTL	Tools for performing basic statistical computations on pairs of actual and predicted time series

What Are Objects, Classes, Instances, Functions, and Methods?

This section describes the computer programming terminology that is used in the documentation of packages for the TSMODEL procedure.

What Are Objects?

A programming object, a fundamental concept in object-oriented programming (OO) and analogous to an object found in nature, is the description of the state and behavior of a system. For example, a dog is a real-world object that has both a state (name, age, weight, fur color, tail length, running speed, number of teeth, friendliness level, and so on) and behavior (running, biting, eating, hunting, wagging its tail, whining, sleeping, tearing the sofa, and so on). A programming object is similar to a real-world object, but it stores its state as internal variables called fields and communicates its behavior to other objects via a set of functions called methods.

Many of the objects in the TSMODEL procedure packages are collector objects that are specifically designed to postprocess the output of other objects and then store the results in an output table. This postprocessing and storing the results can help accelerate your development process. These objects are called collectors because they “gather” the results of other objects and store them for future reference.

Conversely, some other objects are specifically designed to retrieve the data that have been stored in an output table by collectors and make those data available to your custom SAS code. These objects are called repeater objects.

What Are Classes?

A class is a programming recipe for creating objects. Whereas real objects can be described by an unlimited number of features, a class uses a limited number of features to describe as faithfully as possible both the state and behavior of an object inside a computer program. The size and complexity of the feature set that a class uses (that is, how complex the recipe for the object is) depends on the requirements of each application. A class uses fields of varying data types to represent the features that describe the state of an object. Similarly, a class provides a set of methods that enables external objects to query and even modify the object's behavior. Furthermore, a class also generally provides the recipe for a constructor, which is a special method that assigns an initial state to the class fields.

The words “class” and “object” are sometimes used interchangeably because a class is the representation of an object within the constraints of a programming language.

What Are Instances?

An instance of a class is a unique replica of a class that represents an object inside a computer program. A computer program creates instances of a class by allocating a block of memory and using different portions of that block to represent the different fields and methods that are described in the recipe given by the class. A computer program can create multiple unique instances of the same class in memory to represent many independent objects.

What Are Functions?

A function, a fundamental concept in computer programming, is a named block of instructions that performs a specific task. Functions allow programs to be modularized into reusable blocks of code that are written once and can then be invoked many times. In addition, modularization makes a program more readable and enables developers to work simultaneously on different portions of the same program.

What Are Methods?

A method is similar to a function, but it belongs to a specific class. In a computer program, all instances of a class contain the same methods. However, the methods of each instance can access only the instance's own internal variables. Functions can interact with instances of a class by invoking its methods.

How Do You Use Packages?

There are two steps involved in using packages in the TSMODEL procedure. First, you specify in the REQUIRE statement the packages that you want to use in your SAS code. Second, you declare instances of package objects in your SAS code by using a DECLARE OBJECT statement for each instance. The following sections further elaborate on these steps.

The REQUIRE Statement

The REQUIRE statement in the TSMODEL procedure enables your custom SAS code to access the objects and functions in the packages that are specified in the statement. You can use specify multiple packages in a single REQUIRE statement, as follows:

```
REQUIRE package-1 < package-2 ... package-n > ;
```

You can selectively identify which objects within a package you intend to use, as shown in the following statement:

```
REQUIRE package (class-1 ... class-n) ;
```

In contrast to objects, the functions offered by a package are always available when a package is registered via the REQUIRE statement; they do not have to be individually specified.

The REQUIRE statement does not automatically create any instances of the objects in the packages that you specify. For more information about the REQUIRE statement, see the Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*).

The DECLARE OBJECT Statement

You use the DECLARE OBJECT statement in your custom SAS code to create one or more unique instances of package objects. Instances of the same object are completely independent from each other, and there is no limit on how many instances of an object you can create in your program. You can access package functions by simply invoking them directly from your custom SAS code, but you must declare instances of package objects in order to use them in your custom SAS code. You interact with instances of objects by invoking their methods, as is illustrated in the following statements:

```
DECLARE OBJECT name-1 (class) ;
```

```
DECLARE OBJECT name-2 (class) ;
```

```
rc1=name1.SomeMethod (Argument-1 ... Argument-n) ;
```

```
rc2=name2.SomeMethod (Argument-1 ... Argument-n) ;
```

These statements declare two unique instances of the *class* object: *name1* and *name2*. You can interact with these two instances independently by invoking their *SomeMethod* method and passing in the required arguments. Method calls always return a numeric status code (*rc*) to indicate successful or failure. By convention, the objects in the TSMODEL procedure packages return a negative number to indicate that a computational failure has occurred.

Chapter Organization

This book is organized as follows.

This chapter provides an overview that describes what packages for the TSMODEL procedure are and how they are used.

The remaining chapters describe the various packages that are available for the TSMODEL procedure. Each of these chapters is organized as follows:

- The “Overview” section describes the broad purpose of the package; it describes which specific facet of the time series analysis problem the package is designed to tackle. It also provides a table that lists all the objects and functions that are offered by the package along with a brief description of each object’s functionalities. You can use the contents of this table to quickly locate an object that offers the functionalities you need. For packages that contain objects that interact with each other to perform larger and more complex tasks, a data flow diagram is also provided in this section to succinctly describe the interactions between the various objects via their methods. The direction of the arrows in this diagram indicates how the output of object methods feeds into other object methods. This section also describes the status codes that can be returned by an object methods and functions in the packages.
- A section for each object follows the “Overview” section. Sections for objects have the following structure:
 - The “Object Summary” section is divided into three parts: “Synopsis,” “Data Flow Diagram,” and “Methods.” Each object that is offered by a package has its own “Object Summary” section, which starts with a detailed description of the functionality of the object. Read this section if you want to understand more about the inner workings of an object (for example, to find out mathematical techniques that an object uses internally). This section also includes a table that summarizes all the available object methods along with a brief description of their functionalities. Refer to this table to quickly determine how to control the behavior of an object. You can also search for the methods that are prefixed with “Set” and “Get.” In addition, if the object is a collector, this section provides a table that describes the corresponding table schema that the object generates. The table schema description includes the name, SAS data type, and a brief description of each table column.
 - The “Synopsis” section shows how you can declare an instance of the object in your custom SAS code. This section describes the correct sequence of method calls (when a particular sequence is required) and indicates which method arguments are optional or mandatory. For example, for the FORENG object in the ATSM package, you provide forecast data to the FORENG object via its Initialize method, optionally set the forecast lead via the SetOption method, then run the forecast via its Run method, and optionally retrieve a forecast series via the GetForecast method. The “Synopsis” section also optionally offers a data flow diagram of the object that depicts the flow of data into and out of the object via its methods and the relevant data types that are required as arguments by each method. This diagram provides you with a visual overview of how to interact with the object from your program.
 - The “Methods” section provides a detailed description of the object’s methods. A subsection for each method includes its functionality and arguments. Method arguments are divided into either

input or output arguments. Input arguments are not modified by a method internally. Output arguments can be modified by a method internally. The size and value of an output argument after the execution of a method might differ from its original size and value before the method call, but the data type always remains the same. The description of each argument also lists the SAS data types that are acceptable for that argument. Each method argument can accept one or more SAS data types.

- The “Examples” section follows the detailed description of all the package’s objects. This section includes various examples that depict how the various objects provided by the package interact with each other and how they interact with objects in other packages to solve problems. These examples generally use almost all methods in each object and can provide a good starting point for your own custom implementations.
- The “References” section contains references for the methodology and for examples of the procedure.

Syntax Conventions

This section describes the common terminology and design patterns that are used by all packages that can be used with the TSMODEL procedure.

Object Names

Names of collector objects have the prefix “OUT” followed by a descriptive object name. For example, the OUTFREQ collector object is designed to store the results of an instance of the FREQ object in the TSA package. Similarly, the OUTEST object is designed to store the time series model parameter estimates that are computed by an instance of the FORENG object in the ATSM package.

Similarly, names of repeater objects have the prefix “IN” followed by a descriptive object name. For example, the INDIAG object is used to read in the set of diagnostic control specifications that were previously stored into a table by an instance of the OUTDIAG collector object in the ATSM package.

Object Method Names

Many of the objects in the TSMODEL procedure packages allow their behaviors to be controlled via the specification of options. For example, the FORENG object in the ATSM package provides automatic time series forecasting capabilities and enables you to specify the lead time, forecast horizon, and confidence level size of the computed forecast confidence limits, among many other options that affect the forecast. Names of object methods that are designed specifically for setting options have the prefix “Set” followed by the option name, as in the following example:

```
DECLARE OBJECT s (SFS) ;
```

```
rc=s.SetY (y) ;
```

In addition, various objects offer a single overarching method called “SetOption” that is used to set all object properties, with the exception of the input time series data. To quickly determine how to control the behavior of objects, you can search for the description of an overarching “SetOption” method in the documentation of the objects that you plan to use in your custom SAS code. Similarly, names of object methods that are designed specifically to enable you to retrieve both data and properties from objects have the prefix “Get” followed by a descriptive name. For example, the “GetForecast” method in the FORENG object enables you to retrieve various forecast series that are computed by the object and include them in your custom SAS code for further processing.

Return Codes

All object methods and functions offered by the TSMODEL procedure packages return a status code upon their execution. Negative status codes are used to indicate the occurrence of computational failures. Positive return codes, including 0, are customized to the needs of individual objects and functions. Generally, a status code of 0 indicates unconditional success, and a status code greater than 0 indicates conditional success.

Chapter 4

Shared Concepts

Contents

Introduction to Shared Concepts	54
Using CAS Sessions and CAS Engine Librefs	54
Loading a SAS Data Set onto a CAS Server	55
Syntax Common to SAS Econometrics Procedures	56
CLASS Statement	56
DISPLAY Statement	60
DISPLAYOUT Statement	61
EFFECT Statement	62
Collection Effects	64
Multimember Effects	64
Polynomial Effects	66
Spline Effects	69
Splines and Spline Bases	72
ODS Table Names	77
SELECTION Statement	78
Optimization Options	84
Details for SAS Econometrics Procedures	89
Levelization of Classification Variables	89
Specification and Parameterization of Model Effects	91
Effect Operators	92
GLM Parameterization of Classification Variables and Effects	94
Nonsingular Parameterization	99
Class Variable Parameterization with Unbalanced Designs	101
Model Selection Methods	102
Forward Selection	102
Backward Elimination	104
Stepwise Selection	105
Forward-Swap Selection	106
Informative Missingness	106
Multithreading	108
Choosing an Optimization Algorithm	108
First- or Second-Order Algorithms	108
Algorithm Descriptions	109
References	112

Introduction to Shared Concepts

SAS Econometrics procedures run on SAS Viya. One component of SAS Viya is SAS Cloud Analytic Services (CAS), which is the analytic server and associated cloud services. The following subsections describe how to set up and use CAS sessions.

The section “[Syntax Common to SAS Econometrics Procedures](#)” on page 56 describes the common syntax elements that are supported by some of the procedures in this book. In some cases, individual procedures implement these common elements in slightly different ways. When this occurs, the differences are described in the respective procedure chapters.

The section “[Details for SAS Econometrics Procedures](#)” on page 89 provides details that are common to some of the procedures in this book.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Loading a SAS Data Set onto a CAS Server

Procedures in this book require the input data to reside on a CAS server. To work with a SAS data set, you must first load the data set onto the CAS server. Data loaded on the CAS server are called *data tables*. This section lists three methods of loading a SAS data set onto a CAS server. In this section, `mylib` is the name of the caslib that is connected to the `mysess` CAS session.

- You can use a single DATA step to create a data table on the CAS server as follows:

```
data mylib.Sample;
  input y x @@;
  datalines;
.46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7
;
```

Note that DATA step operations might not work as intended when you perform them on the CAS server instead of the SAS client.

- You can create a SAS data set first, and when it contains exactly what you want, you can use another DATA step to load it onto the CAS server as follows:

```
data Sample;
  input y x @@;
  datalines;
.46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7 .78 8
;
data mylib.Sample;
  set Sample;
run;
```

- You can use the CASUTIL procedure as follows:

```
proc casutil sessref=mysess;
  load data=Sample casout="Sample";
quit;
```

The CASUTIL procedure can load data onto a CAS server more efficiently than the DATA step. For more information about the CASUTIL procedure, see *SAS Cloud Analytic Services: User's Guide*.

The `mylib` caslib stores the `Sample` data table, which can be distributed across many machine nodes. You must use a caslib reference in procedures in this book to enable the SAS client machine to communicate with the CAS session. For example, the following SEVSELECT procedure statements use a data table that resides in the `mylib` caslib:

```
proc sevselect data = mylib.Sample;
  ...statements...;
run;
```

You can delete your data table by using the DELETE procedure as follows:

```
proc delete data = mylib.Sample;
run;
```

The Sample data table is accessible only in the mysess session. When you terminate the mysess session, the Sample data table is no longer accessible from the CAS server. If you want your Sample data table to be available to other CAS sessions, then you must *promote* your data table. For more information about data tables, see *SAS Cloud Analytic Services: User's Guide*.

Syntax Common to SAS Econometrics Procedures

CLASS Statement

```
CLASS variable <(options)> . . . <variable <(options)> > > </ global-options> ;
```

This section applies to the following procedures: CNTSELECT, CPANEL, CQLIM, CSPATIALREG, and SEVSELECT.

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. These variables enter the analysis not through their values, but through levels to which the unique values are mapped. For more information about these mappings, see the section “[Levelization of Classification Variables](#)” on page 89.

If the procedure permits a classification variable as a response (dependent variable or target), the response does not need to be specified in the CLASS statement.

You can specify options either as individual variable *options*, by enclosing the options in parentheses after the variable name, or as *global-options*, by placing them after a slash (/). *Global-options* are applied to all variables that are specified in the CLASS statement. If you specify more than one CLASS statement, the *global-options* that are specified in any one CLASS statement apply to all CLASS statements. However, individual CLASS variable *options* override the *global-options*.

Table 4.1 summarizes the values you can use for either an *option* or a *global-option*. The options are described in detail in the list that follows Table 4.1.

Table 4.1 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable
SPLIT	Splits levels of CLASS variables into independent effects

DESCENDING**DESC**

reverses the sort order of the classification variable. If both the DESCENDING and **ORDER=** options are specified, the procedure orders the categories according to the **ORDER=** option and then reverse that order.

MISSING

treats missing values (“.”, “.A”, . . . , “.Z” for numeric variables and blanks for character variables) as valid values for the CLASS variable.

If you do not specify the MISSING option, observations that have missing values for CLASS variables are removed from the analysis.

ORDER=FORMATTED | FREQ | INTERNAL

specifies the sort order for the levels of classification variables. This order determines which parameters in the model correspond to each level in the data. The following table shows how to interpret values of the **ORDER=** keywords:

Value of ORDER=	Levels Sorted By
FORMATTED	External formatted values, except for numeric variables that have no explicit format, which are sorted by their unformatted (internal) values. The sort order is machine-dependent. For numeric variables for which you have supplied no explicit format, the levels are ordered by their internal values.
FREQ	Descending frequency count (levels that have more observations come earlier in the order)
INTERNAL	Unformatted value. The sort order is machine-dependent.

The **ORDER=** option does not apply to missing values, which are always ordered first.

For more information about sort order, see the chapter about the SORT procedure in *Base SAS Procedures Guide* and the discussion of BY-group processing in the “Grouping Data” section of *SAS Programmers Guide: Essentials*. By default, **ORDER=FORMATTED**.

PARAM=keyword

specifies the parameterization method for the classification variable or variables. You can specify any of the *keywords* shown in the following table; design matrix columns are created from CLASS variables according to the corresponding coding schemes.

Table 4.2 Value of **PARAM=**

Value of PARAM=	Coding
EFFECT	Effect coding. The REF= option in the CLASS statement determines the reference level.
GLM	Less-than-full-rank reference cell coding. This <i>keyword</i> can be used only as a <i>global-option</i> and is applied to all CLASS variables; all other individual variable parameterization specifications are ignored. The REF= option in the CLASS statement indirectly determines the reference level through the order of levels.

Table 4.2 *continued*

Value of PARAM=	Coding
ORDINAL THERMOMETER	Cumulative parameterization for an ordinal CLASS variable
POLYNOMIAL POLY	Polynomial coding. If the classification variable is numeric, then the ORDER= option in the CLASS statement is ignored, and the internal unformatted values are used.
REFERENCE REF	Reference cell coding. The REF= option in the CLASS statement determines the reference level.
ORTHEFFECT	Orthogonalizes PARAM=EFFECT coding. The REF= option in the CLASS statement determines the reference level.
ORTHORDINAL ORTHOTHERM	Orthogonalizes PARAM=ORDINAL coding
ORTHPOLY	Orthogonalizes PARAM=POLYNOMIAL coding. If the classification variable is numeric, then the ORDER= option in the CLASS statement is ignored, and the internal unformatted values are used.
ORTHREF	Orthogonalizes PARAM=REFERENCE coding. The REF= option in the CLASS statement determines the reference level.

All parameterizations are full rank, except for the GLM parameterization. If you specify a full rank parameterization for any CLASS variable, then every CLASS variable without a specified coding is given the EFFECT coding.

By default, PARAM=GLM. For more information about how parameterization of classification variables affects the construction and interpretation of model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 91.

REF='level' | *keyword*

REFERENCE='level' | *keyword*

specifies the reference level that is used when you specify a nonsingular parameterization. You can specify the following values:

'level' specifies the *level* of the variable to use as the reference level. Specify the formatted value of the variable if a format is assigned. You can specify this value only for an individual variable *option*.

FIRST designates the first ordered level as reference. You can specify this value either for an individual variable *option* or for a *global-option*.

LAST designates the last ordered level as reference. You can specify this value either for an individual variable *option* or for a *global-option*.

By default, REF=LAST.

SPLIT

specifies that design matrix columns that correspond to any effect that contains a split classification variable can be selected to enter or leave a model independently of the other design columns of that effect. This option applies to procedures that perform model selection.

Suppose that the variable `temp` has three levels ('hot', 'warm', and 'cold'), that the variable `gender` has two levels ('M' and 'F'), and that the variables are used in a PROC SEVSELECT run as follows:

```
proc sevselect data=mylib.data;
  loss y;
  class temp gender / split;
  scalemodel gender gender*temp;
run;
```

The two effects in the SCALEMODEL statement are split into eight independent effects. The effect “gender” is split into two effects that are labeled “gender_M” and “gender_F”. The effect “gender*temp” is split into six effects that are labeled “gender_M*temp_hot”, “gender_F*temp_hot”, “gender_M*temp_warm”, “gender_F*temp_warm”, “gender_M*temp_cold”, and “gender_F*temp_cold”. The previous PROC SEVSELECT step is equivalent to the following:

```
proc sevselect data=mylib.data;
  loss y;
  scalemodel gender_M gender_F
             gender_M*temp_hot gender_F*temp_hot
             gender_M*temp_warm gender_F*temp_warm
             gender_M*temp_cold gender_F*temp_cold;
run;
```

The SPLIT option can be used on individual classification variables. For example, consider the following PROC SEVSELECT step:

```
proc sevselect data=mylib.data;
  loss y;
  class temp(split) gender;
  scalemodel gender gender*temp;
run;
```

In this case, the effect “gender” is not split and the effect “gender*temp” is split into three effects, which are labeled “gender*temp_hot”, “gender*temp_warm”, and “gender*temp_cold”. Furthermore, each of these three split effects now has two parameters that correspond to the two levels of “gender.” The PROC SEVSELECT step is equivalent to the following:

```
proc sevselect data=mylib.data;
  loss y;
  class gender;
  scalemodel gender gender*temp_hot gender*temp_warm gender*temp_cold;
run;
```

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

This section applies to the following procedures: CCDM, CCOPULA, CNTSELECT, CPANEL, CQLIM, CSPATIALREG, ECM, HMM, and SEVSELECT.

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the specific procedure chapters. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

This section applies to the following procedures: CCDM, CCOPULA, CNTSELECT, CPANEL, CQLIM, CSPATIALREG, ECM, HMM, and SEVSELECT.

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the output table name.

key specifies *key* as the ODS table name and also as the output table name.

The table names that you can specify are listed in the specific procedure chapters.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

EFFECT Statement

EFFECT *effect-name* = *effect-type* (*var-list* < / *effect-options* >) ;

This section applies to the following procedures: CNTSELECT and SEVSELECT.

The EFFECT statement enables you to construct special collections of columns for design matrices. These collections are called *constructed effects* to distinguish them from the usual model effects that are formed from continuous or classification variables, as discussed in the section “GLM Parameterization of Classification Variables and Effects” on page 94. For example, the terms A, B, x, A*x, and A*B in the following statements define fixed effects of the usual type in a generalized linear model:

```
proc sevselect;
  loss y;
  class A B;
  scalemodel A B x A*x;
run;
```

A constructed effect, on the other hand, is assigned through the EFFECT statement. For example, in the following program, the EFFECT statement defines a constructed effect named spl:

```
proc sevselect;
  loss y;
  class A B;
  effect spl = spline(x);
  scalemodel A B A*spl;
run;
```

The columns of spl are formed from the data set variable x as a cubic B-spline basis with three equally spaced interior knots.

Each constructed effect corresponds to a collection of columns that are referred to by the name you supply. You can specify multiple EFFECT statements, and all EFFECT statements must precede the SCALEMODEL statement in the SEVSELECT procedure or the MODEL statement in other procedures.

You must specify the following arguments:

effect-name

names the effect. This name can appear in only one EFFECT statement and cannot be the name of a variable in the input data set.

effect-type

specifies the type of effect. You can specify the following *effect-types*:

COLLECTION

specifies a collection effect that defines one or more variables as a single effect that has multiple degrees of freedom. The variables in a collection are considered as a unit for purposes of estimation and inference. For more information, see the section “Collection Effects” on page 64.

MULTIMEMBER MM	specifies a multimember classification effect whose levels are determined by one or more variables that appear in a CLASS statement. For more information, see the section “ Multimember Effects ” on page 64.
POLYNOMIAL POLY	specifies a multivariate polynomial effect in the specified numeric variables. For more information, see the section “ Polynomial Effects ” on page 66.
SPLINE	specifies a regression spline effect whose columns are univariate spline expansions of one or more variables. A spline expansion replaces the original variable with an expanded or larger set of new variables. For more information, see the section “ Spline Effects ” on page 69.

var-list

specifies a list of variables that are used in constructing the effect. You can also specify any of the *effect-options* that are shown in [Table 4.3](#) after a slash following the *var-list*.

Table 4.3 EFFECT Statement Options

Option	Description
Collection Effects Options	
DETAILS	Displays the constituents of the collection effect
Multimember Effects Options	
DETAILS	Displays the levels of the multimember effect
NOEFFECT	Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns
STDIZE	Standardizes the design matrix entries so that each observation has a sum of 1
WEIGHT=	Specifies the weight variable for the contributions of each classification effect
Polynomial Effects Options	
DEGREE=	Specifies the degree of the polynomial
DETAILS	Displays details of the specified polynomial
MDEGREE=	Specifies the maximum degree of any variable in a term of the polynomial
NOSEPARATE	Treats the polynomial as a single effect with multiple degrees of freedom
STANDARDIZE=	Specifies centering and scaling suboptions for the variables that define the polynomial

Table 4.3 *continued*

Option	Description
Spline Effects Options	
BASIS=	Specifies the type of basis (B-spline basis or truncated power function basis) for the spline effect
DATABOUNDARY	Uses the extremes of the data as boundary knots for a B-spline basis
DEGREE=	Specifies the degree of the spline effect
DETAILS	Displays the knots and locations for each spline basis function
KNOTMAX=	Requests equally spaced right-side boundary knots starting at the variables' maximum and ending at the KNOTMAX= value
KNOTMETHOD=	Specifies how to construct the knots for the spline effect
KNOTMIN=	Requests equally spaced left-side boundary knots starting at the KNOTMIN= value and ending at the variables' minimum value
NATURALCUBIC	Specifies a natural cubic spline basis for the spline effect
SEPARATE	Treats the spline basis for each variable as a separate effect when multiple variables are specified
SPLIT	Treats each design matrix column as a separate effect for selection methods

Collection Effects

EFFECT *effect-name*=**COLLECTION** (*var-list* < / **DETAILS** >) ;

You use a collection effect to define a set of variables that are treated as a single effect that has multiple degrees of freedom. The variables in *var-list* can be continuous or classification variables. Columns in the design matrix that are contributed by a collection effect are the design columns of its constituent variables in the order in which they appear in the definition of the collection effect. If you specify the **DETAILS** option, then a table that shows the constituents of the collection effect is displayed.

Multimember Effects

EFFECT *effect-name*=**MULTIMEMBER** (*var-list* < / *mm-options* >) ;

EFFECT *effect-name*=**MM** (*var-list* < / *mm-options* >) ;

A multimember effect is formed from one or more classification variables in such a way that each observation can be associated with one or more levels of the union of the levels of the classification variables. In other words, a multimember effect is a classification-type effect with possibly more than one nonzero column entry for each observation. Multimember effects are useful, for example, in modeling the following:

- nurses' effects on patient recovery in hospitals
- teachers' effects on student scores
- lineage effects in genetic studies

The levels of a multimember effect consist of the union of formatted values of the variables that define this effect. Each such level contributes one column to the design matrix. For each observation, the value that

corresponds to each level of the multimember effect in the design matrix is the number of times that this level occurs for the observation.

For example, the following data provide teacher information and end-of-year test scores for students after two semesters:

Student	Score	Teacher1	Teacher2
Mary	87	Tobias	Cohen
Tom	89	Rodriguez	Tobias
Fred	82	Cohen	Cohen
Jane	88	Tobias	.
Jack	99	.	.

For example, Mary had different teachers in the two semesters, Fred had the same teacher in both semesters, and Jane received instruction only in the first semester.

You can model the effect of the teachers on student performance by specifying a multimember effect as follows:

```
CLASS teacher1 teacher2;
EFFECT teacher = MM(teacher1 teacher2);
```

The levels of the teacher effect are Cohen, Rodriguez, and Tobias, and the associated design matrix columns are as follows:

Student	Cohen	Rodriguez	Tobias
Mary	1	0	1
Tom	0	1	1
Fred	2	0	0
Jane	0	0	1
Jack	.	.	.

You can specify the following *mm-options* after a slash (/):

DETAILS

displays a table that shows the levels of the multimember effect.

NOEFFECT

specifies that, for observations of the multimember variables whose levels are all missing, the values in the corresponding design matrix columns be set to 0. If, in the preceding example, the teacher effect is defined by the following statement, then the associated design matrix columns' values for Jack are all 0:

```
EFFECT teacher = MM(teacher1 teacher2 / noeffect);
```

This statement enables you to include Jack in the analysis even though there is no effect of teachers on his performance.

A situation where it is important to designate observations as having no effect that can be attributed to a classification variable is the analysis of crossover designs, where lagged treatment levels are used to model the carryover effects of treatments between periods. Because there is no carryover effect for the first period, the treatment lag effect in a crossover design can be modeled by using a multimember effect that consists of a single classification variable and the NOEFFECT option, as in the following statements:

```
CLASS Treatment lagTreatment;
EFFECT Carryover = MM(lagTreatment / noeffect);
```

The lagTreatment variable contains a missing value for the first period. Otherwise, it contains the value of the treatment variable for the preceding period.

STDIZE

specifies that for each observation, the entries in the design matrix that corresponds to the multimember effect be scaled to have a sum of 1.

WEIGHT=*weight-list*

specifies numeric variables used to weigh the contributions of each of the classification effects that define the constructed multimember effect. The number of variables in *weight-list* must match the number of classification variables that define the effect.

Polynomial Effects

```
EFFECT effect-name=POLYNOMIAL (var-list </ polynomial-options>);
```

```
EFFECT effect-name=POLY (var-list </ polynomial-options>);
```

The variables in *var-list* must be numeric. A design matrix column is generated for each term of the specified polynomial. By default, each of these terms is treated as a separate effect for the purpose of model building. For example, the following two PROC sevselect,upper() steps yield the identical analysis:

```
proc sevselect;
  loss y;
  effect MyPoly = polynomial(x1-x3/degree=2);
  scalemodel MyPoly;
run;

proc sevselect;
  loss y;
  scalemodel x1 x2 x3 x1*x1 x1*x2 x1*x3 x2*x2 x2*x3 x3*x3;
run;
```

You can specify the following *polynomial-options* after a slash (/):

DEGREE=*n*

specifies the degree of the polynomial, where *n* must be a positive integer and is typically a small integer, such as 1, 2, or 3. By default, DEGREE=1.

DETAILS

displays a table that shows the details of the specified polynomial, including the number of terms generated. If you also specify the [STANDARDIZE](#) option, then a table that shows the standardization details is also produced.

MDEGREE=*n*

specifies the maximum degree of any variable in a term of the polynomial, where *n* must be a positive integer. The default is the degree of the specified polynomial. For example, the following statement generates the terms x_1 , x_2 , x_1^2 , x_1x_2 , x_2^2 , $x_1^2x_2$, $x_1x_2^2$ and $x_1^2x_2^2$:

```
EFFECT MyPoly=POLYNOMIAL(x1 x2/degree=4 MDEGREE=2);
```

NOSEPARATE

treats the polynomial as a single effect that has multiple degrees of freedom. The *effect-name* that you specify is used as the constructed effect name, and the labels of the terms are used as labels of the corresponding parameters.

STANDARDIZE <(*centerscale-opts*)> <= *standardize-opt*>

standardizes the variables that define the polynomial. By default, the standardized variables receive prefix “s_” in the variable names.

You can use the following *centerscale-opts* to specify how the center and scale are estimated:

METHOD=MOMENTS

estimates the center by the variable mean and the scale by the standard deviation. If a weight variable is specified using a WEIGHT statement, the weights are not used in the computation of the mean and standard deviation, and observations that have invalid weights are ignored. Only observations that are used in performing the analysis are used for the standardization.

METHOD=RANGE

estimates the center by the midpoint of the variable range and the scale as half the variable range. Any observation that has a missing value for any regressor used in the model is ignored when the range of variables in a polynomial effect is computed. Observations that have valid regressor values but missing or invalid values of frequency variables, weight variables, or dependent variables are used in computing variable ranges. By default, METHOD=RANGE.

METHOD=WMOMENTS

is the same as METHOD=MOMENTS except that weighted means and weighted standard deviations are used.

PREFIX=NONE | *quoted-string*

specifies the prefix that is appended to standardized variables when forming the term labels. You can specify the following values:

quoted-string specifies the prefix
NONE does not apply a prefix

By default, PREFIX="s_".

Let

n = number of observations used in the analysis
 w = weight variable
 f = frequency variable
 x = variable to be standardized
 $x_{(n)}$ = $\text{Max}_{i=1}^n(x_i)$
 $x_{(1)}$ = $\text{Min}_{i=1}^n(x_i)$
 F = sum of frequencies
 $= \sum_{i=1}^n f_i$
WF = sum of weighted frequencies
 $= \sum_{i=1}^n w_i f_i$

Table 4.4 shows how the center and scale are computed for each of the supported methods.

Table 4.4 Center and Scale Estimates by Method

Method	Center	Scale
RANGE	$(x_{(n)} + x_{(1)})/2$	$(x_{(n)} - x_{(1)})/2$
MOMENTS	$\bar{x} = \sum_{i=1}^n f_i x_i / F$	$\sqrt{\sum_{i=1}^n f_i (x_i - \bar{x})^2 / (F - 1)}$
WMOMENTS	$\bar{x}_w = \sum_{i=1}^n w_i f_i x_i / \text{WF}$	$\sqrt{\sum_{i=1}^n w_i f_i (x_i - \bar{x}_w)^2 / (F - 1)}$

You can control whether the standardization is to center, scale, or both center and scale by specifying one of the following *standardize-opts*:

CENTER

centers but does not scale the variables. For a variable x ,

$$s_x = x - \text{center}$$

CENTERSCALE

centers and scales the variables. This is the default if you do not specify a *standardization-opt*.

For a variable x ,

$$s_x = \frac{x - \text{center}}{\text{scale}}$$

NONE

performs no standardization.

SCALE

scales but does not center the variables. For a variable x ,

$$s_x = \frac{x}{\text{scale}}$$

Spline Effects

This section discusses the construction of spline effects through the EFFECT statement. A spline effect expands variables into spline bases whose form depends on the options that you specify. For more information about regression splines and spline bases, see the section “[Splines and Spline Bases](#)” on page 72. You request a spline effect with the syntax

EFFECT *name*=**SPLINE** (*var-list* < / *spline-options* >) ;

The variables in *var-list* must be numeric. Design matrix columns are generated separately for each of these variables, and the set of columns is collectively referred to with the specified name. By default, the spline basis that is generated for each variable is a cubic B-spline basis that has three equally spaced knots positioned between the minimum and maximum values of that variable. This yields by default seven design matrix columns for each of the variables in the SPLINE effect.

You can specify the following *spline-options* after a slash (/):

BASIS=BSPLINE

uses a B-spline basis for the spline expansion. For splines of degree d that are defined with n knots, this basis consists of $n + d + 1$ columns. In order to completely specify the B-spline basis, d left-side boundary knots and $\max\{d, 1\}$ right-side boundary knots are also required. For information about how to specify the positions of both the internal and boundary knots, see the [KNOTMETHOD=](#), [DATABOUNDARY](#), [KNOTMIN=](#), and [KNOTMAX=](#) suboptions.

BASIS=TPF(*options*)

specifies a truncated power function basis for the spline expansion. For splines of degree d that are defined with n knots for a variable x , this basis consists of an intercept, polynomials $x, x^2, \dots, x^d$, and one truncated power function for each of the n knots. Unlike the B-spline basis, no boundary knots are required. For information about how you can specify the position of the internal knots, see the [KNOTMETHOD=](#) suboption.

You can modify the number of columns when you request BASIS=TPF by specifying the following *options*:

NOINT excludes the intercept column.

NOPOWERS excludes the intercept and polynomial columns.

By default, BASIS=BSPLINE.

DATABOUNDARY

uses the extremes of the data as boundary knots when building a B-spline basis.

DEGREE=*n*

specifies the degree of the spline transformation, where *n* must be a nonnegative integer and is typically a small integer, such as 0, 1, 2, or 3. By default, DEGREE=3.

DETAILS

displays tables that show the knot locations and the knots associated with each spline basis function.

KNOTMAX=*value*

requests that, for each variable in the EFFECT statement, the right-side boundary knots be equally spaced starting at the maximum of the variable and ending at the specified value. This option is ignored for variables whose maximum value is greater than the specified value or if the **DATABOUNDARY** option is also specified.

KNOTMETHOD=*knot-method* < (*knot-options*) >

specifies how to construct the knots for spline effects. You can choose from the following *knot-methods* and affect the knot construction further with the method-specific *knot-options*:

EQUAL < (*n*) >

specifies that *n* equally spaced knots be positioned between the extremes of the data. By default, *n* = 3. For a B-spline basis, any needed boundary knots continue to be equally spaced unless the **DATABOUNDARY** option has also been specified. By default, KNOTMETHOD=EQUAL(3).

LIST(*number-list*)

specifies the list of internal knots to be used in forming the spline basis columns. For a B-spline basis, the data extremes are used as boundary knots.

LISTWITHBOUNDARY(*number-list*)

specifies the list of all knots that are used in forming the spline basis columns. When you use a truncated power function basis, this list is interpreted as the list of internal knots. When you use a B-spline basis of degree *d*, then the first *d* entries are used as left-side boundary knots and the last max(*d*, 1) entries in the list are used as right-side boundary knots.

MULTISCALE < (*multiscale-options*) >

generates multiple B-spline bases, which correspond to sets that have an increasing number of internal knots. As the number of internal knots increases, the spline basis that is generated can approximate features of the data at finer scales. So generating bases at multiple scales facilitates the modeling of both coarse- and fine-grained features of the data. For scale *i*, the spline basis corresponds to 2^i equally spaced internal knots. By default, the bases for scales 0–7 are generated. For each scale, a separate spline effect is generated. The name of the constructed spline effect at scale *i* is formed by appending *_Si* to the effect name that you specify in the EFFECT statement. If you specify multiple variables in the EFFECT statement, then spline bases are generated separately for each variable at each scale and the name of the corresponding effect is obtained by appending the variable name followed by *_Si* to the name in the EFFECT statement. For example, the following statement generates effects named spl_x1_S0, spl_x1_S1, spl_x1_S2, . . . , spl_x1_S7 and spl_x2_S1, spl_x2_S2, . . . , spl_x2_S7:


```
EFFECT spl = spline(x1 x2 / knotmethod=multiscale);
```

The MULTISCALE option is ignored if you specify the BASIS=TPF *spline-option*.

The following *multiscale-options* control which scales are included:

STARTSCALE=*n* specifies the start scale, where *n* is a positive integer. By default, STARTSCALE=0.

ENDSCALE=*n* specifies the end scale, where *n* is a positive integer. By default, ENDSCALE=7.

PERCENTILES(*n*)

requests that internal knots be placed at *n* equally spaced percentiles of the variable or variables named in the EFFECT statement. For example, the following statement positions internal knots at the deciles of the variable x. For a B-spline basis, the extremes of the data are used as boundary knots:

```
EFFECT spl = spline(x / knotmethod=percentiles(9));
```

RANGEFRACTIONS(*fraction-list*)

places internal knots at each fraction of the ranges of the variables in the EFFECT statement. For example, if variable x1 ranges between 1 and 3 and variable x2 ranges between 0 and 20, then the following EFFECT statement uses internal knots 1.2, 2, and 2.5 for variable x1 and internal knots 2, 10, and 15 for variable x2:

```
EFFECT spl = spline(x1 x2 / knotmethod=rangefractions(.1 .5 .75));
```

For a B-spline basis, the data extremes are used as boundary knots.

KNOTMIN=*value*

requests that, for each variable in the EFFECT statement, the left-side boundary knots be equally spaced starting at the specified value and ending at the minimum of the variable. This option is ignored for variables whose minimum value is less than the specified value or if the DATABOUNDARY option is also specified.

NATURALCUBIC

uses a natural cubic spline basis for the spline expansion. Natural cubic splines, also known as restricted cubic splines, are cubic splines that are constrained to be linear beyond the extreme knots. The natural cubic spline basis that is produced by the EFFECT statement is obtained by starting from the unrestricted truncated power function cubic spline basis that is defined with *n* distinct knots and imposes the linearity constraints beyond the extreme knots. This basis consists of an intercept, the polynomial *x*, and *n* – 2 functions that are all linear beyond the largest knot. The *i*th function, *i* = 1, 2, ..., *n* – 2, is 0 to the left of the *i*th knot, which is called the “break knot.” For more information about this basis, see the section “[Splines and Spline Bases](#)” on page 72. You can use the NOINT and NOPOWERS suboptions of the BASIS=TPF option to suppress the intercept and polynomial *x* when the columns of the natural cubic spline basis are formed. When you specify the NATURALCUBIC option, the options BASIS=BSPLINE, DATABOUNDARY, DEGREE=, and KNOTMETHOD=MULTISCALE are not applicable.

SEPARATE

requests that, when multiple variables are specified in the EFFECT statement, the spline basis for each variable be treated as a separate effect. The names of these separated effects are formed by appending an underscore followed by the name of the variable to the name that you specify in the EFFECT statement. For example, the following statement generates the effect names spl_x1 and spl_x2:

```
EFFECT spl = spline(x1 x2 / separate);
```

In procedures that support variable selection, such as the SEVSELECT procedure, these two effects can enter or leave the model independently during the selection process.

SPLIT

treats each individual column in the design matrix that corresponds to the spline effect as a separate effect that can enter or leave the model independently. Names for these split effects are generated by appending the variable name and an index for each column to the name that you specify in the EFFECT statement. For example, the effects generated for the spline effect in the following statement are spl_x1:1, spl_x1:2, ..., spl_x1:7 and spl_x2:1, spl_x2:2, ..., spl_x2:7:

```
EFFECT spl = spline(x1 x2 / split);
```

Splines and Spline Bases

This section provides details about how the EFFECT statement constructs spline bases. A spline function is a piecewise polynomial function in which the individual polynomials have the same degree and connect smoothly at join points whose abscissa values, called knots, are prespecified. You can use spline functions to fit curves to a wide variety of data.

A spline of degree 0 is a step function with steps located at the knots. A spline of degree 1 is a piecewise linear function where the lines connect at the knots. A spline of degree 2 is a piecewise quadratic curve whose values and slopes coincide at the knots. A spline of degree 3 is a piecewise cubic curve whose values, slopes, and curvature coincide at the knots. Visually, a cubic spline is a smooth curve, and it is the most commonly used spline when a smooth fit is desired. Note that when no knots are used, splines of degree d are simply polynomials of degree d .

More formally, suppose you specify knots $k_1 < k_2 < k_3 < \dots < k_n$. Then a spline of degree $d \geq 0$ is a function $S(x)$ with $d-1$ continuous derivatives such that

$$S(x) = \begin{cases} P_0(x) & x < k_1 \\ P_i(x) & k_i \leq x < k_{i+1}; i = 1, 2, \dots, n-1 \\ P_n(x) & x \geq k_n \end{cases}$$

where each $P_i(x)$ is a polynomial of degree d . The requirement that $S(x)$ has $d-1$ continuous derivatives is satisfied by requiring that the function values and all derivatives up to order $d-1$ of the adjacent polynomials at each knot match.

A counting argument yields the number of parameters that define a spline with n knots. There are $n+1$ polynomials of degree d , producing $(n+1)(d+1)$ coefficients. However, there are d restrictions at each of the n knots, so the number of free parameters is $(n+1)(d+1) - nd = n + d + 1$. In mathematical terminology this says that the dimension of the vector space of splines of degree d on n distinct knots is $n + d + 1$.

+ 1. If you have $n + d + 1$ basis vectors, then you can fit a curve to your data by regressing your dependent variable by using this basis for the corresponding design matrix columns. In this context, such a spline is known as a regression spline. The EFFECT statement provides a simple mechanism for obtaining such a basis.

If you remove the restriction that the knots of a spline must be distinct and allow repeated knots, then you can obtain functions that have less smoothness and even discontinuities at the repeated knot location. For a spline of degree d and a repeated knot that has multiplicity $m \leq d$, the piecewise polynomials that join such a knot are required to have only $d - m$ matching derivatives. Note that this increases the number of free parameters by $m - 1$ but also decreases the number of distinct knots by $m - 1$. Hence the dimension of the vector space of splines of degree d with n knots is still $n + d + 1$, provided that any repeated knot has a multiplicity less than or equal to d .

The EFFECT statement supports the commonly used *truncated power function* basis and *B-spline* basis. With exact arithmetic and by using the complete basis, you obtain the same fit with either of these bases. The following subsections provide details about constructing spline bases for the space of splines of degree d with n knots that satisfies $k_1 \leq k_2 \leq k_3 < \dots \leq k_n$.

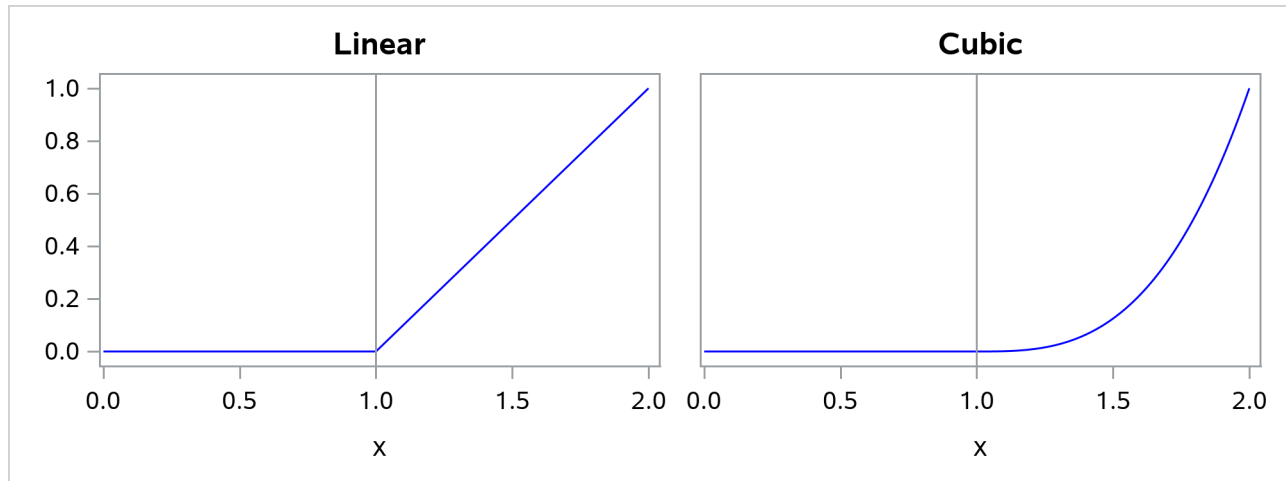
Truncated Power Function Basis

A truncated power function for a knot k_i is a function defined by

$$t_i(x) = \begin{cases} 0 & x < k_i \\ (x - k_i)^d & x \geq k_i \end{cases}$$

Figure 4.1 shows such functions for $d = 1$ and $d = 3$ with a knot at $x = 1$.

Figure 4.1 Truncated Power Functions with Knot at $x = 1$



The name is derived from the fact that these functions are shifted power functions that are truncated to 0 to the left of the knot. These functions are piecewise polynomial functions that have two pieces whose function values and derivatives of all orders up to $d-1$ are 0 at the defining knot. Hence these functions are splines of degree d . It is easy to see that these n functions are linearly independent. However, they do not form a basis, because such a basis requires $n+d-1$ functions. The usual way to add $d+1$ additional basis functions is to use the polynomials $1, x, x^2, \dots, x^d$. These $d+1$ functions together with the n truncated power functions $t_i(x), i = 1, 2, \dots, n$ form the truncated power basis.

Note that each time a knot is repeated, the associated exponent used in the corresponding basis function is reduced by 1. For example, for splines of degree d with three repeated knots $k_i = k_{i+1} = k_{i+2}$, the corresponding basis functions are $t_i(x) = (x - k_i)_+^d$, $t_{i+1}(x) = (x - k_i)_+^{d-1}$, and $t_{i+2}(x) = (x - k_i)_+^{d-2}$. Provided that the multiplicity of each repeated knot is less than or equal to the degree, this construction continues to yield a basis for the associated space of splines.

The main advantage of the truncated power function basis is the simplicity of its construction and the ease of interpreting the parameters in a model that corresponds to these basis functions. However, there are two weaknesses when you use this basis for regression. These functions grow rapidly without bound as x increases, resulting in numerical precision problems when the x data span a wide range. Furthermore, many or even all of these basis functions can be nonzero when evaluated at some x value, resulting in a design matrix that has few zeros and precludes the use of sparse matrix technology to speed up computation. This weakness can be addressed by using a B-spline basis.

B-Spline Basis

A B-spline basis can be built by starting with a set of Haar basis functions, which are functions that are 1 between adjacent knots and 0 elsewhere, and then applying a simple linear recursion relationship d times, yielding the $n + d + 1$ needed basis functions. For the purpose of building the B-spline basis, the n prespecified knots are called internal knots. This construction requires d additional knots, known as boundary knots, to be positioned to the left of the internal knots, and $\max(d, 1)$ boundary knots to be positioned to the right of the internal knots. The actual values of these boundary knots can be arbitrary. The EFFECT statement provides several methods for placing the necessary boundary knots, including the common method of using repeated values of the data extremes as the boundary knots. The boundary knot placement affects the precise form of the basis functions that are generated, but it does not affect the following two desirable properties:

- The B-spline basis functions are nonzero over an interval that spans at most $d + 2$ knots. This yields design matrix columns each of whose rows contain at most $d + 2$ adjacent nonzero entries.
- The computation of the basis functions at any x value is numerically stable and does not require evaluating powers of this value.

The following figures show the B-spline bases that are defined on $[0, 1]$ with four equally spaced internal knots at 0.2, 0.4, 0.6, and 0.8.

Figure 4.2 shows a linear B-spline basis. This basis consists of six functions, each of which is nonzero over an interval that spans at most three knots.

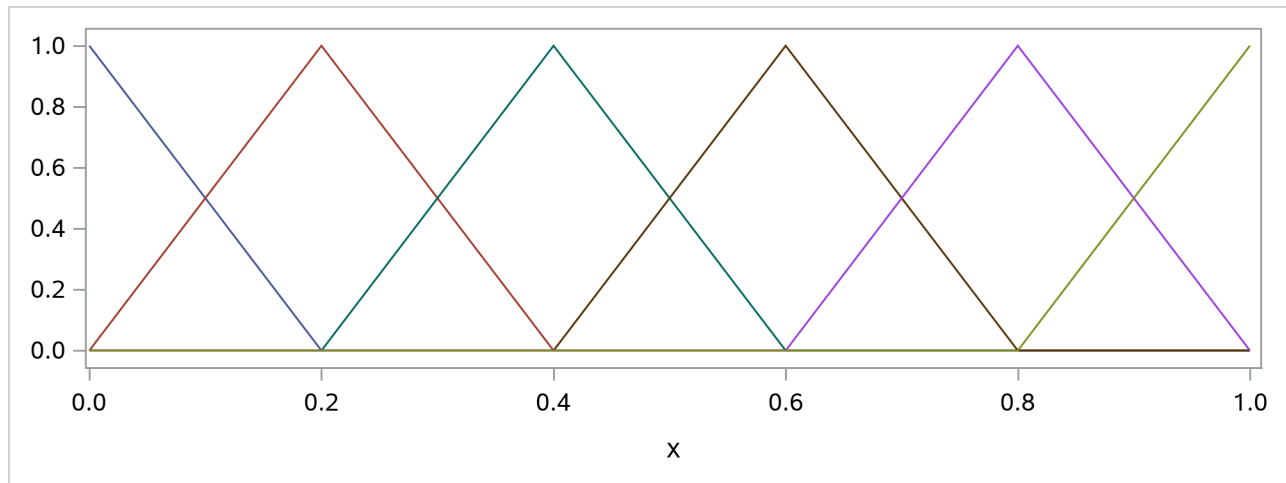
Figure 4.2 Linear B-Spline Basis with Four Equally Spaced Interior Knots

Figure 4.3 shows a cubic B-spline basis where the necessary boundary knots are positioned at $x = 0$ and $x = 1$. This basis consists of eight functions, each of which is nonzero over an interval that spans at most five knots.

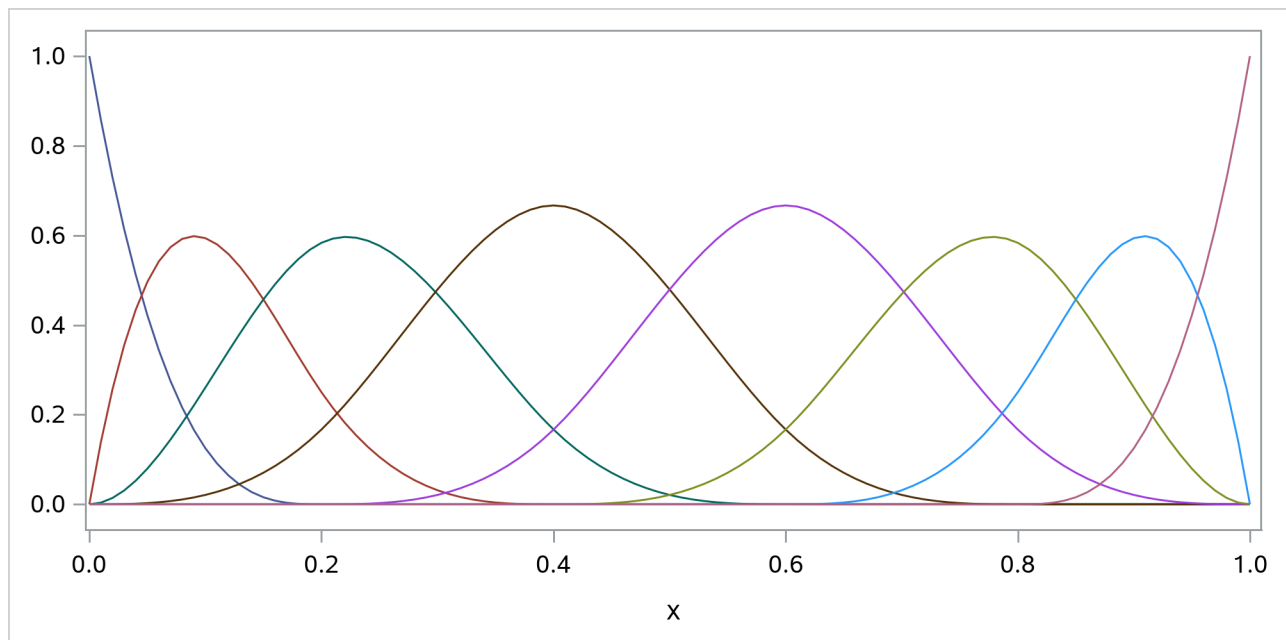
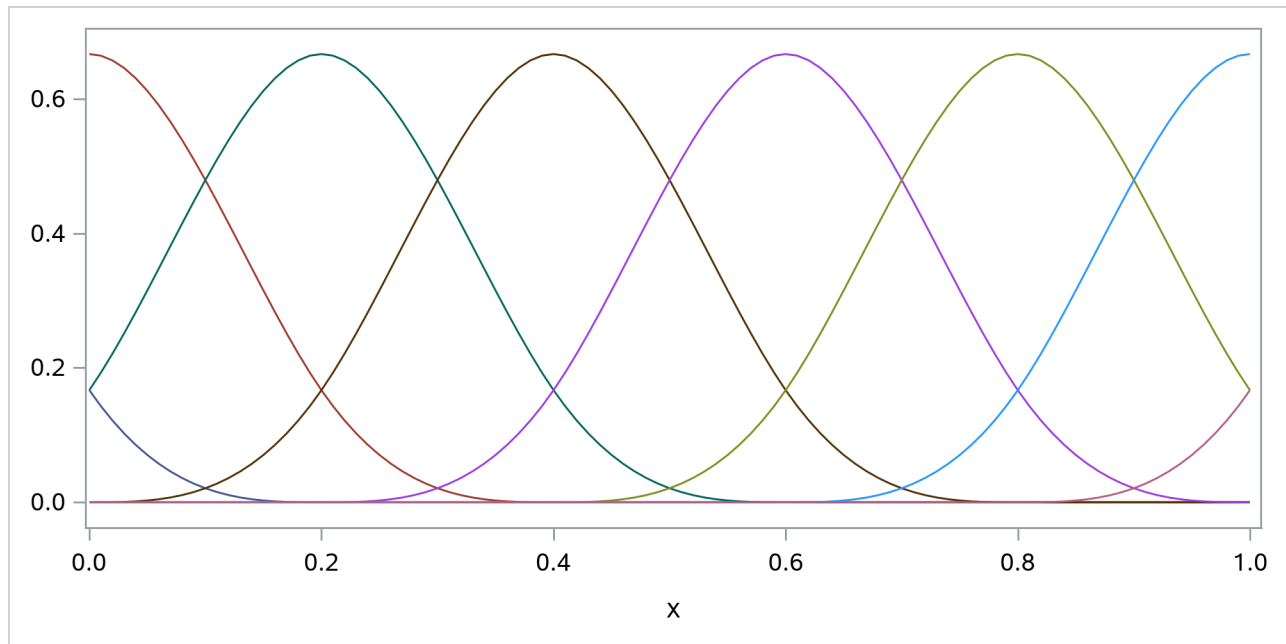
Figure 4.3 Cubic B-Spline Basis with Four Equally Spaced Interior Knots

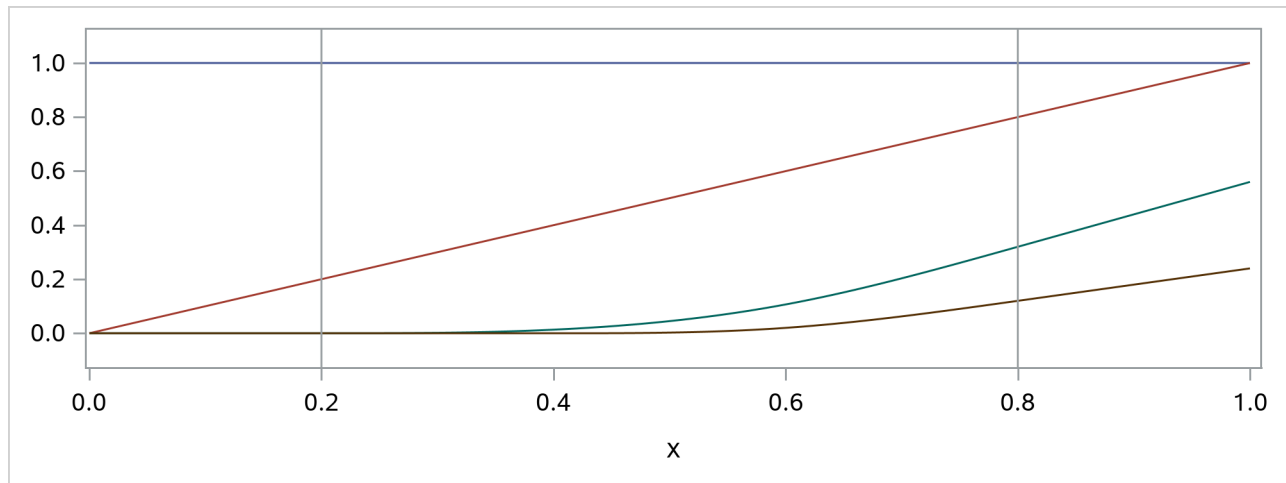
Figure 4.4 shows a different cubic B-spline basis where the necessary left-side boundary knots are positioned at -0.6 , -0.4 , -0.2 , and 0 . The right-side boundary knots are positioned at 1 , 1.2 , 1.4 , and 1.6 . As in the basis shown in Figure 4.3, this basis consists of eight functions, each of which is nonzero over an interval that spans at most five knots. The different positioning of the boundary knots has merely changed the shape of the individual basis functions.

Figure 4.4 Cubic B-Spline Basis with Equally Spaced Boundary and Interior Knots

For more information about this construction, see Hastie, Tibshirani, and Friedman (2001).

Natural Cubic Spline Basis

Natural cubic splines are cubic splines with the additional restriction that the splines are required to be linear beyond the extreme knots. Some authors prefer the terminology “restricted cubic splines” to “natural cubic splines.” The space of unrestricted cubic splines on n knots has the dimension $n + 4$. Imposing the restrictions that the cubic polynomials beyond the first and last knot reduce to linear polynomials reduces the number of degrees of freedom by 4, so a basis for the natural cubic splines consists of n functions. Starting from the truncated power function basis for the unrestricted cubic splines, you can obtain a reduced basis by imposing linearity constraints. For more information about this construction, see Hastie, Tibshirani, and Friedman (2001). [Figure 4.5](#) shows this natural cubic spline basis defined on $[0, 1]$ with four equally spaced internal knots at 0.2, 0.4, 0.6, and 0.8. This basis consists of four basis functions that are all linear beyond the extreme knots at 0.2 and 0.8.

Figure 4.5 Natural Cubic Spline Basis with Four Equally Spaced Knots

ODS Table Names

Each constructed effect produces one or more informative tables when the DETAILS option is also specified. Each table has a name associated with it. This name is used to refer to the table by procedures that support a DISPLAY or DISPLAYOUT statement. You also use this name to refer to the table in ODS statements. These names are listed in [Table 4.5](#).

Table 4.5 ODS Tables Produced by the EFFECT Statement

Table Name	Description	Option
BSplineDetails	B-spline basis details	BASIS=BSPLINE
CollectionLevelInfo	Levels of collection effects	
MMLevelInfo	Levels of multimember effects	
PolyDetails	Number of variables and columns, polynomial degree, and standardization method	
PolyScaling	Centering and scaling details	STANDARDIZE
SplineKnots	knot and boundary knot values	
TPFSplineDetails	Truncated power function spline basis details	BASIS=TPF or NATURALCUBIC

SELECTION Statement

SELECTION < **METHOD**=*method*< (*method-options*)> >> *options*> ;

This section applies to the following procedures: CNTSELECT and SEVSELECT.

Procedures in this book that support model selection use the SELECTION statement to control details about the model selection process.

You can specify the following methods in the SELECTION statement:

METHOD=*method* < (*method-options*)>

specifies the method to be used to select the model. You can also specify *method-options* to apply to the specified method by enclosing them in parentheses after the *method*.

The following *methods* are available and are explained in detail in the section “[Model Selection Methods](#)” on page 102. By default, METHOD=STEPWISE.

BACKWARD	specifies backward elimination. This method starts with all effects in the model and deletes effects.
FORWARD	specifies forward selection. This method starts with no effects in the model and adds effects.
FORWARDSSWAP	specifies forward-swap selection, which is an extension of the forward selection method. Before any addition step, the procedure makes all pairwise swaps of one effect in the model and one effect out of the current model that improve the selection criterion.
MCP	specifies the MCP method, which minimizes ordinary least squares plus the mini-max concave penalty (MCP) function. As an alternative to the convex penalized methods such as LASSO and ELASTICNET, MCP is one of the nonconvex penalized methods.
NONE	specifies no model selection.
SCAD	specifies the SCAD method, which minimizes ordinary least squares plus the smoothly clipped absolute deviation (SCAD) function. As an alternative to the convex penalized methods such as LASSO and ELASTICNET, SCAD is one of the nonconvex penalized methods.
STEPWISE	specifies stepwise regression. This method is similar to the FORWARD method except that effects already in the model do not necessarily stay there.

[Table 4.6](#) lists the applicable *method-options* for each of these methods.

Table 4.6 Applicable *method-options* by *method*

<i>method-option</i>	FORWARD	BACKWARD	STEPWISE
CHOOSE =	x	x	x
COMPETITIVE			x
CRITERION =	x	x	x
MAXEFFECTS =	x		x
MAXSTEPS =	x	x	x
MINEFFECTS =		x	
SELECT =	x	x	x
STOP =	x	x	x

The syntax of the *method-options* that you can specify in parentheses after the **SELECTION=** option *method* follows. As described in Table 4.6, not all selection *method-options* are applicable to every *method*.

CHOOSE=criterion

chooses from the list of models (at each step of the selection process) the model that yields the best value of the specified criterion. If the optimal value of the specified criterion occurs for models at more than one step, then the model that has the smallest number of parameters is chosen. If you omit the **CHOOSE=** *method-option*, then the selected model is the model at the final step in the selection process. The criteria that are supported depend on the type of model that is being fit. For the supported values of *criterion*, see the chapters for the relevant procedures.

COMPETITIVE

is applicable as a *method-option* only when **METHOD=STEPWISE**. If you specify the **COMPETITIVE** *method-option*, then the **SELECT=** criterion is evaluated for all models in which an effect currently in the model is dropped or an effect not yet in the model is added. The effect whose removal from or addition to the model yields the maximum improvement to the **SELECT=** criterion is dropped or added.

CRITERION=criterion

is an alias for the **SELECT=** *method-option*.

MAXEFFECTS=n

specifies the maximum number of effects in any model that is considered during the selection process. This *method-option* is ignored when you specify **METHOD=BACKWARD**. If, at some step of the selection process, the model contains the specified maximum number of effects, then no candidates for addition are considered.

MAXSTEPS=n

specifies the maximum number of selection steps to perform. The default value of *n* is the number of effects in the **MODEL** statement when **METHOD=FORWARD** or **METHOD=BACKWARD**. The default is three times the number of effects when **METHOD=STEPWISE**.

MINEFFECTS=*n*

specifies the minimum number of effects in any model that is considered during backward selection. This *method-option* is ignored unless METHOD=BACKWARD is specified. The backward selection process terminates if, at some step of the selection process, the model contains the specified minimum number of effects.

SELECT=*criterion*

specifies the criterion that the procedure uses to determine the order in which effects enter or leave at each step of the selection method. For each step, the effect whose addition to or removal from the current model yields the maximum improvement in the specified criterion is selected. For supported criteria, see the chapter for the relevant procedure.

STOP=NONE | *criterion*

specifies a criterion that is used to stop the selection process. The criteria that are supported depend on the type of model that is being fit. For information about the supported criteria, see the chapter for the relevant procedure.

If you omit this *method-option* but specify the **SELECT= *method-option***, then the criterion specified in the **SELECT= *method-option*** is also used as the criterion for the **STOP= *method-option***.

You can specify the following values:

NONE

stops the selection process if no suitable add or drop candidates can be found or if a size-based limit is reached. For example, if you specify STOP=NONE MAXEFFECTS=5, then the selection process stops at the first step that produces a model that has five effects.

criterion

stops the selection process if the selection process produces a local extremum of this *criterion* or if a size-based limit is reached. For example, if you specify STOP=AIC MAXSTEPS=5, then the selection process stops before step 5 if the sequence of models has a local minimum of the AIC criterion before step 5. The determination of whether a local minimum is reached is made on the basis of a stop horizon. The default stop horizon is 3, but you can change it by using the **STOPHORIZON=** option. If the stop horizon is *n* and the STOP=*criterion* at any step is better than the stop criterion at the next *n* steps, then the selection process terminates.

In addition, you can also specify the following *options*:

DETAILS=NONE | SUMMARY | ALL**DETAILS=STEPS<(CANDIDATES(ALL | *n*))>**

specifies the level of detail to be produced about the selection process. The default is DETAILS=SUMMARY.

The DETAILS=ALL and DETAILS=STEPS options produce the following output:

- tables that provide information about the model that is selected at each step of the selection process.
- entry and removal statistics for inclusion or exclusion candidates at each step. By default, only the top 10 candidates at each step are shown. If you specify STEPS(CANDIDATES(*n*)), then the

best n candidates are shown. If you specify STEPS(CANDIDATES(ALL)), then all candidates are shown.

- a selection summary table that shows by step the effect that is added to or removed from the model in addition to the values of the SELECT, STOP, and CHOOSE criteria for the resulting model.
- a stop reason table that describes why the selection process stopped.
- a selection reason table that describes why the selected model was chosen.
- a selected effects table that lists the effects that are in the selected model.

The DETAILS=SUMMARY option produces only the selection summary, stop reason, selection reason, and selected effects tables. In addition, for METHOD=ELASTICNET with neither a specified L2 value nor a solver specification, the DETAILS=SUMMARY option also displays an elastic net summary table, which shows the ridge regularization parameter L2 (in the ratio scaling) and CHOOSE criterion values in each L2 search step.

HIERARCHY=NONE | SINGLE | SINGLECLASS

specifies whether and how to apply the model hierarchy requirement. You can specify that only classification effects, or both classification and continuous effects, be subject to the hierarchy requirement. This option is ignored unless you also specify one of the following options: [METHOD=FORWARD](#), [METHOD=BACKWARD](#), or [METHOD=STEPWISE](#).

Model hierarchy refers to the requirement that, for any term to be in the model, all model effects that are contained in the term must be present in the model. For example, in order for the interaction A*B to enter the model, the main effects A and B must be in the model. Likewise, neither effect A nor effect B can leave the model while the interaction A*B is in the model.

You can specify the following values:

NONE	specifies that model hierarchy not be maintained. Any single effect can enter or leave the model at any step of the selection process.
SINGLE	specifies that only one effect enter or leave the model at one time, subject to the model hierarchy requirement. For example, suppose that the model contains the main effects A and B and the interaction A*B. In the first step of the selection process, either A or B can enter the model. In the second step, the other main effect can enter the model. The interaction effect can enter the model only when both main effects have already entered. Also, before A or B can be removed from the model, the A*B interaction must first be removed. All effects (CLASS and interval) are subject to the hierarchy requirement.
SINGLECLASS	is the same as HIERARCHY=SINGLE except that only CLASS effects are subject to the hierarchy requirement.

By default, HIERARCHY=NONE.

ORDERSELECT

specifies that effects in the selected model are displayed in the order in which they first entered the model. If you do not specify the ORDERSELECT option, then effects in the selected model are displayed in the order in which they appear in the SCALEMODEL or MODEL statement.

PLOT <(global-plot-options)> <= plot-request <(options)>>

PLOTS <(global-plot-options)> <= (plot-request <(options)> <...plot-request <(options)>>>

controls the selection process plots that are produced through ODS Graphics. The only procedure in this book that supports this option is the SEVSELECT procedure.

When you specify only one plot request, you can omit the parentheses around it. Here are some examples:

```
plots=all
plots=coefficients(unpack)
plots(unpack)=(coefficients criteria)
```

ODS Graphics must be enabled before you can request plots. For example:

```
ods graphics on;

proc sevselect;
  model y = x1-x100;
  selection method=forward plots=all;
run;

ods graphics off;
```

Global Plot Options

Each *global-plot-option* applies to all plots that the SELECTION statement generates, unless you alter the option by using a specific plot option that follows a *plot-request*. You can specify the following *global-plot-options*:

ENDSTEP=*n*

terminates the step ranges shown on the horizontal axes of plots at the specified step. By default, the step range that is shown terminates at the final step of the selection process.

MAXPARMLABEL=*n*

specifies the maximum number of characters beyond which all parameter labels on the coefficient progression panel and the coefficient progression plot are suppressed. By default, MAXPARMLABEL=20.

MAXSTEPLABEL=*n*

specifies the maximum number of characters of effect labels to display on the horizontal axes of the plots. By default, MAXSTEPLABEL=20. If the length of any effect label exceeds *n*, then step numbers are used on the horizontal axes of the plots.

STARTSTEP=*n*

specifies that the step ranges shown on the horizontal axes of plots start at the specified step. By default, the step range that is shown starts at the initial step of the selection process.

STEPAXIS=EFFECT | NUMBER

specifies the horizontal axis to be used on the plots, where this axis represents the sequence of entering or departing effects. You can specify the following values:

- | | |
|---------------|---|
| EFFECT | requests that each step be labeled by a prefix followed by the name of the effect that enters or leaves at that step. The prefix consists of the step number, followed by a “+” or a “-” sign, depending on whether the effect enters (+) or leaves (-) at that step. |
| NUMBER | requests that each step be labeled by the step number. |

UNPACK | UNPACKPANEL

suppresses paneling. By default, multiple plots can appear in the coefficient progression panel and the criteria panel. Specify UNPACK to see each plot individually. You can also specify UNPACK as a suboption of the CRITERIA and COEFFICIENTS options.

Specific Plot Options

You can specify the following *plot-requests* and their *options*:

ALL

produces all relevant plots.

COEFFICIENTS | COEFFICIENTPANEL <(UNPACK | UNPACKPANEL)>

creates a panel of two plots. The upper plot shows the progression of the parameter values as the selection process proceeds. The lower plot shows the progression of the CHOOSE= criterion. If no CHOOSE= criterion is in effect, then only the coefficient progression is shown. If you specify the UNPACK option, then the coefficient progression and the CHOOSE= criterion progression are shown in separate plots.

CRITERIA | CRITERIONPANEL <(UNPACK | UNPACKPANEL)>

creates a panel of model fit criteria. If you specify the UNPACK option, then each criterion progression is shown in a separate plot.

NONE

suppress all selection process plots, even if you specify other plot options.

SELECTION=method <(method-options)>

is an alias for the **METHOD=** option.

STOPHORIZON=n

specifies the number of consecutive steps at which the **STOP=** criterion must worsen in order for a local extremum to be detected. For example, suppose that **STOP=AIC** and the sequence of AIC values at steps 1 to 6 of a selection are 10, 7, 4, 6, 5, 2. If **STOPHORIZON=2**, then the AIC criterion is deemed to have a local minimum at step 3 because the AIC value at the next two steps are greater than the value 4 that occurs at step 3. However, if **STOPHORIZON=3**, then the value at step 3 is not deemed to be a local minimum because the AIC value at step 6 is lower than the AIC value at step 3. If you specify **STOP=NONE** then the stop horizon value is ignored. If you specify **STOP=SL**, then n is ignored and **STOPHORIZON=1** is used. By default, **STOPHORIZON=3** unless otherwise stated in individual procedure chapters.

Optimization Options

This section applies to the SEVSELECT procedure.

This section describes options that are typically available in the PROC statement or the NLOPTIONS statement of the procedures in this book that perform optimizations. The following notation is used to describe the options. $\boldsymbol{\beta}$ denotes the $p \times 1$ vector of parameters for the optimization and β_i is its i th element. The objective function being minimized, its $p \times 1$ gradient vector, and its $p \times p$ Hessian matrix are denoted as $f(\boldsymbol{\beta})$, $\mathbf{g}(\boldsymbol{\beta})$, and $\mathbf{H}(\boldsymbol{\beta})$, respectively. The gradient with respect to the i th parameter is denoted as $g_i(\boldsymbol{\beta})$. Superscripts in parentheses denote the iteration count; for example, $f(\boldsymbol{\beta})^{(k)}$ is the value of the objective function at iteration k .

ABSCONV=r

ABSTOL=r

specifies an absolute function convergence criterion. For minimization, termination requires $f(\boldsymbol{\beta})^{(k)} \leq r$, where $\boldsymbol{\beta}$ is the vector of parameters in the optimization and $f(\cdot)$ is the objective function. The default value of r is the negative square root of the largest double-precision value, which serves only as a protection against overflows.

ABSFCNV=r

ABSFTOL=r

specifies an absolute function difference convergence criterion. For all techniques except NMSIMP, termination requires a small change of the function value in successive iterations:

$$|f(\boldsymbol{\beta}^{(k-1)}) - f(\boldsymbol{\beta}^{(k)})| \leq r$$

Here, $\boldsymbol{\beta}$ is the vector of parameters in the optimization and $f(\cdot)$ is the objective function. The same formula is used for the NMSIMP technique, but $\boldsymbol{\beta}^{(k)}$ is defined as the vertex that has the lowest function value and $\boldsymbol{\beta}^{(k-1)}$ is defined as the vertex that has the highest function value in the simplex. By default, **ABSFCNV=0**.

ABSGCONV= r **ABSGTOL= r**

specifies an absolute gradient convergence criterion. Termination requires the maximum absolute gradient element to be small:

$$\max_j |g_j(\boldsymbol{\beta}^{(k)})| \leq r$$

Here, $\boldsymbol{\beta}$ is the vector of parameters in the optimization and $g_j(\cdot)$ is the gradient of the objective function with respect to the j th parameter. This criterion is not used by the NMSIMP technique. By default, ABSGCONV=1E-5.

ABSXCONV= r **ABSXTOL= r**

specifies an absolute parameter convergence criterion: For all techniques except NMSIMP, termination requires a small Euclidean distance between successive parameter vectors,

$$\|\boldsymbol{\beta}^{(k)} - \boldsymbol{\beta}^{(k-1)}\|_2 \leq r$$

For the NMSIMP technique, termination requires either a small length $\alpha^{(k)}$ of the vertices of a restart simplex,

$$\alpha^{(k)} \leq r$$

or a small simplex size,

$$\delta^{(k)} \leq r$$

where the simplex size $\delta^{(k)}$ is defined as the L1 distance from the simplex vertex $\boldsymbol{\xi}^{(k)}$ that has the smallest function value to the other p simplex points $\boldsymbol{\beta}_l^{(k)} \neq \boldsymbol{\xi}^{(k)}$:

$$\delta^{(k)} = \sum_{\boldsymbol{\beta}_l \neq \boldsymbol{\xi}} \|\boldsymbol{\beta}_l^{(k)} - \boldsymbol{\xi}^{(k)}\|_1$$

The default is $r = 1\text{E}-8$ for the NMSIMP technique and $r = 0$ otherwise.

FCONV= r **FTOL= r**

specifies a relative function difference convergence criterion. For all techniques except NMSIMP, termination requires a small relative change of the function value in successive iterations,

$$\frac{|f(\boldsymbol{\beta}^{(k)}) - f(\boldsymbol{\beta}^{(k-1)})|}{\max(|f(\boldsymbol{\beta}^{(k-1)})|, \text{FSIZE})} \leq r$$

where FSIZE is defined by the **FSIZE=** option. Here, $\boldsymbol{\beta}$ denotes the vector of parameters that participate in the optimization, and $f(\cdot)$ is the objective function. The same formula is used for the NMSIMP technique, but $\boldsymbol{\beta}^{(k)}$ is defined as the vertex that has the lowest function value and $\boldsymbol{\beta}^{(k-1)}$ is defined as the vertex that has the highest function value in the simplex.

The default value is $r=2 \times \epsilon$ where ϵ is the machine precision, which is the smallest double-precision floating-point number such that $1 + \epsilon > 1$.

FCONV2=r**FTOL2=r**

specifies a second function convergence criterion. For all techniques except NMSIMP, termination requires a small predicted reduction of the objective function:

$$df^{(k)} \approx f(\boldsymbol{\beta}^{(k)}) - f(\boldsymbol{\beta}^{(k)} + \mathbf{s}^{(k)})$$

The predicted reduction

$$\begin{aligned} df^{(k)} &= -\mathbf{g}^{(k)'} \mathbf{s}^{(k)} - \frac{1}{2} \mathbf{s}^{(k)'} \mathbf{H}^{(k)} \mathbf{s}^{(k)} \\ &= -\frac{1}{2} \mathbf{s}^{(k)'} \mathbf{g}^{(k)} \leq r \end{aligned}$$

is computed by approximating the objective function f by the first two terms of the Taylor series and substituting the Newton step,

$$\mathbf{s}^{(k)} = -[\mathbf{H}^{(k)}]^{-1} \mathbf{g}^{(k)}$$

For the NMSIMP technique, termination requires a small standard deviation of the function values of the $p + 1$ simplex vertices $\boldsymbol{\beta}_l^{(k)}$, $l = 0, \dots, p$,

$$\sqrt{\frac{1}{n+1} \sum_l \left[f(\boldsymbol{\beta}_l^{(k)}) - \bar{f}(\boldsymbol{\beta}^{(k)}) \right]^2} \leq r$$

where $\bar{f}(\boldsymbol{\beta}^{(k)}) = \frac{1}{p+1} \sum_l f(\boldsymbol{\beta}_l^{(k)})$. If there are p_{act} boundary constraints active at $\boldsymbol{\beta}^{(k)}$, the mean and standard deviation are computed only for the $n + 1 - p_{act}$ unconstrained vertices.

The default value is $r = 1\text{E-}6$ for the NMSIMP technique and $r = 0$ otherwise.

FSIZE=r

specifies the FSIZE parameter of the relative function and relative gradient termination criteria. The default value is $r = 0$. For more information, see the FCONV= and GCONV= options.

GCONV=r**GTOL=r**

specifies a relative gradient convergence criterion. For all techniques except CONGRA and NMSIMP, termination requires that the normalized predicted function reduction be small:

$$\frac{\mathbf{g}(\boldsymbol{\beta}^{(k)})' [\mathbf{H}^{(k)}]^{-1} \mathbf{g}(\boldsymbol{\beta}^{(k)})}{\max(|f(\boldsymbol{\beta}^{(k)})|, \text{FSIZE})} \leq r$$

where FSIZE is defined by the FSIZE= option. Here, $\boldsymbol{\beta}$ denotes the vector of parameters that participate in the optimization, $f(\cdot)$ is the objective function, and $\mathbf{g}(\cdot)$ is the gradient. For the CONGRA technique (where a reliable Hessian estimate $\mathbf{H}$ is not available), the following criterion is used:

$$\frac{\|\mathbf{g}(\boldsymbol{\beta}^{(k)})\|_2 \|\mathbf{s}(\boldsymbol{\beta}^{(k)})\|_2}{\|\mathbf{g}(\boldsymbol{\beta}^{(k)}) - \mathbf{g}(\boldsymbol{\beta}^{(k-1)})\|_2 \max(|f(\boldsymbol{\beta}^{(k)})|, \text{FSIZE})} \leq r$$

This criterion is not used by the NMSIMP technique. By default, GCONV=1E-8.

GCONV2=*r***GTOL2=*r***

specifies another relative gradient convergence criterion. For the TRUREG, NRRIDG, and NEWRAP techniques, the following criterion of Browne (1982) is used:

$$\max_j \frac{|\mathbf{g}_j(\boldsymbol{\beta}^{(k)})|}{\sqrt{f(\boldsymbol{\beta}^{(k)})\mathbf{H}_{j,j}^{(k)}}} \leq r$$

This criterion is not used by the other techniques.

By default, GCONV2=0.

MAXFUNC=*n***MAXFU=*n***

specifies the maximum number *n* of function calls in the optimization process. The default values are as follows, depending on the optimization technique:

- TRUREG, NRRIDG, and NEWRAP: 125
- QUANEW and DBLDOG: 500
- CONGRA: 1,000
- NMSIMP: 3,000

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually performed can exceed the number that is specified by this option. You can specify the optimization technique in the **TECHNIQUE=** option.

MAXITER=*n***MAXIT=*n***

specifies the maximum number *n* of iterations in the optimization process. The default values are as follows, depending on the optimization technique:

- TRUREG, NRRIDG, and NEWRAP: 50
- QUANEW and DBLDOG: 200
- CONGRA: 400
- NMSIMP: 1,000

These default values also apply when *n* is specified as a missing value. You can specify the optimization technique in the **TECHNIQUE=** option.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. The time specified by *r* is checked only once at the end of each iteration. Therefore, the actual running time can be longer than *r*. The default value is the largest floating-point double representation of your computer.

MINITER=*n***MINIT**=*n*

specifies the minimum number of iterations. If you request more iterations than are actually needed for convergence to a stationary point, the optimization algorithms can behave strangely. For example, the effect of rounding errors can prevent the algorithm from continuing for the required number of iterations. By default, MINITER=0.

TECHNIQUE=*technique***TECH**=*technique*

specifies the optimization technique for obtaining maximum likelihood estimates. You can specify one of the following *techniques*:

CONGRA	performs a conjugate-gradient optimization.
DBLDOG	performs a version of double-dogleg optimization.
NEWRAP	performs a Newton-Raphson optimization with line search.
NMSIMP	performs a Nelder-Mead simplex optimization.
NONE	performs no optimization.
NRRIDG	performs a Newton-Raphson optimization with ridging.
QUANEW	performs a dual quasi-Newton optimization.
TRUREG	performs a trust-region optimization

The default method varies by the procedure. For the SEVSELECT procedure, the default is TECHNIQUE=TRUREG.

For more information, see the section “[Choosing an Optimization Algorithm](#)” on page 108.

XCONV=*r***XTOL**=*r*

specifies the relative parameter convergence criterion. Convergence requires a small relative parameter change in subsequent iterations,

$$\frac{\max_j |\beta_j^{(k)} - \beta_j^{(k-1)}|}{\max(|\beta_j^{(k)}|, |\beta_j^{(k-1)}|, \text{XSIZE})} \leq r$$

where XSIZE is defined by the **XSIZE**= option. and $\beta_j^{(i)}$ is the estimate of the *j*th parameter at iteration *i*. For the NMSIMP technique, the same formula is used, but $\beta_j^{(k)}$ is defined as the vertex that has the lowest function value and $\beta_j^{(k-1)}$ is defined as the vertex that has the highest function value in the simplex. The default value is $r = 1\text{E}-8$ for the NMSIMP technique and $r = 0$ otherwise.

XSIZE=*r*

specifies the XSIZE parameter of the relative parameter termination criterion. The value of *r* must be greater than or equal to 0; the default is $r = 0$. For more information, see the XCONV= option.

Details for SAS Econometrics Procedures

Levelization of Classification Variables

This section applies to the following procedures: CNTSELECT, CPANEL, CQLIM, CSPATIALREG, and SEVSELECT.

A classification variable enters the statistical analysis or model not through its values but through its levels. The process of associating values of a variable with levels is called *levelization*.

During the process of levelization, observations that share the same value are assigned to the same level. The manner in which values are grouped can be affected by the inclusion of formats. The sort order of the levels can be determined by specifying the ORDER= option in the procedure statement. In procedures in this book, you can also control the sorting order separately for each variable in the CLASS statement.

Consider the data on nine observations in Table 4.7. The variable A is integer-valued, and the variable X is a continuous variable that has a missing value for the fourth observation. The fourth and fifth columns of Table 4.7 apply two different formats to the variable X.

Table 4.7 Example Data for Levelization

Obs	A	x	FORMAT x 3.0	FORMAT x 3.1
1	2	1.09	1	1.1
2	2	1.13	1	1.1
3	2	1.27	1	1.3
4	3	.	.	.
5	3	2.26	2	2.3
6	3	2.48	2	2.5
7	4	3.34	3	3.3
8	4	3.34	3	3.3
9	4	3.14	3	3.1

By default, levelization of the variables groups the observations by the formatted value of the variable, except for numerical variables for which no explicit format is provided. Numerical variables for which no explicit format is provided are sorted by their internal value. The levelization of the four columns in Table 4.7 leads to the level assignment in Table 4.8.

Table 4.8 Values and Levels

Obs	A		X		FORMAT x 3.0		FORMAT x 3.1	
	Value	Level	Value	Level	Value	Level	Value	Level
1	2	1	1.09	1	1	1	1.1	1

Table 4.8 *continued*

Obs	A		X		FORMAT x 3.0		FORMAT x 3.1	
	Value	Level	Value	Level	Value	Level	Value	Level
2	2	1	1.13	2	1	1	1.1	1
3	2	1	1.27	3	1	1	1.3	2
4	3	2	.	.	.	.	.	.
5	3	2	2.26	4	2	2	2.3	3
6	3	2	2.48	5	2	2	2.5	4
7	4	3	3.34	7	3	3	3.3	6
8	4	3	3.34	7	3	3	3.3	6
9	4	3	3.14	6	3	3	3.1	5

The sort order for the levels of CLASS variables can be specified in the **ORDER=** option in the **CLASS** statement.

When **ORDER=FORMATTED** (which is the default) is in effect for numeric variables for which you have supplied no explicit format, the levels are ordered by their internal values. To order numeric classification levels that have no explicit format by their BEST12. formatted values, you can specify the BEST12. format explicitly for the CLASS variables.

Table 4.9 shows how values of the **ORDER=** option are interpreted.

Table 4.9 Interpretation of Values of **ORDER=** Option

Value of ORDER=	Levels Sorted By
FORMATTED	External formatted value, except for numeric variables that have no explicit format, which are sorted by their unformatted (internal) value. The sort order is machine-dependent.
FREQ	Descending frequency count (levels that have the most observations come first in the order)
INTERNAL	Unformatted value. The sort order is machine-dependent.

For more information about sort order, see the chapter about the SORT procedure in the *Base SAS Procedures Guide* and the discussion of BY-group processing in the “Grouping Data” section of *SAS Programmers Guide: Essentials*.

When the **MISSING** option is specified in the **CLASS** statement, the missing values (‘.’ for a numeric variable and blanks for a character variable) are included in the levelization and are assigned a level. Table 4.10 displays the results of levelizing the values in Table 4.7 when the **MISSING** option is in effect.

Table 4.10 Values and Levels When the MISSING Option Is Specified

Obs	A		X		FORMAT x 3.0		FORMAT x 3.1	
	Value	Level	Value	Level	Value	Level	Value	Level
1	2	1	1.09	2	1	2	1.1	2
2	2	1	1.13	3	1	2	1.1	2
3	2	1	1.27	4	1	2	1.3	3
4	3	2	.	1	.	1	.	1
5	3	2	2.26	5	2	3	2.3	4
6	3	2	2.48	6	2	3	2.5	5
7	4	3	3.34	8	3	4	3.3	7
8	4	3	3.34	8	3	4	3.3	7
9	4	3	3.14	7	3	4	3.1	6

When the MISSING option is not specified, it is important to understand the implications of missing values for your statistical analysis. When a procedure in this book levelizes the CLASS variables, an observation for which any CLASS variable has a missing value is excluded from the analysis.

Procedures in this book print a “Number of Observations” table that shows the number of observations that are read from the data set and the number of observations that are used in the analysis. Pay careful attention to this table—especially when your data table contains missing values—to ensure that no observations are unintentionally excluded from the analysis.

Specification and Parameterization of Model Effects

This section applies to the following procedures: CNTSELECT, CPANEL, CQLIM, CSPATIALREG, HMM, and SEVSELECT.

Procedures in this book that have a MODEL or SCALEMODEL statement support the formation of effects. An *effect* is an element in a linear model structure that is formed from one or more variables. At some point the statistical representations of these models involve linear structures such as

$$\mathbf{X}\boldsymbol{\beta}$$

or

$$\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\boldsymbol{\gamma}$$

The model matrices $\mathbf{X}$ and $\mathbf{Z}$ are formed according to effect-construction rules.

Procedures that also have a CLASS statement support the rich set of effects that is discussed in this section.

Procedures that also have an EFFECT statement enable you to construct special *constructed* effects that are discussed in “EFFECT Statement” on page 62.

In order to correctly interpret the results from a statistical analysis, you need to understand how construction (*parameterization*) rules apply to regression-type models, whether these are scale regression models as in the SEVSELECT procedure or count regression models as in the CNTSELECT procedures.

Effects are specified by a special notation that uses variable names and operators. There are two types of variables: classification (or CLASS) variables and continuous variables. *Classification variables* can be either numeric or character and are specified in a CLASS statement. For more information, see the section “[Levelization of Classification Variables](#)” on page 89. An independent variable that is not declared in the CLASS statement is assumed to be *continuous*. For example, the heights and weights of subjects are continuous variables.

Two primary operators (crossing and nesting) are used for combining the variables, and several additional operators are used to simplify effect specification. Operators are discussed in the section “[Effect Operators](#)” on page 92.

Procedures in this book that have a CLASS statement support a general linear model (GLM) parameterization and might also support nonsingular parameterizations for the classification variables. The GLM parameterization, commonly called *dummy* parameterization, is the default for all procedures in this book. For more information, see the sections “[GLM Parameterization of Classification Variables and Effects](#)” on page 94 and “[Nonsingular Parameterization](#)” on page 99.

NOTE: This section uses the MODEL statement in all examples for illustration purposes. All discussion also applies to the SCALEMODEL statement in the SEVSELECT procedure (where you specify only the regression effects) and to other types of model specification statements in other procedures (such as the ZEROMODEL statement in the CNTSELECT procedure).

Effect Operators

Table 4.11 summarizes the operators that are available for selecting and constructing effects. These operators are discussed in the following sections.

Table 4.11 Available Effect Operators

Operator	Example	Description
Interaction	A*B	Crosses the levels of the effects
Nesting	A(B)	Nests A levels within B levels
Bar operator	A B C	Specifies all interactions
At sign operator	A B C@2	Reduces interactions in bar effects
Dash operator	A1-A10	Specifies sequentially numbered variables
Colon operator	A:	Specifies variables with common prefix
Double dash operator	A- -C	Specifies sequential variables in data set order

Bar and At Sign Operators

You can shorten the specification of a large factorial model by using the bar operator. For example, two ways of writing the model for a full three-way factorial model follow:

```
model Y = A B C A*B A*C B*C A*B*C;
```

```
model Y = A|B|C;
```

When the bar (|) is used, the right and left sides become effects, and the cross of them becomes an effect. Multiple bars are permitted. The expressions are expanded from left to right, using rules 2–4 in Searle (1971, p. 390).

- Multiple bars are evaluated from left to right. For example, $A | B | C$ is evaluated as follows:

$$\begin{aligned} A | B | C &\rightarrow \{ A | B \} | C \\ &\rightarrow \{ A B A*B \} | C \\ &\rightarrow A B A*B C A*C B*C A*B*C \end{aligned}$$

- Crossed and nested groups of variables are combined. For example, $A(B) | C(D)$ generates $A*C(B D)$, among other terms.
- Duplicate variables are removed. For example, $A(C) | B(C)$ generates $A*B(C C)$, among other terms, and the extra C is removed.
- Effects are discarded if a variable occurs on both the crossed and nested parts of an effect. For example, $A(B) | B(D E)$ generates $A*B(B D E)$, but this effect is eliminated immediately.

You can also specify the maximum number of variables involved in any effect that results from bar evaluation by specifying that maximum number, preceded by an at sign (@), at the end of the bar effect. For example, the following specification selects only those effects that contain two or fewer variables:

```
model Y = A|B|C@2;
```

The preceding example is equivalent to specifying the following MODEL statement:

```
model Y = A B C A*B A*C B*C;
```

More examples of using the bar and at operators follow:

$A C(B)$	is equivalent to	$A C(B) A*C(B)$
$A(B) C(B)$	is equivalent to	$A(B) C(B) A*C(B)$
$A(B) B(D E)$	is equivalent to	$A(B) B(D E)$
$A B(A) C$	is equivalent to	$A B(A) C A*C B*C(A)$
$A B(A) C@2$	is equivalent to	$A B(A) C A*C$
$A B C D@2$	is equivalent to	$A B A*B C A*C B*C D A*D B*D C*D$
$A*B(C*D)$	is equivalent to	$A*B(C D)$

Colon, Dash, and Double Dash Operators

You can simplify the specification of a large model when some of your variables have a common prefix by using the colon (:) operator and the dash (-) operator. The dash operator enables you to list variables that are numbered sequentially, and the colon operator selects all variables with a given prefix. For example, if your data set contains the variables X1 through X9, the following MODEL statements are equivalent:

```
model Y = X1 X2 X3 X4 X5 X6 X7 X8 X9;
```

```
model Y = X1-X9;
```

```
model Y = X:;
```

If your data set contains only the three covariates X1, X2, and X9, then the colon operator selects all three variables:

```
model Y = X:;
```

However, the following specification returns an error because X3 through X8 are not in the data set:

```
model Y = X1-X9;
```

The double dash (- -) operator enables you to select variables that are stored sequentially in the SAS data set, whether or not they have a common prefix. You can use the CONTENTS procedure (see [Base SAS Procedures Guide](#)) to determine your variable ordering. For example, if you replace the dash in the preceding MODEL statement with a double dash, as follows, then all three variables are selected:

```
model Y = X1--X9;
```

If your data set contains the variables A, B, and C, then you can use the double dash operator to select these variables by specifying the following:

```
model Y = A--C;
```

GLM Parameterization of Classification Variables and Effects

Table 4.12 shows the types of effects that are available in procedures in this book; they are discussed in more detail in the following subsections. Let A, B, and C represent classification variables, and let X and Z represent continuous variables.

Table 4.12 Available Types of Effects

Effect	Example	Description
Intercept	Default	Intercept (unless the NOINT option is specified)
Regression	X Z	Continuous variables
Polynomial	X*Z	Interaction of continuous variables
Main	A B	CLASS variables
Interaction	A*B	Crossing of CLASS variables
Nested	A(B)	Main effect A nested within CLASS effect B
Continuous-by-class	X*A	Crossing of continuous and CLASS variables
Continuous-nesting-class	X(A)	Continuous variable X1 nested within CLASS variable A
General	X*Z*A(B)	Combinations of different types of effects

Table 4.13 shows some examples of MODEL statements that use various types of effects.

Table 4.13 MODEL Statement Effect Examples

Specification	Type of Model
<code>model Y=X;</code>	Simple regression
<code>model Y=X Z;</code>	Multiple regression
<code>model Y=X X*X;</code>	Polynomial regression
<code>model Y=A;</code>	One-way analysis of variance (ANOVA)
<code>model Y=A B C;</code>	Main-effects ANOVA
<code>model Y=A B A*B;</code>	Factorial ANOVA with interaction
<code>model y=A B (A) C (B A) ;</code>	Nested ANOVA
<code>model Y=A X;</code>	Analysis of covariance (ANCOVA)
<code>model Y=A X (A) ;</code>	Separate-slopes regression
<code>model Y=A X X*A;</code>	Homogeneity-of-slopes regression

Intercept

By default, linear models that are created by procedures in this book automatically include a column of 1s in **X**. This column corresponds to an intercept parameter. In many procedures, you can use the **NOINT** option in the MODEL statement to suppress this intercept. For example, the **NOINT** option is useful when the MODEL statement contains a classification effect and you want the parameter estimates to be in terms of the mean response for each level of that effect.

Regression Effects

Numeric variables or polynomial terms that involve them can be included in the model as regression effects (covariates). The actual values of such terms are included as columns of the relevant model matrices. You can use the **bar operator** along with a regression effect to generate polynomial effects. For example, $X \mid X \mid X$ expands to $X \ X^2 \ X^3$, which is a cubic model.

Main Effects

If a classification variable has m levels, the GLM parameterization generates m columns for its main effect in the model matrix. Each column is an indicator variable for a particular level. The order of the columns is the sort order of the values of their levels and can be controlled by the **ORDER=** option in the **CLASS** statement.

Table 4.14 is an example where β_0 denotes the intercept and A and B are classification variables that have two and three levels, respectively.

Table 4.14 Example of Main Effects

Data		I	A		B		
A	B	β_0	A1	A2	B1	B2	B3
1	1	1	1	0	1	0	0
1	2	1	1	0	0	1	0
1	3	1	1	0	0	0	1
2	1	1	0	1	1	0	0

Table 4.14 *continued*

Data		I	A		B		
2	2	1	0	1	0	1	0
2	3	1	0	1	0	0	1

There are usually more columns for these effects than there are degrees of freedom to estimate them. In other words, the GLM parameterization of main effects is *singular*.

Interaction Effects

Often a model includes interaction (crossed) effects to account for how the effect of a variable changes along with the values of other variables. With an interaction, the terms are first reordered to correspond to the order of the variables in the **CLASS** statement. Thus, B*A becomes A*B if A precedes B in the **CLASS** statement. Then, the GLM parameterization generates columns for all combinations of levels that occur in the data. The order of the columns is such that the rightmost variables in the interaction change faster than the leftmost variables (Table 4.15).

Table 4.15 Example of Interaction Effects

Data		I	A		B			A*B					
A	B	β_0	A1	A2	B1	B2	B3	A1B1	A1B2	A1B3	A2B1	A2B2	A2B3
1	1	1	1	0	1	0	0	1	0	0	0	0	0
1	2	1	1	0	0	1	0	0	1	0	0	0	0
1	3	1	1	0	0	0	1	0	0	1	0	0	0
2	1	1	0	1	1	0	0	0	0	0	1	0	0
2	2	1	0	1	0	1	0	0	0	0	0	1	0
2	3	1	0	1	0	0	1	0	0	0	0	0	1

In the preceding matrix, main-effects columns are not linearly independent of crossed-effects columns. In fact, the column space for the crossed effects contains the space of the main effect.

When your model contains many interaction effects, you might be able to code them more parsimoniously by using the **bar operator** (**|**). The bar operator generates all possible interaction effects. For example, A | B | C expands to A B A*B C A*C B*C A*B*C. To eliminate higher-order interaction effects, use the **at sign** (**@**) in conjunction with the bar operator. For example, A | B | C | D@2 expands to A B A*B C A*C B*C D A*D B*D C*D.

Nested Effects

Nested effects are generated in the same manner as crossed effects. Hence, the design columns that are generated by the following two statements are the same (but the ordering of the columns is different):

```
model Y=A B(A) ;
```

```
model Y=A A*B;
```

The nesting operator in procedures in this book is more of a notational convenience than an operation that is distinct from crossing. Nested effects are typically characterized by the property that the nested variables do not appear as main effects. The order of the variables within nesting parentheses is made to correspond to the

order of these variables in the **CLASS** statement. The order of the columns is such that variables outside the parentheses index faster than those inside the parentheses, and the rightmost nested variables index faster than the leftmost variables (Table 4.16).

Table 4.16 Example of Nested Effects

Data		I	A		B(A)					
A	B	β_0	A1	A2	B1A1	B2A1	B3A1	B1A2	B2A2	B3A2
1	1	1	1	0	1	0	0	0	0	0
1	2	1	1	0	0	1	0	0	0	0
1	3	1	1	0	0	0	1	0	0	0
2	1	1	0	1	0	0	0	1	0	0
2	2	1	0	1	0	0	0	0	1	0
2	3	1	0	1	0	0	0	0	0	1

Continuous-Nesting-Class Effects

When a continuous variable nests or crosses with a classification variable, the design columns are constructed by multiplying the continuous values into the design columns for the classification effect (Table 4.17).

Table 4.17 Example of Continuous-Nesting-Class Effects

Data		I	A		X(A)	
X	A	β_0	A1	A2	X(A1)	X(A2)
21	1	1	1	0	21	0
24	1	1	1	0	24	0
22	1	1	1	0	22	0
28	2	1	0	1	0	28
19	2	1	0	1	0	19
23	2	1	0	1	0	23

This model estimates a separate intercept and a separate slope for X within each level of A.

Continuous-by-Class Effects

Continuous-by-class effects generate the same design columns as continuous-nesting-class effects. Table 4.18 shows the construction of the X*A effect. The two columns for this effect are the same as the columns for the X(A) effect in Table 4.17.

Table 4.18 Example of Continuous-by-Class Effects

Data		I	X	A		X*A	
X	A	β_0	X	A1	A2	X*A1	X*A2
21	1	1	21	1	0	21	0
24	1	1	24	1	0	24	0
22	1	1	22	1	0	22	0

Table 4.18 *continued*

Data		I	X	A		X*A	
28	2	1	28	0	1	0	28
19	2	1	19	0	1	0	19
23	2	1	23	0	1	0	23

You can use continuous-by-class effects together with pure continuous effects to test for homogeneity of slopes.

General Effects

An example that combines all the effects is $X1*X2*A*B*C(D\ E)$. The continuous list comes first, followed by the crossed list, followed by the nested list in parentheses.

Effects might be renamed by the procedure to correspond to ordering rules. For example, $B*A(E\ D)$ might be renamed $A*B(D\ E)$ to satisfy the following:

- Classification variables that occur outside parentheses (crossed effects) are sorted in the order in which they appear in the CLASS statement.
- Variables within parentheses (nested effects) are sorted in the order in which they appear in the CLASS statement.

The sequencing of the parameters that are generated by an effect is determined by the variables whose levels are indexed faster:

- Variables in the crossed list index faster than variables in the nested list.
- Within a crossed or nested list, variables to the right index faster than variables to the left.

For example, suppose that a model includes four effects—A, B, C, and D—each having two levels, 1 and 2, and that the CLASS statement is as follows:

```
class A B C D;
```

Then the order of the parameters for the effect $B*A(C\ D)$, which is renamed $A*B(C\ D)$, is as follows:

$$\begin{aligned}
 &A_1B_1C_1D_1 \rightarrow A_1B_2C_1D_1 \rightarrow A_2B_1C_1D_1 \rightarrow A_2B_2C_1D_1 \rightarrow \\
 &A_1B_1C_1D_2 \rightarrow A_1B_2C_1D_2 \rightarrow A_2B_1C_1D_2 \rightarrow A_2B_2C_1D_2 \rightarrow \\
 &A_1B_1C_2D_1 \rightarrow A_1B_2C_2D_1 \rightarrow A_2B_1C_2D_1 \rightarrow A_2B_2C_2D_1 \rightarrow \\
 &A_1B_1C_2D_2 \rightarrow A_1B_2C_2D_2 \rightarrow A_2B_1C_2D_2 \rightarrow A_2B_2C_2D_2
 \end{aligned}$$

Note that first the crossed effects B and A are sorted in the order in which they appear in the CLASS statement so that A precedes B in the parameter list. Then, for each combination of the nested effects in turn, combinations of A and B appear. The B effect changes fastest because it is rightmost in the cross list. Then A changes next fastest, and D changes next fastest. The C effect changes most slowly because it is leftmost in the nested list.

Nonsingular Parameterization

When a parameterization of main effects provides the same number of columns for the effects as there are degrees of freedom to estimate them, the parameterization is called *nonsingular*. A variety of nonsingular parameterizations for classification effects are available for many procedures in this book. In most of these procedures you use the `PARAM=` option in the `CLASS` statement to specify the parameterization.

Consider a model with one `CLASS` variable `A` that has four levels, 1, 2, 5, and 7. Details of the possible choices for the `PARAM=` option follow.

REFERENCE

Three columns are created to indicate group membership of the nonreference levels. For the reference level, all three dummy variables have a value of 0. For example, if the reference level is 7 (`REF=7`), the design matrix columns for `A` are as follows.

Reference Coding			
A	Design Matrix		
	A1	A2	A5
1	1	0	0
2	0	1	0
5	0	0	1
7	0	0	0

Parameter estimates of `CLASS` main effects that use the reference coding scheme estimate the difference in the effect of each nonreference level compared to the effect of the reference level.

EFFECT

Three columns are created to indicate group membership of the nonreference levels. For the reference level, all three dummy variables have a value of -1 . For example, if the reference level is 7 (`REF=7`), the design matrix columns for `A` are as follows.

Effect Coding			
A	Design Matrix		
	A1	A2	A5
1	1	0	0
2	0	1	0
5	0	0	1
7	-1	-1	-1

Parameter estimates of `CLASS` main effects that use the effect coding scheme estimate the difference in the effect of each nonreference level compared to the average effect over all four levels.

ORDINAL | THERMOMETER

Three columns are created to indicate group membership of the higher levels of the effect. For the first level of the effect (which for `A` is 1), all three dummy variables have a value of 0. The design matrix columns for `A` are as follows.

Ordinal Coding			
A	Design Matrix		
	A2	A5	A7
1	0	0	0
2	1	0	0
5	1	1	0
7	1	1	1

The first level of the effect is a control or baseline level. Parameter estimates of CLASS main effects, using the ORDINAL coding scheme, estimate the differences between effects of successive levels. When the parameters have the same sign, the effect is monotonic across the levels.

POLYNOMIAL | POLY Three columns are created. The first represents the linear term (x), the second represents the quadratic term (x^2), and the third represents the cubic term (x^3), where x is the level value. If the CLASS levels are not numeric, they are translated into 1, 2, 3, ... according to their sort order. The design matrix columns for A are as follows.

Polynomial Coding			
A	Design Matrix		
	APOLY1	APOLY2	APOLY3
1	1	1	1
2	2	4	8
5	5	25	125
7	7	49	343

ORTHEFFECT The columns are obtained by applying the Gram-Schmidt orthogonalization to the columns for PARAM=EFFECT. The design matrix columns for A are as follows.

Orthogonal Effect Coding			
A	Design Matrix		
	AOEFF1	AOEFF2	AOEFF3
1	1.41421	-0.81650	-0.57735
2	0	1.63299	-0.57735
5	0	0	1.73205
7	-1.41421	-0.81649	-0.57735

ORTHORDINAL | ORTHOTHERM The columns are obtained by applying the Gram-Schmidt orthogonalization to the columns for PARAM=ORDINAL. The design matrix columns for A are as follows.

Orthogonal Ordinal Coding			
Design Matrix			
A	AOORD1	AOORD2	AOORD3
1	-1.73205	0	0
2	0.57735	-1.63299	0
5	0.57735	0.81650	-1.41421
7	0.57735	0.81650	1.41421

ORTHPOLY

The columns are obtained by applying the Gram-Schmidt orthogonalization to the columns for PARAM=POLY. The design matrix columns for A are as follows.

Orthogonal Polynomial Coding			
Design Matrix			
A	AOPOLY1	AOPOLY2	AOPOLY5
1	-1.15311	0.90712	-0.92058
2	-0.73380	-0.54041	1.47292
5	0.52414	-1.37034	-0.92058
7	1.36277	1.00363	0.36823

ORTHREF

The columns are obtained by applying the Gram-Schmidt orthogonalization to the columns for PARAM=REFERENCE. The design matrix columns for A are as follows.

Orthogonal Reference Coding			
Design Matrix			
A	AOREF1	AOREF2	AOREF3
1	1.73205	0	0
2	-0.57735	1.63299	0
5	-0.57735	-0.81650	1.41421
7	-0.57735	-0.81650	-1.41421

Class Variable Parameterization with Unbalanced Designs

Procedures in this book initially parameterize the CLASS variables by looking at the levels of the variables across the complete data set. If you have an *unbalanced* replication of levels across variables or BY groups, then the design matrix and the parameter interpretation might be different from what you expect. For example, suppose you have a model that has one CLASS variable A with three levels (1, 2, and 3), and another CLASS variable B with two levels (1 and 2). If the third level of A occurs only with the first level of B, if you use the EFFECT parameterization, and if your model contains the effect A(B) and an intercept, then the design for A within the second level of B is not a differential effect. In particular, the design looks like the following:

		Design Matrix			
B	A	A(B=1)		A(B=2)	
		A1	A2	A1	A2
1	1	1	0	0	0
1	2	0	1	0	0
1	3	-1	-1	0	0
2	1	0	0	1	0
2	2	0	0	0	1

Procedures in this book detect linear dependency among the last two design variables and set the parameter for A2(B=2) to 0, resulting in an interpretation of these parameters as if they were reference- or dummy-coded. The REFERENCE or GLM parameterization might be more appropriate for such problems.

Model Selection Methods

The model selection methods are specified in the **METHOD=** option in the **SELECTION** statement. The following methods are available, although specific procedures might support only a subset of these methods. Furthermore, the examples in this section refer to fit criteria that might not be supported by a specific procedure.

Forward Selection

This section applies to the following procedures: CNTSELECT and SEVSELECT.

METHOD=FORWARD specifies the forward selection technique, which begins with just the intercept and then sequentially adds the effect that most improves the fit. The process terminates when no significant improvement can be obtained by adding any effect.

In the traditional implementation of forward selection, the statistic that is used to determine whether to add an effect is the significance level of a hypothesis test that reflects an effect's contribution to the model if it is included. At each step, the effect that is most significant is added. The process stops when the significance level for adding any effect is greater than some specified entry significance level.

An alternative approach to address the critical problem of when to stop the selection process that is implemented by the procedures in this book is to assess the quality of the models that are produced by the forward selection method and choose the model from this sequence that “best” balances goodness of fit against model complexity. You can use several criteria for this purpose. These criteria fall into two groups—information criteria and criteria that are based on out-of-sample prediction performance.

It is important to keep in mind that forward selection bases the decision about what effect to add at any step by considering models that differ by one effect from the current model. This search paradigm cannot guarantee reaching a “best” subset model. Furthermore, the add decision is greedy in the sense that the effect that is deemed most significant is the effect that is added. The optimality of this one step choice depends on the optimality of the selection criterion for the model that contains that effect. You can specify the criterion to optimize at each step by using the **SELECT=** option. For example, the following statement requests that at each step the effect that is added be the one that produces a model that has the smallest value of the SBC:


```
selection method=forward(select=SBC);
```

When all effects are variables (that is, effects have one degree of freedom and no hierarchy), using AIC, AICC, or SBC as the selection criterion for forward selection produces the same sequence of additions. However, if the degrees of freedom contributed by different effects are not constant or if an out-of-sample prediction-based criterion is used, then different sequences of additions might be obtained.

If you specify only the **SELECT=** criterion, then this criterion is also used to decide when to stop the selection process. In the previous example, not only do effects enter based on the SBC value, but the selection terminates when SBC has a local minimum.

You use the **CHOOSE=** option to specify the criterion for selecting one model from the sequence of models produced. If you do not specify a **CHOOSE=** criterion, then the model at the final step is the selected model.

For example, if you specify the following statement, then forward selection terminates at the step where the SBC reaches a (local) minimum:

```
selection method=forward(select=SBC choose=AIC);
```

However, the selected model is the first one that has the minimum value of Akaike's information criterion. In some cases, this minimum value might occur at a step much earlier than the final step. In other cases, the AIC might start increasing only if more steps are performed—that is, a larger value is used for the significance level for entry. If you want to minimize AIC, then too many steps are performed in the former case and too few in the latter case. To address this issue, you can use the **STOP=** option to specify a stopping criterion. When you specify a stopping criterion, forward selection continues until a local extremum of the stopping criterion in the sequence of models generated is reached. To be deemed a local extremum, a criterion value at a particular step must be better than its value at the next n steps, where n is known as the “stop horizon.” By default, the stop horizon is three steps, but you can change this by specifying the **STOPHORIZON=** option.

For example, if you specify the following statement, then forward selection terminates at the step where the effect to be added at the next step would produce a model that has an AIC statistic larger than the AIC statistic of the current model:

```
selection method=forward(select=SBC stop=AIC) stophorizon=1;
```

In most cases, provided that the stop horizon is long enough that the local extremum of the named criterion occurs before the final step, specifying either of the following statements selects the same model, but more steps are done in the first case:

```
selection method=forward(select=SL choose=CRITERION);
selection method=forward(select=SL stop=CRITERION);
```

In some cases, there might be a better local extremum that cannot be reached if you specify the **STOP=** option but can be found if you use the **CHOOSE=** option. Also, you can use the **CHOOSE=** option in preference to the **STOP=** option if you want to examine how the named criterion behaves as you move beyond the step where the first local minimum of this criterion occurs.

You can specify both the **CHOOSE=** and **STOP=** options. You can also use these options together with options that specify size-based limits on the selected model. You might want to consider models that are generated by forward selection and have at most some fixed number of effects, but select from within this set based on a criterion that you specify. For example, specifying the following statements requests that forward selection continue until there are 20 effects in the final model and chooses among the sequence of models the one that has the smallest value of the corrected Akaike's information criterion:

```
selection method=forward(stop=none maxeffects=20 choose=AICC);
```

You can also combine these options to select a model where one of two conditions is met. For example, the following statement chooses whatever occurs first between a local minimum of the Schwarz Bayesian information criterion (SBC) and a local minimum of the corrected Akaike's information criterion (AICC):

```
selection method=forward(stop=AICC choose=SBC);
```

You can find discussion and references to studies about criteria for variable selection in Burnham and Anderson (2002), along with some cautions and recommendations.

Examples of Forward Selection Specifications

The following statement adds effects that at each step produce the lowest value of the SBC statistic and stops at the step where adding any effect would increase the SBC statistic:

```
selection method=forward stophorizon=1;
```

The following statement adds effects that at each step produce the lowest value of the AIC statistic and stops at the first step whose AIC value is smaller than the AIC value at the next three steps:

```
selection=forward(select=AIC);
```

Backward Elimination

This section applies to the following procedures: CNTSELECT and SEVSELECT.

METHOD=BACKWARD specifies the backward elimination technique. This technique starts from the full model, which includes all independent effects. Then effects are deleted one by one until a stopping condition is satisfied. At each step, the effect that shows the smallest contribution to the model is deleted.

Just as with forward selection, you can use the **SELECT=** option to change the criterion that is used to assess effect contributions. You can also specify a stopping criterion in the **STOP=** option and use a **CHOOSE=** option to provide a criterion for selecting among the sequence of models produced. For more information, see the discussion in the section “Forward Selection” on page 102.

Examples of Backward Selection Specifications

The following statement removes effects that at each step produce the largest value of the Schwarz Bayesian information criterion (SBC) statistic and stops at the step where removing any effect increases the SBC statistic:

```
selection method=backward stophorizon=1;
```

The following statement bases removal of effects on the corrected Akaike's information criterion (AICC) and stops at the first step whose AICC value is smaller than the AICC value at the next three steps:

```
selection method=backward(select=AICC);
```

Stepwise Selection

This section applies to the following procedures: CNTSELECT and SEVSELECT.

METHOD=STEPWISE specifies the stepwise method, which modifies the forward selection technique by allowing effects already in the model to be removed.

Just as you can in forward selection and backward elimination, you can use the **SELECT=** option to change the criterion that is used to assess effect contributions. You can also use the **STOP=** option to specify a stopping criterion and use a **CHOOSE=** option to provide a criterion for selecting among the sequence of models produced. For more information, see the section “[Forward Selection](#)” on page 102.

The procedures in this book support further modification to the stepwise method. In the standard stepwise method, no effect can enter the model if removing any effect currently in the model would yield an improved value of the selection criterion. In the modification, you can use the **COMPETITIVE** option to specify that addition and deletion of effects should be treated competitively. The selection criterion is evaluated for all models that are produced by deleting an effect from the current model or by adding an effect to this model. The action that most improves the selection criterion is the action taken.

Examples of Stepwise Selection Specifications

The following statement requests stepwise selection based on the SBC criterion:

```
selection method=stepwise;
```

First, if removing any effect yields a model that has a lower SBC statistic than the current model, then the effect that produces the smallest SBC statistic is removed. If removing any effect increases the SBC statistic, then provided that adding some effect lowers the SBC statistic, the effect that produces the model that has the lowest SBC is added.

The following statement requests the stepwise method where effects enter and leave based on the AIC statistic, but with the following extra check: if any effect to be added or removed yields a model whose SBC statistic is greater than the SBC statistic of the current model, then the stepwise method terminates at the current model.

```
selection method=stepwise(select=AIC stop=SBC) stophorizon=1;
```

The following statement requests stepwise selection that is based on the AICC criterion and treats additions and deletions competitively:

```
selection method=stepwise(select=AICC competitive);
```

Each step evaluates the AICC statistics that correspond to the removal of any effect in the current model or the addition of any effect to the current model and chooses the addition or removal that produced the minimum value, provided that this minimum is lower than the AICC statistic of the current model.

Forward-Swap Selection

This section applies to the following procedures: CNTSELECT and SEVSELECT.

METHOD=FORWARDSSWAP specifies the forward-swap selection method, which is an extension of the forward selection method. The forward-swap selection method incorporates steps that improve a model by replacing an effect in the model with an effect that is not in the model.

The forward-swap selection method begins by finding the one-effect model that produces the best value of the selection criterion. Then another effect (the one that yields the greatest improvement in the selection criterion) is added. After the two-effect model is obtained, each of the effects in the model is compared to each effect that is not in the model. For each comparison, the forward-swap method determines whether removing one effect and replacing it with the other effect improves the selection criterion. After comparing all possible swaps, the forward-swap method makes the swap that produces the greatest improvement in the selection criterion. Comparisons begin again, and the process continues until the forward-swap method finds that no other swap could improve the selection criterion. Thus, the two-variable model that is produced is considered the “best” two-variable model that the technique can find. Another variable is then added to the model, and the comparing-and-swapping process is repeated to find the “best” three-variable model, and so on.

The difference between the stepwise selection method and the forward-swap selection method is that all swaps are evaluated before any addition is made in the forward-swap method. In the stepwise selection method, the “worst” effect might be removed without considering what adding the “best” remaining effects might accomplish. Because the forward-swap method needs to examine all possible pairwise effect swaps at each step of the selection process, the forward-swap method is much more computationally expensive than the stepwise selection method; it might not be appropriate for models that contain a large number of effects.

Informative Missingness

This section applies to the following procedures: CNTSELECT, CPANEL, CQLIM, CSPATIALREG, and SEVSELECT.

The concept of informative missingness is one way to account for missing values in statistical analyses and, in particular, statistical modeling. Missing values can be a problem because they reduce the amount of available data. When you work with classification variables (factors, which are leveled variables), you can treat a missing value as an actual level of the variable and allow it to participate in the analysis.

However, when continuous variables have missing values, the observation is removed from the analysis. In data that have many missing values, removing observations can reduce the amount of available data greatly, and the sets of observations used in one model versus another model can vary based on which variables are included in the model.

Of course, there are many reasons for missing values, and substituting values for missing values has to be done with caution. For example, the famous Framingham Heart study data set contains 5,209 observations on subjects in a longitudinal study that helped understand the relationship between smoking, cholesterol,

and coronary heart disease. One of the variables in the data set is AgeCHDdiag. This variable represents the age at which a patient was diagnosed with coronary heart disease (CHD). If you include this variable in a statistical model, only 1,449 observations are available, because the value cannot be observed unless a patient has experienced CHD. Including this variable acts as a filter that reduces the analysis set to the subjects who have CHD. You cannot impute the value for subjects where the variable has a missing value, because you cannot impute an age at which someone who has not had CHD would have contracted coronary heart disease.

With informative missingness, you are not so much substituting imputed values for the missing values as you are modeling the missingness. Consider a simple linear regression model:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Suppose that some of the values for the regressor variable x are missing. The fitted model uses only observations for which y and x have been observed.

In order to predict the outcome y for an observation that has a missing x , either you assume that y is missing or you substitute a value (such as the average value, $\bar{x}$) for the missing x . Because the estimate for the intercept is in the simple linear regression model, the predicted value would be the average response of the nonmissing values, $\bar{y}$.

With informative missingness, you extend the model by adding extra effects for each effect that contains at least one continuous variable. In the simple linear regression model, you add one column to the model and slightly change the content of the x variable:

$$y = \beta_0 + \beta_1 x^* + \beta_2 x_{\text{miss}} + \epsilon_1$$

The variable x^* contains the original values of x if they are not missing, and the average of x otherwise:

$$x^* = \begin{cases} x & \text{if } x \text{ is not missing} \\ \bar{x} & \text{otherwise} \end{cases}$$

The variable x_{miss} is a dummy variable whose value is 1 when x is missing, and 0 otherwise:

$$x_{\text{miss}} = \begin{cases} 1 & \text{if } x \text{ is missing} \\ 0 & \text{otherwise} \end{cases}$$

The fitted model is not the same model that results from substituting $\bar{x}$ for the missing values during training, because the model that simply substitutes $\bar{x}$ for the missing values is

$$y = \beta_0 + \beta_1 x^* + \epsilon_2$$

The informative missing model has an extra parameter, and unless all values of x_{miss} are 0 (in which case there are no missing values), the informative missing model has a higher R-square value, because it picks up more variation.

The parameter estimate for β_2 measures the amount by which the predicted value differs from a predicted value at $\bar{x}$.

Multithreading

Threading refers to the organization of computational work into multiple tasks (processing units that can be scheduled by the operating system). A task is associated with a thread. Multithreading refers to the concurrent execution of threads. When multithreading is possible, substantial performance gains can be realized compared to sequential (single-threaded) execution. The number of threads spawned by a procedure in this book is determined by your installation.

The tasks that are multithreaded by procedures in this book are primarily defined by dividing the data that are processed on a single machine among the threads—that is, the procedures implement multithreading through a data-parallel model. For example, if the input data table has 1,000 observations and the procedure is running on four threads, then 250 observations are associated with each thread. All operations that require access to the data are then multithreaded. These operations include the following (not all operations are required for all procedures):

- variable levelization
- effect levelization
- formation of the initial crossproducts matrix
- formation of approximate Hessian matrices for candidate evaluation during model selection
- objective function calculation
- gradient calculation
- Hessian calculation
- scoring of observations

In addition, operations on matrices such as sweeps can be multithreaded provided that the matrices are of sufficient size to realize performance benefits from managing multiple threads for the particular matrix operation.

Choosing an Optimization Algorithm

This section applies to the SEVSELECT procedure.

First- or Second-Order Algorithms

The factors that go into choosing a particular optimization technique for a particular problem are complex. Trial and error can be involved.

For many optimization problems, computing the gradient takes more computer time than computing the function value. Computing the Hessian sometimes takes *much* more computer time and memory than computing the gradient, especially when there are many decision variables. Unfortunately, optimization techniques that do not use some kind of Hessian approximation usually require many more iterations than techniques that do use a Hessian matrix, and as a result the total run time of these techniques is often longer. Techniques that do not use the Hessian also tend to be less reliable. For example, they can terminate more easily at stationary points than at global optima.

Table 4.19 shows which derivatives are required for each optimization technique.

Table 4.19 Derivatives Required

Algorithm	First-Order	Second-Order
TRUREG	x	x
NEWRAP	x	x
NRRIDG	x	x
QUANEW	x	-
DBLDOG	x	-
CONGRA	x	-
NMSIMP	-	-

The second-derivative methods TRUREG, NEWRAP, and NRRIDG are best for small problems for which the Hessian matrix is not expensive to compute. Sometimes the NRRIDG algorithm can be faster than the TRUREG algorithm, but TRUREG can be more stable. The NRRIDG algorithm requires only one matrix with $p(p+1)/2$ double words; TRUREG and NEWRAP require two such matrices. Here, p denotes the number of parameters in the optimization.

The first-derivative methods QUANEW and DBLDOG are best for medium-sized problems for which the objective function and the gradient can be evaluated much faster than the Hessian. In general, the QUANEW and DBLDOG algorithms require more iterations than TRUREG, NRRIDG, and NEWRAP, but each iteration can be much faster. The QUANEW and DBLDOG algorithms require only the gradient to update an approximate Hessian, and they require slightly less memory than TRUREG or NEWRAP.

The first-derivative method CONGRA is best for large problems for which the objective function and the gradient can be computed much faster than the Hessian and for which too much memory is required to store the (approximate) Hessian. In general, the CONGRA algorithm requires more iterations than QUANEW or DBLDOG, but each iteration can be much faster. Because CONGRA requires only a factor of p double-word memory, many large applications can be solved only by CONGRA.

The no-derivative method NMSIMP is best for small problems for which derivatives are not continuous or are very difficult to compute.

Each optimization method uses one or more convergence criteria that determine when it has converged. An algorithm is considered to have converged when any one of the convergence criteria is satisfied. For example, under the default settings, the QUANEW algorithm converges if $\text{ABSGCONV} < 1\text{E-}5$, $\text{FCONV} < 2 \times \epsilon$, or $\text{GCONV} < 1\text{E-}8$.

By default, procedures in this book apply the NRRIDG algorithm because it can take advantage of multi-threading in Hessian computations and inversions. If the number of parameters becomes large, specifying the QUANEW technique, which is a first-order method with good overall properties, is recommended.

Algorithm Descriptions

The following subsections provide details about each optimization technique and follow the same order as Table 4.19.

Trust Region Optimization (TRUREG)

The trust region method uses the gradient $\mathbf{g}(\boldsymbol{\beta}^{(k)})$ and the Hessian matrix $\mathbf{H}(\boldsymbol{\beta}^{(k)})$; thus, it requires that the objective function $f(\boldsymbol{\beta})$ have continuous first- and second-order derivatives inside the feasible region.

The trust region method iteratively optimizes a quadratic approximation to the nonlinear objective function within a hyperelliptic trust region with radius Δ that constrains the step size that corresponds to the quality of the quadratic approximation. The trust region method is implemented based on Dennis, Gay, and Welsch (1981), Gay (1983), and Moré and Sorensen (1983).

The trust region method performs well for small- to medium-sized problems, and it does not need many function, gradient, and Hessian calls. However, if the computation of the Hessian matrix is computationally expensive, one of the dual quasi-Newton or conjugate gradient algorithms might be more efficient.

Newton-Raphson Optimization with Line Search (NEWRAP)

The NEWRAP technique uses the gradient $\mathbf{g}(\boldsymbol{\beta}^{(k)})$ and the Hessian matrix $\mathbf{H}(\boldsymbol{\beta}^{(k)})$; thus, it requires that the objective function have continuous first- and second-order derivatives inside the feasible region. If second-order derivatives are computed efficiently and precisely, the NEWRAP method can perform well for medium-sized to large problems, and it does not need many function, gradient, and Hessian calls.

This algorithm uses a pure Newton step when the Hessian is positive-definite and when the Newton step reduces the value of the objective function successfully. Otherwise, a combination of ridging and line search is performed to compute successful steps. If the Hessian is not positive-definite, a multiple of the identity matrix is added to the Hessian matrix to make it positive-definite (Eskow and Schnabel 1991).

In each iteration, a line search is performed along the search direction to find an approximate optimum of the objective function. The line-search method uses quadratic interpolation and cubic extrapolation.

Newton-Raphson Ridge Optimization (NRRIDG)

The NRRIDG technique uses the gradient $\mathbf{g}(\boldsymbol{\beta}^{(k)})$ and the Hessian matrix $\mathbf{H}(\boldsymbol{\beta}^{(k)})$; thus, it requires that the objective function have continuous first- and second-order derivatives inside the feasible region.

This algorithm uses a pure Newton step when the Hessian is positive-definite and when the Newton step reduces the value of the objective function successfully. If at least one of these two conditions is not satisfied, a multiple of the identity matrix is added to the Hessian matrix.

Because the NRRIDG technique uses an orthogonal decomposition of the approximate Hessian, each iteration of NRRIDG can be slower than that of the NEWRAP technique, which works with a Cholesky decomposition. However, NRRIDG usually requires fewer iterations than NEWRAP.

The NRRIDG method performs well for small- to medium-sized problems, and it does not require many function, gradient, and Hessian calls. However, if the computation of the Hessian matrix is computationally expensive, one of the dual quasi-Newton or conjugate gradient algorithms might be more efficient.

Quasi-Newton Optimization (QUANEW or DUQUANEW)

The dual quasi-Newton method uses the gradient $\mathbf{g}(\boldsymbol{\beta}^{(k)})$, and it does not need to compute second-order derivatives because they are approximated. It works well for medium-sized to moderately large optimization problems, where the objective function and the gradient can be computed much faster than the Hessian. However, in general the QUANEW technique requires more iterations than the TRUREG, NEWRAP, and NRRIDG techniques, which compute second-order derivatives. The QUANEW technique provides an appropriate balance between the speed and stability required for most nonlinear mixed model applications.

The QUANEW technique implemented by procedures in this book is the dual quasi-Newton algorithm, which updates the Cholesky factor of an approximate Hessian.

In each iteration, a line search is performed along the search direction to find an approximate optimum. The line-search method uses quadratic interpolation and cubic extrapolation to obtain a step size α that satisfies

the Goldstein conditions (Fletcher 1987). One of the Goldstein conditions can be violated if the feasible region defines an upper limit of the step size. Violating the left-side Goldstein condition can affect the positive-definiteness of the quasi-Newton update. In that case, either the update is skipped or the iterations are restarted with an identity matrix, resulting in the steepest descent or ascent search direction.

Double-Dogleg Optimization (DBLDOG)

The double-dogleg optimization method combines the ideas of the quasi-Newton and trust region methods. In each iteration, the double-dogleg algorithm computes the step $s^{(k)}$ as the linear combination of the steepest descent or ascent search direction $s_1^{(k)}$ and a quasi-Newton search direction $s_2^{(k)}$:

$$s^{(k)} = \alpha_1 s_1^{(k)} + \alpha_2 s_2^{(k)}$$

The step is requested to remain within a prespecified trust region radius (Fletcher 1987, p. 107). Thus, the DBLDOG subroutine uses the dual quasi-Newton update but does not perform a line search.

The double-dogleg optimization technique works well for medium-sized to moderately large optimization problems, where the objective function and the gradient can be computed much faster than the Hessian. The implementation is based on Dennis and Mei (1979) and Gay (1983), but it is extended for dealing with boundary and linear constraints. The DBLDOG technique generally requires more iterations than the TRUREG, NEWRAP, and NRRIDG techniques, which require second-order derivatives; however, each of the DBLDOG iterations is computationally cheap. Furthermore, the DBLDOG technique requires only gradient calls for the update of the Cholesky factor of an approximate Hessian.

Conjugate Gradient Optimization (CONGRA)

Second-order derivatives are not required by the CONGRA algorithm and are not even approximated. The CONGRA algorithm can be expensive in function and gradient calls, but it requires only $O(p)$ memory for unconstrained optimization. In general, many iterations are required to obtain a precise solution, but each of the CONGRA iterations is computationally cheap.

The CONGRA subroutine should be used for optimization problems with large p . For the unconstrained or boundary-constrained case, CONGRA requires only $O(p)$ bytes of working memory, whereas all other optimization methods require order $O(p^2)$ bytes of working memory. During p successive iterations, uninterrupted by restarts or changes in the working set, the conjugate gradient algorithm computes a cycle of p conjugate search directions. In each iteration, a line search is performed along the search direction to find an approximate optimum of the objective function. The line-search method uses quadratic interpolation and cubic extrapolation to obtain a step size α that satisfies the Goldstein conditions. One of the Goldstein conditions can be violated if the feasible region defines an upper limit for the step size.

Nelder-Mead Simplex Optimization (NMSIMP)

The Nelder-Mead simplex method does not use any derivatives and does not assume that the objective function has continuous derivatives. The objective function itself needs to be continuous. This technique is quite expensive in the number of function calls, and it might be unable to generate precise results for $p \gg 40$.

The original Nelder-Mead simplex algorithm is implemented and extended to boundary constraints. This algorithm does not compute the objective for infeasible points, but it changes the shape of the simplex adapting to the nonlinearities of the objective function. This change contributes to an increased speed of convergence and uses a special termination criterion.

References

- Browne, M. W. (1982). “Covariance Structures.” In *Topics in Applied Multivariate Analysis*, edited by D. M. Hawkins, 72–141. Cambridge: Cambridge University Press.
- Burnham, K. P., and Anderson, D. R. (2002). *Model Selection and Multimodel Inference*. 2nd ed. New York: Springer-Verlag.
- Dennis, J. E., Gay, D. M., and Welsch, R. E. (1981). “An Adaptive Nonlinear Least-Squares Algorithm.” *ACM Transactions on Mathematical Software* 7:348–368.
- Dennis, J. E., and Mei, H. H. W. (1979). “Two New Unconstrained Optimization Algorithms Which Use Function and Gradient Values.” *Journal of Optimization Theory and Applications* 28:453–482.
- Eskow, E., and Schnabel, R. B. (1991). “Algorithm 695: Software for a New Modified Cholesky Factorization.” *ACM Transactions on Mathematical Software* 17:306–312.
- Fletcher, R. (1987). *Practical Methods of Optimization*. 2nd ed. Chichester, UK: John Wiley & Sons.
- Gay, D. M. (1983). “Subroutines for Unconstrained Minimization.” *ACM Transactions on Mathematical Software* 9:503–524.
- Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer-Verlag.
- Moré, J. J., and Sorensen, D. C. (1983). “Computing a Trust-Region Step.” *SIAM Journal on Scientific and Statistical Computing* 4:553–572.
- Searle, S. R. (1971). *Linear Models*. New York: John Wiley & Sons.

Chapter 5

CARIMA Procedure

Contents

Overview: CARIMA Procedure	114
Using CAS Sessions and CAS Engine Librefs	114
Getting Started: CARIMA Procedure	115
Syntax: CARIMA Procedure	116
Functional Summary	116
PROC CARIMA Statement	118
BY Statement	119
ESTIMATE Statement	119
FORECAST Statement	121
ID Statement	121
IDENTIFY Statement	125
Details: CARIMA Procedure	125
Accumulation	125
Missing Value Interpretation	127
Parameter Estimation	127
Transformations	127
Forecasting	127
Data Table Output	128
Printed Output	130
ODS Table Names	130
Example: CARIMA Procedure	131
Example 5.1: Seasonal Model for the Airline Series	131
References	134

Overview: CARIMA Procedure

The CARIMA procedure analyzes and forecasts equally spaced univariate time series data, by using the autoregressive integrated moving-average (ARIMA) or autoregressive moving-average (ARMA) model.

An ARIMA model predicts a value in a response time series as a linear combination of its own past values and past errors (also called shocks or innovations).

The ARIMA approach was first popularized by Box and Jenkins (1976), and ARIMA models are often referred to as Box-Jenkins models.

The CARIMA procedure provides a comprehensive set of tools for parameter estimation and for forecasting univariate time series. It also offers a high degree of flexibility in the types of ARIMA models that you can analyze; it supports seasonal, subset, and factored ARIMA models. For each fitted model, PROC CARIMA prints parameter estimates along with standard errors and *p*-values, and it prints multistep-ahead variable forecasts along with confidence intervals.

Before you use PROC CARIMA, you should exercise care and judgment and be familiar with Box-Jenkins methods.

The ARIMA class of time series models is complex and powerful, and some degree of expertise is needed to use them correctly.

PROC CARIMA requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: CARIMA Procedure

This section outlines the use of the CARIMA procedure and gives a cursory description of the ARIMA modeling process for readers who are less familiar with these methods.

Given an input data table that contains numerous time series variables recorded at a specific frequency, the CARIMA procedure can fit an ARIMA model to the series and produce forecasts.

For example, suppose that the input data table `mylib.Sales` contains sales data that were recorded monthly, the variable that represents time is `Date`, and the forecasts are to be recorded in the output data table `mylib.NextYear`. If you believe that sales in the current month are affected only by sales in the previous month, then you can use the CARIMA procedure to fit an AR(1) model and forecasts sales for the next four months as follows:

```
data mylib.sales;
  format date date9.;
  input date : date9. shoes socks laces dresses
          coats shirts ties belts hats blouses;
  datalines;
01JAN1994 3557 3718 6368.80 575 987 10.8200 15.0000 102.600 12410 15013
01FEB1994 5128 4174 8123.20 565 1000 12.1200 15.1000 99.900 13556 12413
01MAR1994 5222 4482 7807.20 406 1005 11.7800 15.3000 102.000 11063 12752

... more lines ...

01DEC1998 5399 4795 6075.30 614 1239 21.2000 1.9000 79.700 23004 22044
01JAN1999 6405 4981 6812.10 607 1196 20.7000 14.9000 99.900 20583 26093
;

proc carima data=mylib.sales outfor=mylib.nextyear;
  id date interval=month;
  identify _numeric_;
  estimate p =1;
  forecast lead=4;
```

```
run;
```

These statements assume that the SAS library is named mylib, but you can substitute any appropriately named SAS library. These statements generate forecasts for every numeric variable in the input data table mylib.Sales for the next four months and store these forecasts in the output data table mylib.NextYear. Other output data tables can be specified to store the parameter estimates and summary data.

The CARIMA procedure can forecast time series data, whose observations are equally spaced by a specific time interval (for example, monthly or weekly), or transactional data, whose observations are not spaced with respect to any particular time interval.

Given an input data table that contains transactional variables that are not recorded at any specific frequency, the CARIMA procedure accumulates the data to a specific time interval and forecasts the accumulated series. For an example, see “[Example 5.1: Seasonal Model for the Airline Series](#)” on page 131.

Syntax: CARIMA Procedure

The following statements are available in the CARIMA procedure:

```
PROC CARIMA options ;
BY variables ;
ID variable INTERVAL= interval < options > ;
IDENTIFY variable-list ;
ESTIMATE < options > ;
FORECAST < options > ;
```

The PROC CARIMA and IDENTIFY statements are required. The CARIMA procedure supports multiple IDENTIFY, ESTIMATE, and FORECAST statements. The FORECAST statement forecasts all variables in the *variable-list* that is specified in the preceding IDENTIFY statement according to the model that is specified in the preceding ESTIMATE statement. If no ESTIMATE statement is specified before a FORECAST statement, then prediction is carried out according to a white noise model.

Functional Summary

The statements and options that control the CARIMA procedure are summarized in [Table 5.1](#).

Table 5.1 Functional Summary

Description	Statement	Option
Statements		
Specify data tables and options	PROC CARIMA	
Request BY-group processing	BY	
Specify the variables to model	IDENTIFY	
Specify the variables to estimate	ESTIMATE	
Specify the variables to forecast	FORECAST	

Table 5.1 *continued*

Description	Statement	Option
Data Table Options		
Specify the input data table	PROC CARIMA	DATA =
Specify the output data table	PROC CARIMA	OUT =
Specify parameter output data table	PROC CARIMA	OUTEST =
Specify forecast output data table	PROC CARIMA	OUTFOR =
Specify summary output data table	PROC CARIMA	OUTSUM=
Accumulation Options		
Specify the accumulation frequency	ID	INTERVAL =
Specify the interval alignment	ID	ALIGN =
Specify the starting time ID value	ID	START =
Specify the ending time ID value	ID	END =
Specify the accumulation statistic	ID, FORECAST	ACCUMULATE =
Specify how to interpret missing values	ID, FORECAST	SETMISSING =
Estimation Options		
Specify the convergence criterion	ESTIMATE	CONVERGE =
Specify the perturbation value for computing numerical derivatives	ESTIMATE	DELTA =
Specify the degrees of differencing	ESTIMATE	DIFF =
Specify the maximum number of iterations	ESTIMATE	MAXITER =
Specify the estimation method	ESTIMATE	METHOD =
Specify a constant term	ESTIMATE	MU =
Specify whether to include an intercept in the model	ESTIMATE	NOINT =
Specify whether to restrict candidate estimates for the AR and MA parameter estimates to stationarity and invertibility regions	ESTIMATE	NOSTABLE =
Specify the AR order	ESTIMATE	P =
Specify the MA order	ESTIMATE	Q =
Specify the singularity criterion	ESTIMATE	SINGULAR =
Specify the transformation type	ESTIMATE	TRANSFORM =
Forecasting Horizon		
Specify the forecast horizon or lead	FORECAST	LEAD =
Specify the confidence level	FORECAST	ALPHA =
Miscellaneous Options		
Specify the time ID format	ID	FORMAT =
Specify the method for trimming the data in the BY groups	ID	TRIMID =

The following sections describe the PROC CARIMA statement and then describe the other statements in alphabetical order.

PROC CARIMA Statement

PROC CARIMA *options* ;

You can specify the following *options*:

DATA=*libref.data-table*

names the input data table for PROC CARIMA to use. The default is the most recently created data table. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 114.

data-table specifies the name of the input data table.

OUT=*libref.data-table*

names the output data table to contain the forecasts of the variables specified in the subsequent FORECAST statements. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 114.

If an ID statement is specified, its variable is also included in the OUT= data table and its values are accumulated on the basis of the ACCUMULATE= option in the ID statement.

OUTEST=*libref.data-table*

names the output data table to contain the model parameter estimates and the associated test statistics and probability values. The OUTEST= data table is useful for evaluating the significance of the model parameters and understanding the model dynamics.

OUTFOR=*libref.data-table*

names the output data table to contain the forecast time series components (actual, predicted, lower confidence limit, upper confidence limit, prediction error, and prediction standard error). The OUTFOR= data table is useful for displaying the forecasts in tabular or graphical form.

OUTSUM=*libref.data-table*

names the output data table to contain the summary statistics and the forecast summation. The summary statistics are based on the accumulated time series when the ACCUMULATE= or SETMISSING= option is specified in the ID statement. The OUTSUM= data table is useful for summarizing results when large numbers of series are forecast.

BY Statement

BY *variables* ;

A BY statement can be used with PROC CARIMA to obtain separate dummy variable definitions for groups of observations that are defined by the BY *variables*.

ESTIMATE Statement

ESTIMATE < *options* > ;

The ESTIMATE statement specifies the type of ARIMA model to be fit for the *variable* in the preceding IDENTIFY statement.

You can specify the following *options*:

CONVERGE=*value*

specifies the convergence criterion, where *value* is a number between 0 and 1, exclusive. Convergence is assumed when the largest change in the estimate for any parameter is less than the specified *value*. If the absolute value of the parameter estimate is greater than 0.01, the relative change is used; otherwise, the absolute change in the estimate is used. By default, CONVERGE=0.001.

DELTA=*value*

specifies the perturbation value for computing numerical derivatives, where *value* is a number between 0 and 1, exclusive. By default, DELTA=0.001.

DIFF=(*value-list*)

specifies the degrees of differencing for the *variable-list* that is specified in the preceding IDENTIFY statement. For example, you can specify DIFF=(1) to take a first-order difference for all variables specified in the *variable-list* of the preceding IDENTIFY statement. If you have monthly data and expect seasonal patterns, you can specify DIFF=(1,12) to difference all variables in the *variable-list* of the preceding IDENTIFY statement twice: once at lag 1, and then again at lag 12. Each analyzed series is then the difference between the current period-to-period change in the specified variable and the change 12 periods ago. Each value in the *value-list* is an integer greater than 0. By default, DIFF=(0).

MAXITER=*value*

specifies the maximum number of iterations, where *value* is a positive integer. By default, MAXITER=50.

METHOD=*method-name*

specifies the estimation method to be used.

You can specify the following forecasting *method-names*:

- CLS** specifies the conditional least squares method.
- ML** specifies the maximum likelihood method.
- ULS** specifies the unconditional least squares method.

By default, METHOD=CLS.

MU=value

specifies a constant term for the ARIMA model, where *value* is a number. By default, MU=0.

NOINT=0 | 1

specifies whether to include an intercept in the model. You can specify the following values:

- 0** includes an intercept term in the model.
- 1** omits the intercept term from the model. This suppresses the fitting of a constant (or intercept) parameter in the model. That is, the value specified in the MU value is omitted.

By default, NOINT=0.

NOSTABLE=0 | 1

specifies whether to restrict the candidate estimates for the autoregressive and moving average parameters in the stationary and invertible regions, respectively. You can specify the following values:

- 0** restrict candidate estimates in stable and invertible regions.
- 1** do not restrict candidate estimates.

By default, NOSTABLE=0.

P=value

specifies the degree of the autoregressive polynomial, where *value* is an integer greater or equal than 0. By default, P=0.

Q=value

specifies the degree of the moving average polynomial, where *value* is an integer greater or equal than 0. By default, Q=0.

SINGULAR=value

specifies the criterion for checking singularity, where *value* is a number between 0 and 1, exclusive. If a pivot of a sweep operation is less than *value*, the matrix is deemed singular. Sweep operations are performed on the Jacobian matrix during final estimation and on the covariance matrix when preliminary estimates are obtained. The default is 1E-7.

TRANSFORM=transform-option

specifies the time series transformation to apply to the series in the ESTIMATE statement. You can specify the following *transform-options*:

- BOXCOX(*n*)** applies a Box-Cox transformation with parameter number, *n*, where *n* is a number between -5 and 5.
- LOG** applies a logarithmic transformation.
- LOGISTIC** applies a logistic transformation.
- NONE** does not apply a transformation.
- SQRT** applies a square-root transformation.

By default, TRANSFORM=NONE.

FORECAST Statement

FORECAST < options > ;

The FORECAST statement forecasts the future values of the variable list that is specified in the preceding IDENTIFY statement according to the model that is fitted in the preceding ESTIMATE STATEMENT.

You can specify any number of FORECAST statements. You can specify the following *options*:

ALPHA=*number*

specifies the significance level to use in computing the confidence limits of the forecast, where *number* must be between 0 and 1. By default, ALPHA=0.05, which produces 95% confidence intervals.

LEAD=*number*

specifies the *number* of periods ahead to forecast, where *number* must be an integer greater than or equal to 0. By default, LEAD=0.

ID Statement

ID *variable* **INTERVAL**=*interval* < options > ;

The ID statement names a numeric *variable* that identifies observations in the input and output data tables. The ID variable's values are assumed to be SAS date or datetime values. In addition, the ID statement specifies the (desired) frequency that is associated with the time series. The ID statement *options* also specify how to accumulate the observations and how to align the time ID values to form the time series to be forecast. The specified information affects all variables that are specified in subsequent IDENTIFY statements. If an ID statement is not specified, the observation number (with respect to the BY group) is used as the time ID.

You must specify the **INTERVAL**=*interval* option:

INTERVAL=*interval*

specifies the frequency of the input time series or the frequency for the time series to be accumulated from the input data. For example, if the input data consist of quarterly observations, then specify INTERVAL=QTR. For example, INTERVAL=QTR implies a seasonal cycle of length 4. If the ACCUMULATE= option is also specified, the INTERVAL= option determines the time periods for the accumulation of observations.

The basic intervals are YEAR, SEMIYEAR, QTR, MONTH, SEMIMONTH, TENDAY, WEEK, WEEKDAY, DAY, HOUR, MINUTE, SECOND. For more information about the intervals that can be specified, see the “Date Intervals, Formats, and Functions” chapter in *SAS/ETS User's Guide*.

You can also specify the following *options*:

ACCUMULATE=*accumulate-option*

specifies how to accumulate the observations within each time period. The frequency (width of each time interval) is specified in the INTERVAL= option. The ID variable contains the time ID values. Each time ID variable value corresponds to a specific time period. The accumulated values form the time series, which is used in subsequent model fitting and forecasting.

This option is particularly useful when there are gaps in the input data or when there are multiple input observations that coincide with a particular time period (for example, transactional data). The EXPAND

procedure in *SAS/ETS User's Guide* offers additional frequency conversions and transformations that can also be useful in creating a time series.

The following *accumulate-options* determine how the observations are accumulated within each time period on the basis of the ID variable and the frequency that is specified in the INTERVAL= option:

TOTAL SUM	accumulates observations based on the total sum of their values.
AVERAGE AVG	accumulates observations based on the average of their values.
MINIMUM MIN	accumulates observations based on the minimum of their values.
MAXIMUM MAX	accumulates observations based on the maximum of their values.
N	accumulates observations based on the number of nonmissing observations.
NMISS	accumulates observations based on the number of missing observations.
STDDEV STD	accumulates observations based on the standard deviation of their values.
CSS	accumulates observations based on the corrected sum of squares of their values.
USS	accumulates observations based on the uncorrected sum of squares of their values.

By default, ACCUMULATE=TOTAL.

If the ACCUMULATE= option is specified, the SETMISSING= option is useful for specifying how to treat accumulated missing values. If missing values should be interpreted as 0, then specify SETMISSING=0. For more information about accumulation, see the section “[Accumulation](#)” on page 125.

ALIGN=*align-option*

controls the alignment of SAS date or datetime values that are used to identify the time period of output observations. Although any date or datetime value within the time period can identify the time period, the ALIGN= option requests that the representative date or datetime for the time period be calculated as the beginning date or datetime of the time period, the ending date or datetime of the time period, or the middle date or datetime of the time period. In addition to aligning the time ID values consistently for observations you supply, the ALIGN= option specifies the method for calculating the time ID values for observations in the forecast and backcast time periods, which often you do not supply. You can specify the following *align-options*:

BEGINNING BEG B	represents each time period by using the beginning SAS date or datetime value of the time period.
ENDING END E	represents each time period by using the ending SAS date or datetime value of the time period.
MIDDLE MID M	represents each time period by using the middle SAS date or datetime value of the time period. The middle is calculated as the average of the beginning and ending values.

By default, ALIGN=BEGINNING.

END=*date | datetime*

specifies a SAS date or datetime literal value that represents the end of the data. If the last time ID *variable* value is less than the END= value, the series is extended with missing values. If the last time ID *variable* value is greater than the END= value, the series is truncated. For example, **END= '1jan2008'**D requests that data for time periods after the first of January 2008 not be used. The option **END= "&sysdate"**D uses the automatic macro variable SYSDATE to extend or truncate the series to the current date. This option and the START= option can be used to ensure that data associated with each BY group contain the same number of observations.

FORMAT=*format*

specifies the SAS format for the time ID values. If you omit this option, the default format is implied by the INTERVAL= option.

SETMISSING=*option*

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series. You can specify the following values to determine how to interpret missing values:

<i>n</i>	interprets a missing value as having the value <i>n</i> . You can specify any number for <i>n</i> , but not a missing value. If a missing value indicates a 0 value, specify SETMISSING=0. You typically use SETMISSING=0 for transactional data because the absence of recorded data usually implies no activity.
MISSING	interprets a missing value as a missing value. Use this option if a missing value indicates an unknown value.
AVERAGE AVG	interprets a missing value as the average value of all accumulated nonmissing values in the span of the series.
MINIMUM MIN	interprets a missing value as the minimum value of all accumulated nonmissing values in the span of the series.
MEDIAN MED	interprets a missing value as the median value of all accumulated nonmissing values in the span of the series.
MAXIMUM MAX	interprets a missing value as the maximum value of all accumulated nonmissing values in the span of the series.
FIRST	interprets a missing value as the first nonmissing value of all accumulated nonmissing values in the span of the series.
LAST	interprets a missing value as the last nonmissing value of all accumulated nonmissing values in the span of the series.
PREVIOUS PREV	interprets a missing value as the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.
NEXT	interprets a missing value as the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.

By default, SETMISSING=MISSING.

START=*date* | *datetime*

specifies a SAS date or datetime literal value that represents the beginning of the data. If the first time ID variable value is greater than the START= value, the series is prefixed with missing values. If the first time ID variable value is less than the START= value, the series is truncated. This option and the END= option can be used to ensure that data associated with each BY group contain the same number of observations.

TRIMID=*method*

specifies the method for trimming the data in the BY groups. The output time ID variable span that is calculated by the *method* is dependent on the input time ID variable span, irrespective of missing values of the time series variables. Depending on the *method* and the input time ID variable data, leading or trailing missing values can be added to the time series variables.

After the output time ID variable span is calculated by the *method*, the ending value of the output time ID variable is recalculated according to the value of the LEAD= option (if one is specified).

You can specify one of the following *methods*:

- NONE** uses the same starting and ending values of the output time ID variable for all BY groups. The span of the output time ID variable includes all values that are input as a time ID value for all BY groups. The time series variables are extended with leading or trailing missing values as required.
- LEFT** uses the identifying date for the first time period that is input for the BY group as the starting value of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading missing values. The ending value of the time ID variable is the same for all BY groups. The time series variables are extended with trailing missing values as required.
- RIGHT** uses the identifying date for the last time period that is input for the BY group as the ending value of the output time ID variable for each BY group. The time series values in each BY group are not extended with trailing missing values. The starting value of the time ID variable is the same for all BY groups. The time series variables are extended with leading missing values as required.
- BOTH** uses the span of the input time ID variable for the BY group as the span of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading or trailing missing values.

By default, TRIMID=NONE.

IDENTIFY Statement

IDENTIFY *variable-list* ;

The identify statement specifies the variables you want to model. Specify one or more variables in a space-separated *variable-list*.

Details: CARIMA Procedure

The CARIMA procedure can be used to fit ARIMA models either to time series data or to as transactional data. If the data are transactional, then the procedure must first accumulate the data into a time series before it can forecast them. The procedure uses the sequential steps in [Table 5.2](#) to produce forecasts, with the options that control the step listed to the right.

Table 5.2 PROC CARIMA Processing Steps and Control Options

Step	Operation	Options	Statement
1	Accumulation	ACCUMULATE=	ID
2	Missing value interpretation	SETMISSING=	ID
3	Parameter estimation	P=, Q=	ESTIMATE
4	Inverse transformation	TRANSFORM, DIFF	ESTIMATE
5	Forecasting	LEAD=	FORECAST

Each of the steps shown in [Table 5.2](#) is described in the following sections.

Accumulation

If the ACCUMULATE= option is specified in the ID statement, data table observations are accumulated within each time period. The frequency (width of each time interval) is specified in the INTERVAL= option, and the ID variable contains the time ID values. Each time ID value corresponds to a specific time period. Accumulation is particularly useful when the input data table contains transactional data, whose observations are not spaced with respect to any particular time interval. The accumulated values form the time series that is used in subsequent analyses by the CARIMA procedure.

For example, suppose a data table contains the following observations:

```
19MAR1999    10
19MAR1999    30
11MAY1999    50
12MAY1999    20
23MAY1999    20
```

If the INTERVAL=MONTH option is specified in the ID statement, all the preceding observations fall within three time periods: March 1999, April 1999, and May 1999. The observations are accumulated within each time period as follows:

- If the ACCUMULATE=NONE option is specified, an error is generated because the ID variable values are not equally spaced with respect to the specified frequency (MONTH).
- If the ACCUMULATE=TOTAL option is specified, the resulting time series is

```
O1MAR1999    40
O1APR1999    .
O1MAY1999    90
```

- If the ACCUMULATE=AVERAGE option is specified, the resulting time series is

```
O1MAR1999    20
O1APR1999    .
O1MAY1999    30
```

- If the ACCUMULATE=MINIMUM option is specified, the resulting time series is

```
O1MAR1999    10
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=MEDIAN option is specified, the resulting time series is

```
O1MAR1999    20
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=MAXIMUM option is specified, the resulting time series is

```
O1MAR1999    30
O1APR1999    .
O1MAY1999    50
```

- If the ACCUMULATE=FIRST option is specified, the resulting time series is

```
O1MAR1999    10
O1APR1999    .
O1MAY1999    50
```

- If the ACCUMULATE=LAST option is specified, the resulting time series is

```
O1MAR1999    30
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=STDDEV option is specified, the resulting time series is

O1MAR1999	14.14
O1APR1999	.
O1MAY1999	17.32

As you can see from the preceding examples, even though the data table observations contained no missing values, the accumulated time series can have missing values.

Missing Value Interpretation

After the data have been accumulated to form a time series on the basis of the `INTERVAL=` and the `ACCUMULATE=` options, missing values are interpreted. Sometimes missing values should be interpreted as unknown values. In this case, you can specify the `SETMISSING=` option to interpret how to treat missing values. In other situations, missing values are known, such as when missing values are created from accumulation and the absence of an observation should be interpreted the observation having the value 0. In this case, you can specify `SETMISSING=0` in order to treat missing observations as 0 values. In other cases, missing values should be interpreted as global values, such as minimum or maximum values of the accumulated series. The accumulated and interpreted time series is used in subsequent analyses.

Parameter Estimation

The default method for estimating parameters is conditional least squares. You can also use maximum likelihood estimation and unconditional least squares, by specifying the corresponding value in the `METHOD=` option in the `ESTIMATE` statement.

Transformations

If the `TRANSFORM=` option is specified in the `ESTIMATE` statement, the time series is transformed before the parameters are estimated and the series is forecast. Only strictly positive series can be transformed. An error is generated when the `TRANSFORM=` option is used with a nonpositive series.

Forecasting

After the model parameters are estimated, one-step-ahead forecasts are generated for the full range of the accumulated and optionally transformed time series data, and multistep forecasts are generated from the end of the time series to the future time period as specified in the `LEAD=` option. If there are missing values at the end of the time series, the forecast horizon will be greater than that specified by the `LEAD=` option.

Data Table Output

The CARIMA procedure can create the OUT=, OUTEST=, OUTFOR=, and OUTSUM= data tables. These data tables contain the variables that are listed in the BY, ID, and IDENTIFY statements, in addition to statistics that are related to the parameters estimated by the ESTIMATE statement. In general, if a forecasting step that is related to an output data table fails, the values of that step are not recorded or are set to missing in the related output data table and appropriate error or warning messages (or both) are recorded in the log.

OUT= Data Table

The OUT= data table contains the variables that are specified in the BY, ID, and IDENTIFY statements. If the ID statement is specified, its variable values are aligned and extended according to the values of ALIGN= and INTERVAL= options. The values of the variables that are specified in the IDENTIFY statements are accumulated according to the values of the ACCUMULATE= option, and missing values are interpreted according to the values of the SETMISSING= option.

If any forecasting steps fail for a particular variable, the variable is extended by missing values.

OUTEST= Data Table

The OUTEST= data table contains the variables that are specified in the BY and IDENTIFY statements and the variables in the following list.

NAME	Variable name
MODEL	Forecasting model
MODELVAR	Model variable
DSVAR	Data table variable
VARTYPE	Variable type
TRANSFORM	Transformation
COMPONENT	Component
COMPMODEL	Component model
PARM	Parameter name
FACTOR	Factor
LAG	Lag
SHIFT	Shift
EST	Value of parameter estimate
STDERR	Standard error of parameter estimate
TVALUE	<i>t</i> values of parameter estimate
STDERR	<i>p</i> -values of parameter estimate

If the parameter estimation step fails for a particular variable, no observations are output to the OUTEST= data table for that variable.

OUTFOR= Data Table

The OUTFOR= data table contains the variables that are specified in the BY and IDENTIFY statements and the variables in the following list.

NAME	Variable name
ID Variable	Values of the variable specified in the ID statement
ACTUAL	Actual values
PREDICT	Predicted values
ERROR	Prediction errors
STD	Prediction standard errors
UPPER	Upper confidence limits
LOWER	Lower confidence limits

If the forecasting step fails for a particular variable, no observations are recorded in the OUTFOR= data table for that variable. If the TRANSFORM= option is specified, the values in the preceding IDENTIFY statement are the inverse transform forecasts. If the MEDIAN option is specified, the median forecasts are stored; otherwise, the mean forecasts are stored.

OUTSUM= Data Table

The OUTSUM= data table contains the variables that are specified in the BY statement and the variables in the following list. The OUTSUM= data table records the summary statistics for each variable that is specified in an IDENTIFY statement.

The following variables that are related to summary statistics are based on the ACCUMULATE= and SETMISSING= options:

NAME	Variable name
NOBS	Number of observations
N	Number of nonmissing observations
NMISS	Number of missing observations
MIN	Minimum value
MAX	Maximum value
MEAN	Mean value
STDDEV	Standard deviation
STATUS	forecasting status (nonzero values imply that no forecast was generated for the series)
START	Starting date of each series
END	Ending date of each series
STARTOBS	Beginning observation number of each series
ENDOBS	Ending observation number of each series

Printed Output

The CARIMA procedure always prints a summary of the processing that is performed on the time series data. This is extremely useful for gauging the work that is performed by the CAS server when it executes PROC CARIMA.

The CARIMA procedure also prints by default the parameter estimates of fitted models in the ParameterEstimates table and the multistep-ahead variable forecasts in the Forecasts table. The name of the modeled variable is printed in the VariableInfo table.

You can also print parameter estimates, variable forecasts, and other summary statistics by using targeted data queries to subset and display the information from the tables that PROC CARIMA produces. For example, you might want to print results in the OUTFOR= data table only for BY groups that have a particular _STATUS_ value in the OUTSUM= table.

ODS Table Names

The CARIMA procedure assigns a name to each table that it creates. You can use this name to refer to the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in [Table 5.3](#).

Table 5.3 ODS Tables Produced in the CARIMA Procedure

ODS Table Name	Description	Option
Forecasts	Forecast components, such as prediction, prediction standard error, lower and upper confidence limits	Default
OutInfo	Summary of the processing that is performed on the time series data	Default
ParameterEstimates	Parameter estimates	Default
VariableInfo	Name of the variable specified in the IDENTIFY statement	Default

Example: CARIMA Procedure

Example 5.1: Seasonal Model for the Airline Series

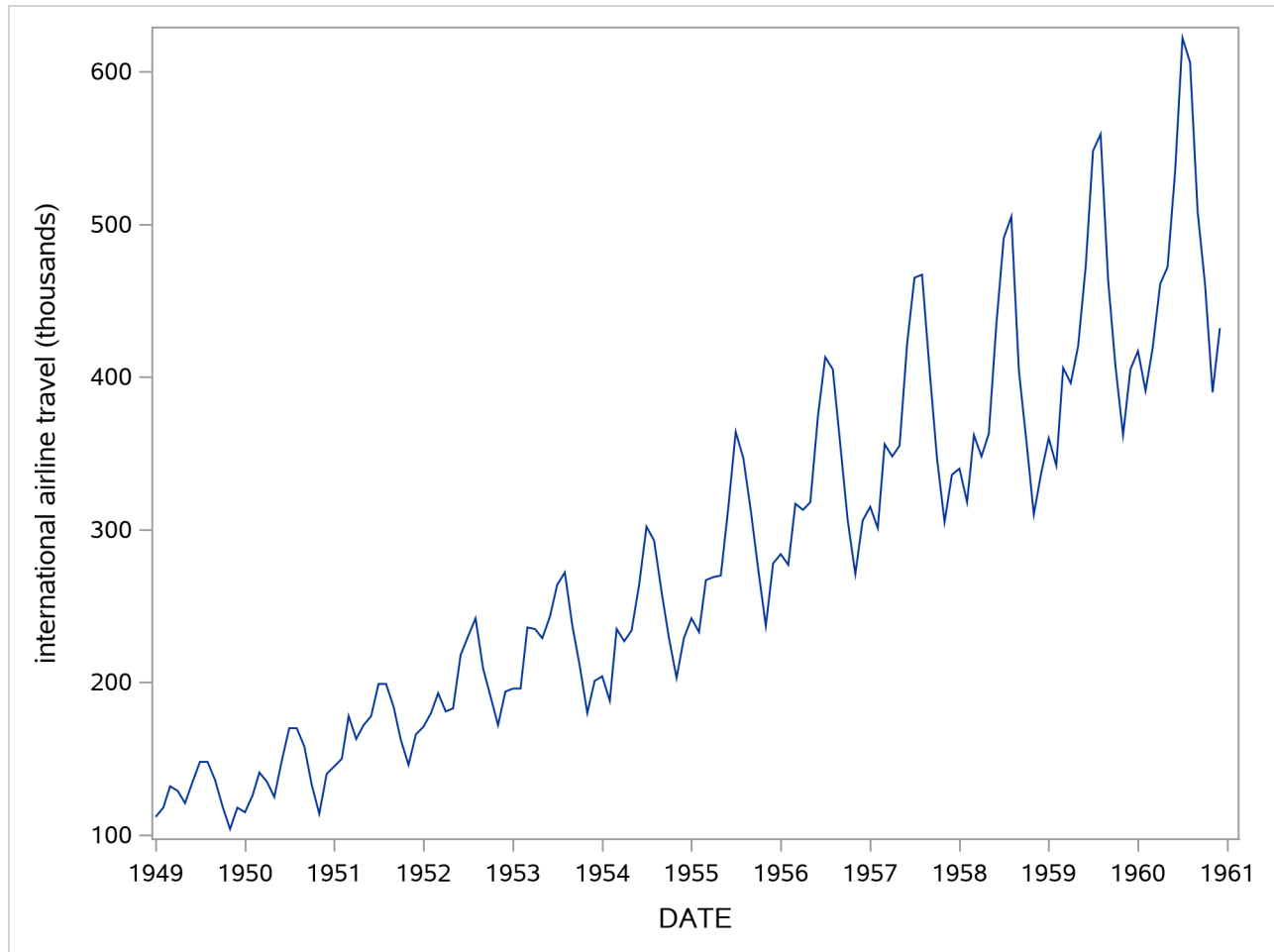
The airline passenger data have been used in time series analysis literature as an example of a nonstationary seasonal time series.

The following statements plot the air data:

```
proc sgplot data=sashelp.air;  
  series x = date y = air;  
run;
```

Output 5.1.1 indicates that the series is nonstationary and seasonal.

Output 5.1.1 Airline Passenger Data



The following DATA step loads the air data into a CAS session. The DATA step assumes that the SAS library is named mylib, but you can substitute any appropriately named SAS library.

```
data mylib.myair;
  set sashelp.air;
run;
```

You can use the following statements to fit an $ARIMA(0,1,1) \times (0,1,1)_{12}$ model without a mean term to the logarithms of the airline passengers series. The model estimates and forecasts are stored in the data tables `mylib.cariest` and `mylib.carifor`.

```
proc carima data=mylib.myair outest=mylib.cariest outfor=mylib.carifor;
  id date interval=month;
  identify air;
  estimate q=(1)(12) noint method = ML transform = log diff = (1,12);
  forecast lead = 4;
run;
```

Output 5.1.2 shows the summary of the time series processing.

Output 5.1.2 Summary of Time Series Processing
The CARIMA Procedure

Processing Summary	
CAS table	MYAIR
Number of analysis variables	1
Number of rows read	288
Number of groups read	1
Memory for group packages (KB)	15
Time to load groups (seconds)	0.0108208656
Minimum time ID	JAN1949
Maximum time ID	DEC1960
Minimum time periods	144
Maximum time periods	144
Number of nodes run	4
Number of nodes with data	3
Number of nodes with groups	1
Number of threads budgeted	1
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	1
Maximum threads active	1
Number of groups processed by submitted code	1
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0632238388

Output 5.1.3 shows the name of the modeled variable.

Output 5.1.3 Variable Information

Variable Info	
Variable Name	AIR

Output 5.1.4 shows the parameter estimates from the fitted model.

Output 5.1.4 Parameter Estimates

Maximum Likelihood Estimates					
Parameter	Estimate	Standard Error	t Value	Pr > t	Lag
MA1_1	0.401936	0.079877	5.03	<.0001	1
MA2_12	0.556862	0.084027	6.63	<.0001	12

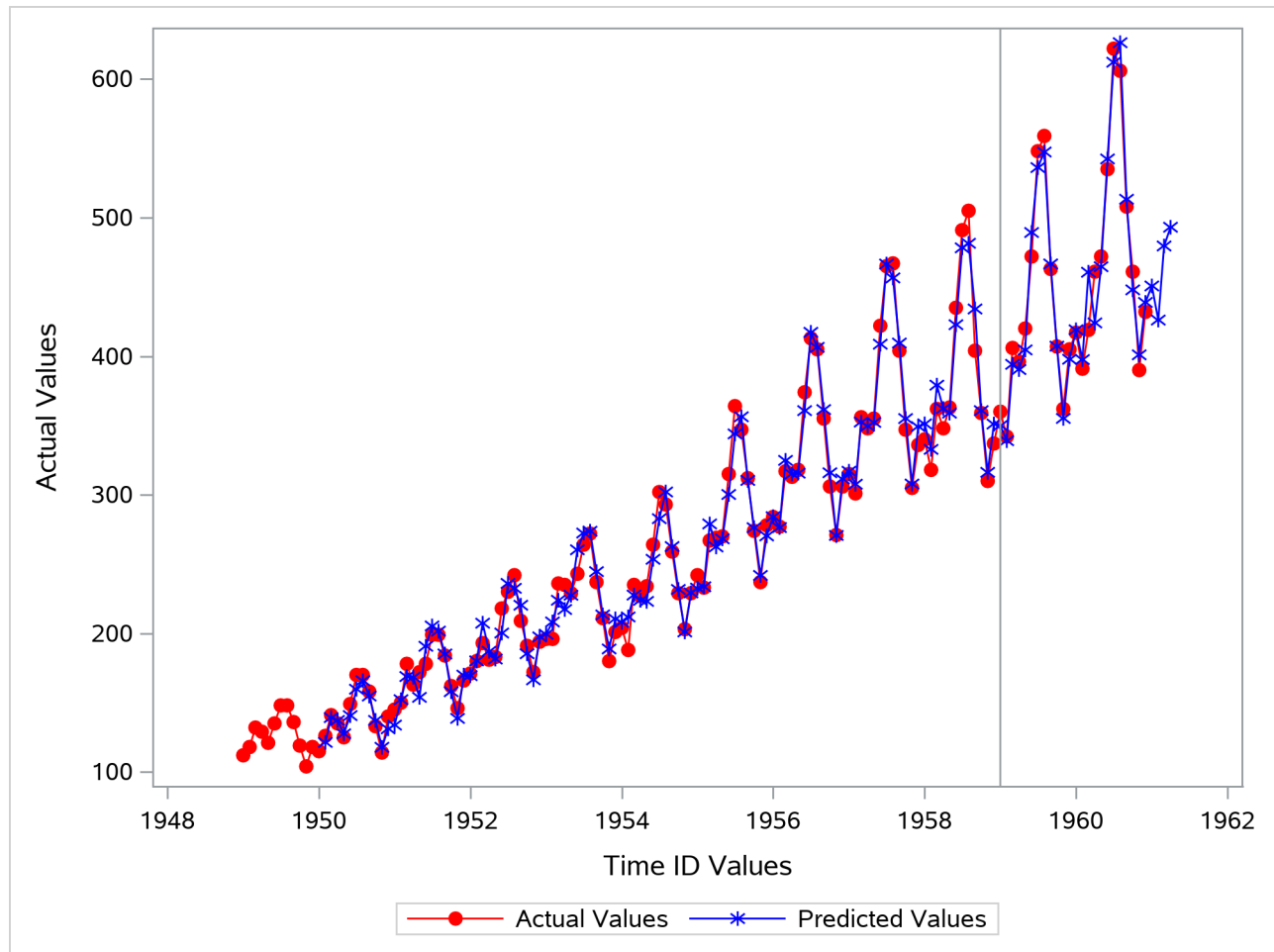
Output 5.1.5 shows the multistep-ahead variable forecasts.

Output 5.1.5 Variable Forecasts

Forecasts for AIR using Model 1				
95%				
Time ID	Forecast	Std Error	Confidence Limits	
JAN1961	450.7320	16.6829	418.9155	484.3014
FEB1961	426.1129	18.3793	391.2224	463.2533
MAR1961	479.5664	23.2533	435.6018	526.7297
APR1961	493.1052	26.2879	443.5853	546.5997

The following statements plot the forecasts that are produced by the $\text{ARIMA}(0,1,1) \times (0,1,1)_{12}$ model. The plot is shown in Output 5.1.6.

```
proc sgplot data=mylib.carifor;
  series x=date y=actual / markers lineattrs=(color=red)
                                markerattrs=(symbol=circlefilled color=red);
  series x=date y=predict / markers lineattrs=(color=blue)
                                markerattrs=(symbol=asterisk color=blue);
  refline '01JAN1959'd / axis=x;
run;
```

Output 5.1.6 Airline Passengers Forecasts

References

Box, G. E. P., and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*. Rev. ed. San Francisco: Holden-Day.

Chapter 6

CAUSALDISCOVERY Procedure

Contents

Overview: CAUSALDISCOVERY Procedure	136
Using CAS Sessions and CAS Engine Librefs	136
Getting Started: CAUSALDISCOVERY Procedure	137
Syntax: CAUSALDISCOVERY Procedure	140
Functional Summary	141
PROC CAUSALDISCOVERY Statement	142
DISPLAY Statement	143
DISPLAYOUT Statement	144
EVALUATE Statement	144
LEARN Statement	145
ROOTVAR Statement	147
SIMULATE Statement	147
VAR Statement	149
Details: CAUSALDISCOVERY Procedure	149
DAG Structure Learning Overview	149
Topological Order Permutation (TOP) Algorithm	153
Minimizing Conditional Variance (MCV) Algorithm	155
Causal Discovery with Panel Data	155
Missing Values	156
Data Table Output	156
ODS Table Names	158
Examples: CAUSALDISCOVERY Procedure	159
Example 6.1: Simulation Study of DAG Structure Learning	159
Example 6.2: Learning a Large DAG by Using the MCV Algorithm	163
Example 6.3: Using Causal Discovery on Panel Data with Fixed Effects	165
References	170

Overview: CAUSALDISCOVERY Procedure

The directed acyclic graph (DAG) is critical for causal inference, which has been widely applied in many fields, including machine learning, artificial intelligence (AI), economics, finance, medicine, and so on. However, usually the structure of a DAG must be provided by experts, because learning the structure of a DAG is a well-known NP-hard problem. The CAUSALDISCOVERY procedure supports two algorithms for DAG structure learning: the topological order permutation (TOP) algorithm and the minimizing conditional variance (MCV) algorithm. Both algorithms are fast and accurate; they can reach 100% accuracy in many cases, even when there are hundreds of variables.

In addition to DAG structure learning, PROC CAUSALDISCOVERY can simulate a DAG (ground truth) and generate samples that are based on the simulated DAG. It can also evaluate the accuracy of the learned DAG by computing several metrics for comparing the estimated DAG to the ground truth.

Because PROC CAUSALDISCOVERY runs on SAS Cloud Analytic Services (CAS), it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the `CASHOST` and `CASPORT` of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the `TERMINATE` option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 54 in Chapter 4, “Shared Concepts.”

Getting Started: CAUSALDISCOVERY Procedure

This example illustrates how the CAUSALDISCOVERY procedure estimates the DAG from which the data are generated, in order to solve a seven-variable DAG structure learning problem.

The following DATA step produces the simulated data set to be used in the example:

```
data one;
  input x1 - x7;
datalines;
  0.6712068564    -0.2826079580    -0.5349256450    -0.8261341320
 -3.0394109400    -1.8825135390     5.2510795861
 -0.7107886740    -2.4798707050     1.2698001196    -6.2078732110
 -14.3827270100   -11.0299665000    20.9643672460

... more lines ...

 -0.5282917840    0.4435880321     -0.6905104940    0.4259155417
 -0.1943036260    0.6881816788     -0.6409083280
 -0.3955187900    -0.5117192740    -1.0742776010    0.7085965238
 2.1642802138     1.5589762381     -2.7343111790
;
```

What is the relationship among these seven variables? That is the question that the DAG structure learning is trying to answer. Note that as the number of variables increases, the number of DAGs increases superexponentially. When the number of variables is seven, there are more than one billion seven-variable DAGs. Finding the one true DAG from which the data are generated is not easy. However, PROC CAUSALDISCOVERY might help you find that DAG in seconds or even less than a second.

The following DATA step loads the data set one into the data table one in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.one; set one; run;
```

The following code inputs the data set, specifies the variables, and names the output data table. The INITMETHOD=DATA option in the LEARN statement shows whether there is any difference between the topological order of the final estimated DAG and the order that is specified in the VAR statement.

```
proc causaldiscovery;
  var x1 - x7;
  learn data=mylib.one / initmethod=data out=mylib.oest;
run;
```

The number of observations is shown in [Figure 6.1](#).

Figure 6.1 Number of Observations
The CAUSALDISCOVERY Procedure

Number of Observations Read	10000
Number of Observations Used	10000

The algorithm information is shown in Figure 6.2.

Figure 6.2 Algorithm Information
The CAUSALDISCOVERY Procedure

Algorithm Information	
Algorithm	Topological Order Permutation
Initialization Method	Data
Maximum Iterations	200
Parallel	Yes
Seed	1

The topological orders are shown in Figure 6.3. This table shows that there is a difference between the topological order of the final estimated DAG and the order that is specified in the VAR statement.

Figure 6.3 Initial and Final Topological Orders

Topological Orders		
Variable	Initial Order	Final Order
x1	1	1
x2	2	3
x3	3	2
x4	4	4
x5	5	6
x6	6	5
x7	7	7

The estimated full DAG that is expressed as the adjacency matrix is shown in Figure 6.4. In an adjacency matrix $W = (w_{ij})$, element w_{ij} is either 0 or 1. If $w_{ij} = 0$, this means that there is no edge from variable i to variable j ; otherwise, there is an edge from variable i to variable j (that is, variable i is a parent of variable j , or variable j is a child of variable i).

Figure 6.4 Estimated Full Adjacency Matrix

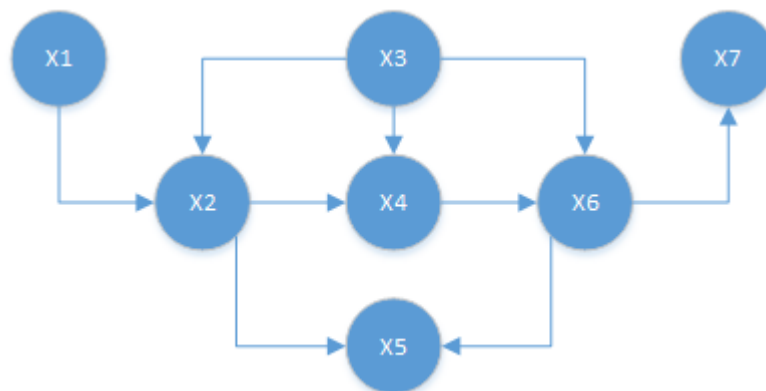
Estimated Full Adjacency Matrix							
Variable	x1	x2	x3	x4	x5	x6	x7
x1	0	1	1	1	1	1	1
x2	0	0	0	1	1	1	1
x3	0	1	0	1	1	1	1
x4	0	0	0	0	1	1	1
x5	0	0	0	0	0	0	1
x6	0	0	0	0	1	0	1
x7	0	0	0	0	0	0	0

The estimated DAG based on the significance level 0.0001% is shown in [Figure 6.5](#).

Figure 6.5 Estimated Adjacency Matrices Based on the Significance Level 0.0001%

Estimated Adjacency Matrices								
Alpha	Variable	x1	x2	x3	x4	x5	x6	x7
1E-6	x1	0	1	0	0	0	0	0
	x2	0	0	0	1	1	0	0
	x3	0	1	0	1	0	1	0
	x4	0	0	0	0	0	1	0
	x5	0	0	0	0	0	0	0
	x6	0	0	0	0	1	0	1
	x7	0	0	0	0	0	0	0

As shown in the adjacency matrix in [Figure 6.5](#), X1 and X3 have no parents; X2's parents are X1 and X3; X4's parents are X2 and X3; X5's parents are X2 and X6; X6's parents are X3 and X4; and X7's parent is X6. [Figure 6.6](#) shows this seven-variable DAG.

Figure 6.6 Estimated DAG

In fact, this estimated DAG is the true DAG for the data generating process, as shown in the following SAS code:

```

data one;
  call streaminit('pcg', 12345);
  x1 = 0; x2 = 0; x3 = 0; x4 = 0;
  x5 = 0; x6 = 0; x7 = 0;
  do i = 1 to 10000;
    x1 = rand('Normal');
    x3 = rand('Normal');
    x2 = x1 - 0.5*x3 + rand('Normal');
    x4 = 1.5*x2 - x3 + rand('Normal');
    x6 = 0.5*x3 + 2*x4 + rand('Normal');
    x5 = -x2 + 1.5*x6 + rand('Normal');
    x7 = -2*x6 + rand('Normal');
    output;
  end;
  drop i;
run;

```

Syntax: CAUSALDISCOVERY Procedure

The following statements are available in the CAUSALDISCOVERY procedure. Items within angle brackets (<>) are optional.

```

PROC CAUSALDISCOVERY ;
  DISPLAY <table-list> </options> ;
  DISPLAYOUT table-spec-list </options> ;
  EVALUATE <ESTDAG=libref.data-table> <TRUEDAG=libref.data-table> </options> ;
  LEARN <DATA=libref.data-table> <OUTDAG=libref.data-table> </options> ;
  ROOTVAR variable-list ;
  SIMULATE <OUT=libref.data-table> <OUTDATA=libref.data-table> </options> ;
  VAR variable-list ;

```

The PROC CAUSALDISCOVERY and VAR statements are required. Table 6.1 lists the required options in each statement and in combinations of statements. Note that the procedure does nothing if you do not specify a SIMULATE, LEARN, or EVALUATE statement. By default, the procedure uses output tables from one statement as input for another statement when you specify any of these three statements together. If you specify an input table in the Omitted Options column, it is ignored unless it existed on the libref before the procedure call.

Table 6.1 Required Options in the CAUSALDISCOVERY Procedure

Statement(s)	Required Options	Omitted Options
EVALUATE	ESTDAG=	
LEARN	TRUEDAG=	
LEARN and EVALUATE	DATA=	
	DATA= in LEARN	ESTDAG= in EVALUATE
	OUTDAG= in LEARN	

Table 6.1 *continued*

Statement(s)	Required Options	Omitted Options
SIMULATE	TRUEDAG= in EVALUATE	
SIMULATE and EVALUATE	None	
SIMULATE and LEARN	ESTDAG= in EVALUATE	TRUEDAG= in EVALUATE
SIMULATE , LEARN , and EVALUATE	OUT= in SIMULATE	
	OUTDATA= in SIMULATE	DATA= in LEARN
	OUT= in SIMULATE	DATA= in LEARN
	OUTDATA= in SIMULATE	ESTDAG= in EVALUATE
	OUTDAG= in LEARN	TRUEDAG= in EVALUATE

Functional Summary

The statements and options available in the CAUSALDISCOVERY procedure are summarized in Table 6.2. The sections after the table describe each statement in alphabetical order, following the PROC CAUSALDISCOVERY statement.

Table 6.2 Functional Summary

Description	Statement	Option
Input Data Table Options		
Specifies the input table containing the estimated DAG structure	EVALUATE	ESTDAG=
Specifies the input data table containing the true adjacency matrix	EVALUATE	TRUEDAG=
Specifies the input data table	LEARN	DATA=
Output Data Table Options		
Writes the accuracy metrics to an output data set	EVALUATE	OUT=
Writes the initial and final topological orders, as well as the matrices of parameter estimates, standard deviations, <i>t</i> values, and <i>p</i> -values to an output data set	LEARN	OUT=
Writes the learned DAG to an output data set	LEARN	OUTDAG=
Writes the iteration history to an output data set	LEARN	OUTITER=
Writes the parents and children of variables and paths of the estimated DAGs to an output data set	LEARN	OUTPATH=
Writes the simulated DAG to an output data set	SIMULATE	OUT=
Writes the simulated data to an output data set	SIMULATE	OUTDATA=
Writes the topological order of the simulated DAG to an output data set	SIMULATE	OUTORDER=
Writes the parameter values to an output data set	SIMULATE	OUTPARAM=
Variables		
Specifies the variables to be generated based on the DAG	VAR	

Table 6.2 *continued*

Description	Statement	Option
Root Variables		
Specifies the assumed root variables of the DAG	ROOTVAR	
Learning Algorithm Options		
Specifies the DAG structure learning algorithm	LEARN	ALGORITHM=
Specifies that the DAG structure learning algorithm perform calculations by using graphics processing unit (GPU) hardware	LEARN	GPU
Specifies the method to use for determining the initial topological order	LEARN	INITMETHOD=
Specifies the maximum number of iterations	LEARN	MAXITER=
Specifies the random number seed value for the random number generator	LEARN	SEED=
Specifies that all structure learnings are to be performed in a single thread	LEARN	NONPARALLEL
Simulation Options		
Specifies the number of edges	SIMULATE	NEDGES=
Specifies the sample size	SIMULATE	NOBS=
Specifies how the noise terms are generated	SIMULATE	NOISE()
Specifies how the parameter values are generated	SIMULATE	PARM()
Specifies the seed value for the random number generator	SIMULATE	SEED=
Options to Control the Estimated DAGs		
Specifies the significance levels on whose basis the DAG edges are determined	LEARN	ALPHA=
Output Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	

PROC CAUSALDISCOVERY Statement

PROC CAUSALDISCOVERY ;

The PROC CAUSALDISCOVERY statement invokes the CAUSALDISCOVERY procedure.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 158. A path is a table name that is prefixed with dot-separated grouping information. For example, a TopoOrders table that the procedure produces has the path *Causaldiscovery.TopoOrders*. A partial pathname does not include all groups; for example, *TopoOrders* is a partial pathname for *Causaldiscovery.TopoOrders*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, *TopoOrders* selects *Causaldiscovery.TopoOrders*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/order/” selects all pathnames that contain the substring “order”; in particular, the *Causaldiscovery.TopoOrders* table is selected. Specifying “!/order/” selects all pathnames that do not contain the substring “order”; in particular, the *Causaldiscovery.TopoOrders* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the output table name.

key specifies *key* as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 158. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

EVALUATE Statement

EVALUATE < **ESTDAG**=*libref.data-table* > < **TRUEDAG**=*libref.data-table* > < / *options* > ;

The EVALUATE statement specifies the options that are related to the accuracy evaluation of the DAG structure learning. You can specify the following options:

ESTDAG=*libref.data-table*

ESTADJ=*libref.data-table*

specifies the input data table that contains the adjacency matrix for the estimated DAGs. If you specify both the EVALUATE and LEARN statements, you should specify the OUTDAG= option table in the LEARN statement but not specify the ESTDAG= option table in the EVALUATE statement. This is because the OUTDAG= output table is used as the ESTDAG= input table in the EVALUATE statement. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 136.

TRUEDAG=*libref.data-table*

TRUEADJ=*libref.data-table*

specifies the input data table that contains the adjacency matrix for the true DAG. If you specify both the EVALUATE and SIMULATE statements, you should specify the OUT= option table in the SIMULATE statement but not specify the TRUEDAG= option table in the EVALUATE statement. This is because the OUT= output table is used as the TRUEDAG= input table in the EVALUATE statement. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

You can also specify the following *option* after a slash (/):

OUT=*libref.data-table*

writes the accuracy metrics to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

LEARN Statement

LEARN < **DATA=***libref.data-table* > < **OUTDAG=***libref.data-table* > < / *options* > ;

The LEARN statement specifies the options that are related to the DAG structure learning. You can specify the following options:

DATA=*libref.data-table*

specifies the input data table. If you specify both the LEARN and SIMULATE statements, you should specify the OUTDATA= option table in the SIMULATE statement but not specify the DATA= option table in the LEARN statement. This is because the OUTDATA= output table is used as the DATA= input table in the LEARN statement.

OUTDAG=*libref.data-table*

writes the estimated DAGs to the specified output data table. You must specify this option if you also specify the EVALUATE statement; this table is then used as the ESTDAG= input table in the EVALUATE statement. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136. .

You can also specify the following *options* after a slash (/):

ALGORITHM=MCV | TOP

ALG=MCV | TOP

specifies the DAG structure learning algorithm. You can specify the following values:

MCV specifies the minimizing conditional variance (MCV) algorithm.

TOP specifies the topological order permutation (TOP) algorithm.

By default, ALGORITHM=TOP.

ALPHA=number | (number-list)

specifies the significance level on whose basis the edges in the DAG are determined. If you specify a *number-list*, several DAGs are included in the OUTDAG= option table, one for each value in the *number-list*. The *number* and all numbers in the *number-list* must be between 0 and 1. By default, ALPHA=0.000001.

GPU

specifies that the DAG structure learning algorithm perform calculations by using graphics processing unit (GPU) hardware. When ALGORITHM=MCV, this option is ignored.

INITMETHOD=DATA | RANDOM | VARIANCE

specifies the method to use for determining the initial topological order. You can specify the following values:

DATA	matches the order of the variables in the VAR statement.
RANDOM	determines the order randomly.
VARIANCE	determines the order by using the ascending order of variance of variables in the VAR statement.

By default, INITMETHOD=VARIANCE.

MAXITER=number

specifies the maximum number of iterations. The *number* must be a nonnegative integer. By default, MAXITER=200.

NONPARALLEL

specifies that all structure learnings are to be performed in a single thread on a single machine. If you omit this option, the structure learning algorithm uses as many threads or CPUs as possible.

OUT=libref.data-table

writes the initial and final topological orders, as well as the matrices of parameter estimates, standard errors, *t* values, and *p*-values to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

OUTITERHIST=libref.data-table**OUTITER=libref.data-table**

writes the iteration history to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

OUTPATH=libref.data-table**OUTP=libref.data-table**

writes the parents and children of variables and paths of the estimated DAGs to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

SEED=number

specifies the random number seed value for the random number generator. If the seed value is 0, a random number stream is generated by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences. By default, SEED=1.

ROOTVAR Statement

ROOTVAR *variable-list* ;

The ROOTVAR statement specifies the root variable names in the input data table that you specify in the DATA= option in the LEARN statement. Root variables are variables that have no parents or incoming edges. They serve only as potential parents of other variables during the causal discovery process.

SIMULATE Statement

SIMULATE < **OUT=***libref.data-table* > < **OUTDATA=***libref.data-table* > < / *options* > ;

The SIMULATE statement specifies the options to be used to first simulate a DAG and then generate the observations from the simulated DAG. You can specify the following options:

OUT=*libref.data-table*

OUTDAG=*libref.data-table*

writes the simulated DAG to the specified output data table. You must specify this option if you also specify the EVALUATE statement; this table is then used as the TRUEDAG= input table in the EVALUATE statement. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

OUTDATA=*libref.data-table*

writes the simulated data to the specified output data table. You must specify this option if you also specify the LEARN statement; this table is then used as the DATA= input table in the LEARN statement. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

You can also specify the following *options* after a slash (/):

NEDGES=*number*

E=*number*

specifies the number of edges in the DAG. The *number* must be a nonnegative integer. By default, $NEDGES = \min(d(d-1)/2, 2d)$, where d is the number of nodes or variables.

NOBS=*number*

N=*number*

SAMPLESIZE=*number*

specifies the number of observations in the OUTDATA= option table. The *number* must be a positive integer. By default, NOBS=10000.

NOISE(DIST=distribution)

specifies how the noise terms in the structural causal model are generated. You can specify the following *distribution*:

GAUSSIAN(gaussian-options)**NORMAL(gaussian-options)**

specifies the Gaussian distribution for the noise. You can specify the following *gaussian-options*:

MEAN=number

specifies the mean for the distribution. By default, MEAN=0.

STD=number

specifies the standard deviation for the distribution. By default, STD=1.

VAR=number

specifies the variance for the distribution. By default, VAR=1.

NOTE: You can specify the STD= option or the VAR= option, but not both.

OUTORDER=libref.data-table

writes the topological order of the simulated DAG to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

OUTPARAM=libref.data-table

writes the parameter values to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 136.

PARM(DIST=distribution)

specifies how to generate the parameter values that represent the weights of the edges. You can specify the following *distribution*:

UNIFORM(uniform-options)

specifies the uniform distribution for the parameter values. You can specify the following *uniform-options*:

LB=number

specifies the lower bound of the distribution. By default, LB=-1.

EXCLLB=number

specifies the lower bound of the excluded range. By default, EXCLLB=-0.5.

EXCLUB=number

specifies the upper bound of the excluded range. By default, EXCLUB=0.5.

UB=number

specifies the upper bound of the distribution. By default, UB=1.

The range of the uniform distribution is (LB, EXCLLB] U [EXCLUB, UB). The middle (EXCLLB, EXCLUB) range is excluded to avoid numerical instability in the simulated data set, as is common practice in the DAG literature.

SEED=number

specifies the random number seed value for the random number generator. If the seed value is 0, a random number stream is generated by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences. By default, SEED=1.

VAR Statement

VAR *variable-list* ;

The VAR statement specifies the variable names in the input data table that you specify in the DATA= option in the LEARN statement. If you omit the DATA= option in the LEARN statement, or if you omit the LEARN statement, the VAR statement creates the list of variable names for the SIMULATE and EVALUATE statements.

Details: CAUSALDISCOVERY Procedure

DAG Structure Learning Overview

A graph consists of nodes (also known as vertices) and edges. A directed acyclic graph (DAG) is a type of graph such that (1) the edge has direction from one node to another, and (2) if you start from one node and travel along the directed edges, you cannot come back to the starting node; that is, there is no directed loop. [Figure 6.7](#) shows an example of a DAG. By contrast, the graph in [Figure 6.8](#) is not a DAG because the edge between X1 and X3 has no direction. The graph in [Figure 6.9](#) is not a DAG because there is a directed loop from X1 to X3, then to X2, and then back to X1. The graph in [Figure 6.10](#) is not a DAG because there is a directed edge from X5 to X5 itself, which makes a directed loop.

Figure 6.7 Example of a DAG

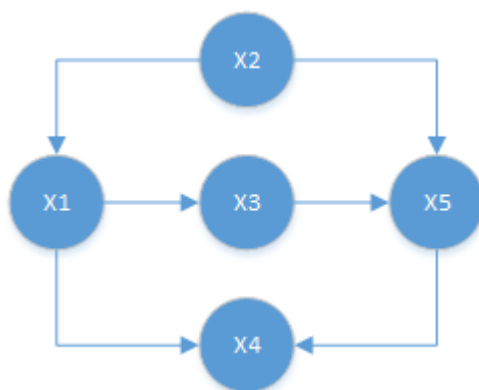


Figure 6.8 First Example of a Graph That Is Not a DAG

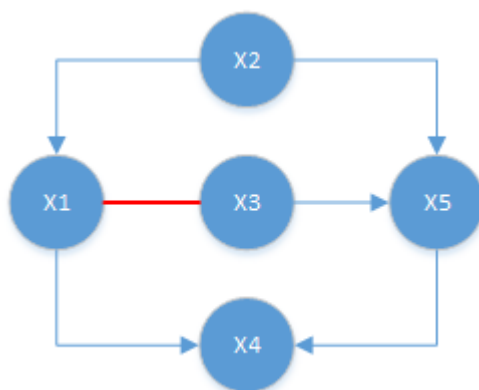


Figure 6.9 Second Example of a Graph That Is Not a DAG

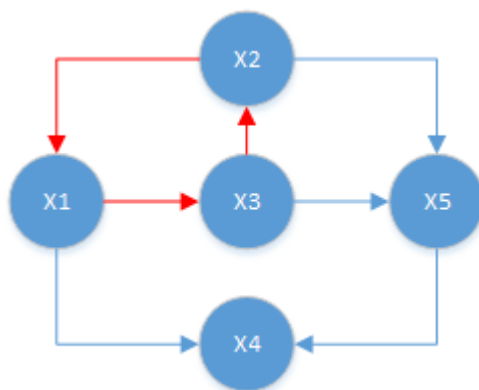
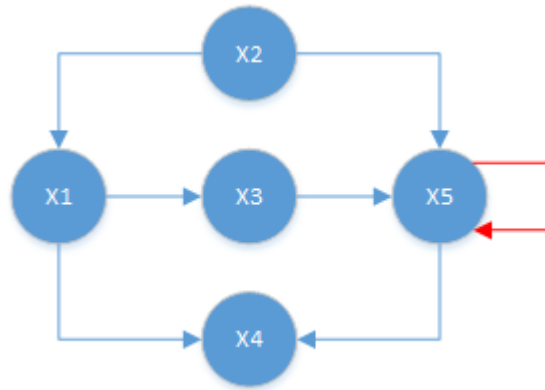


Figure 6.10 Third Example of a Graph That Is Not a DAG

If there is a directed edge from node i to node j , then node i is the parent of node j , or equivalently, node j is the child of node i . All parents of a node construct the parent set of that node. Another important concept is the topological order of a DAG, which defines the order of the nodes so that one node's parent set must be a subset of the nodes whose orders are lower than the order of that node. For example, the DAG shown in Figure 6.7 has the topological order $\{X2, X1, X3, X5, X4\}$, and $X2$ is a root variable that has no parents.

DAGs play a critical role in causal inference (Pearl 1988), which has been widely used in many fields, including machine learning (Bayesian networks), AI, economics, finance, medicine, and so on. The goal of DAG structure learning is to estimate the DAG from which the data are generated for a given data set; that is, in the DAG structure learning problem, the DAG's nodes correspond to the variables in the data set one to one, which is a known correspondence, but whether there is an edge between any two nodes (in this case, variables) and what is the direction of that edge (if it exists) should be estimated. It is well known that the DAG structure learning problem is NP-hard (Chickering, Heckerman, and Meek 2004). The searching space of DAGs is superexponential with respect to the number of variables. The exact recursive equation (Murphy 2007) to calculate the number of DAGs for a given number of variables is

$$G(d) = \sum_{k=1}^d (-1)^{(k+1)} 2^{k(d-k)} C(d, k) G(d-k)$$

where d is the number of variables and $C(d, k)$ is the number of combinations of choosing k items from d different items, where $C(d, k) = \frac{d!}{k!(d-k)!}$. The following SAS code calculates the number of DAGs for up to 20 variables. The results are shown in Output 6.11.

```

* g(n) = sum_{k=1}^n { (-1)^(k+1) * C(n, k) * 2^{k(n-k)} * g(n-k) };
%let M = 20;
proc iml;
  g = J(&M.+1, 1, 1);
  f = J(&M.+1, 1, 1);
  do n = 1 to &M.;
    f[n+1] = f[n]*n;
    g[n+1] = 0;
    do k = 1 to n;
      g[n+1] = g[n+1] +
        (-1)##(k+1) * (f[n+1]/f[k+1]/f[n-k+1]) * 2##(k*(n-k)) * g[n-k+1];
    end;
  end;
end;

```

```

dat = (1:&M.)` || g[2:&M.+1];
cols={'d' 'nDags'};
create dagcount from dat[colname=cols];
append from dat;
close;
quit;
proc print data=dagcount noobs label;
  label d="Number of Variables" nDags="Number of DAGs";
run;

```

Figure 6.11 Number of DAGs and Number of Variables

Number of Variables	Number of DAGs
1	1
2	3
3	25
4	543
5	29281
6	3781503
7	1138779265
8	783702329343
9	1.2134425E15
10	4.175099E18
11	3.1603459E22
12	5.2193965E26
13	1.8676601E31
14	1.4394281E36
15	2.3772527E41
16	8.3756671E46
17	6.2707921E52
18	9.9421195E58
19	3.327719E65
20	2.3448805E72

It is not surprising that although there are exact algorithms for DAG structure learning, those algorithms cannot be practical when you have a large number of variables. In general, there are two approaches to learning the DAG structure: one is score-based, and the other is constraint-based. The score-based approach uses a score function to measure how well a structure fits the training data and tries to find the structure that has the best score. The constraint-based approach uses independence tests to determine the edges and the directions. For more information, see Peters, Janzing, and Schölkopf (2017), Glymour, Zhang, and Spirtes (2019), Kitson et al. (2021), and the references therein.

PROC CAUSALDISCOVERY supports two types of score-based algorithms: the topological order permutation (TOP) algorithm (see Chen, Huang, and Chvosta 2022) and the minimizing conditional variance (MCV) algorithm (see Chen et al. 2024).

Topological Order Permutation (TOP) Algorithm

The topological order permutation (TOP) algorithm searches for the correct topological order first, instead of directly searching the whole DAG structure. This algorithm assumes that the full DAG with the correct topological order would lead to the best score. The full DAG (also known as a transitive tournament) is the DAG in which there is a directed edge between any two variables; that is, if there are d variables, then in the full DAG there are $d(d - 1)/2$ directed edges. Note that the topological order and the full DAG map one-to-one to each other. That is, for a given topological order, there is one and only one full DAG that corresponds to it; for a given full DAG, there is one and only one topological order that corresponds to it. Hence, sometimes the terms full DAG and topological order are used interchangeably.

For the data generating process, the DAG is linked with the structural causal model (SCM), which is described by the following equation:

$$X_i = f_i(\text{pa}(X_i), U_i), i = 1, \dots, d$$

where d is the number of variables, X_i is the i th variable, $\text{pa}(X_i)$ is the parent set of variable X_i , U_i is the error term for i th equation, $U = (U_i)$ follows some joint distribution $G(\cdot)$, and $f_i(\cdot)$ is some function for the i th equation.

In PROC CAUSALDISCOVERY, the TOP algorithm assumes that the data generating process follows the linear SCM with additive errors:

$$X_i = \sum_{X_j \in \text{pa}(X_i)} \tilde{w}_{ji} * X_j + U_i, i = 1, \dots, d$$

where the weighted adjacency matrix $\tilde{W} = (\tilde{w}_{ji})$ and $\tilde{w}_{ji}$ is nonzero if and only if X_j is the parent of X_i , and U_i is independent of any other errors. When you specify the ROOTVAR statement, those specified root variables are not approximated but serve only as parents of other variables. They appear only on the right-hand side of the equation. This additional information is used to improve the algorithm's efficiency and accuracy.

The score in the TOP algorithm for the preceding linear SCM is defined as the sum of square errors (SSRs); that is, for a given data set,

$$L = \sum_{n=1}^N \sum_{i=1}^d (X_i - \sum_{X_j \in \text{pa}(X_i)} \tilde{w}_{ji} * X_j)^2$$

where L stands for the loss and is also the score, and N is the sample size. The goal is to find the full DAG that minimizes the loss. This process can be done iteratively. Each iteration has two steps:

1. For the current topological order, evaluate the score.
2. Enumerate the neighbors of the current topological order and evaluate their scores. If none of them have a better score than the current topological order, then stop and return the current topological order as the solution. Otherwise, set the current topological order as the topological order that corresponds to the best score among the neighbors, and start the next iteration.

For a topological order $r = (r_i)_{i=1,\dots,d}$, its neighbors in the TOP algorithm are defined as the set of topological orders $\{r' : r' \in A \cup B \cup C\}$. An element $r^{(A)}$ in set A is the topological order such that there exist k_1 and k_2 , $1 \leq k_1 < k_2 \leq d$, for any i , $1 \leq i \leq d$, where the i th element of $r^{(A)}$ satisfies the following conditions:

- If $r_i < k_1$ or $r_i > k_2$, then $r_i^{(A)} = r_i$.
- If $r_i = k_2$, then $r_i^{(A)} = k_1$.
- If $k_1 \leq r_i < k_2$, then $r_i^{(A)} = r_i + 1$.

An element $r^{(B)}$ in set B is the topological order such that there exist k_1 and k_2 , $1 \leq k_1 < k_2 \leq d$, for any i , $1 \leq i \leq d$, where the i th element of $r^{(B)}$ satisfies the following conditions:

- If $r_i < k_1$ or $r_i > k_2$, then $r_i^{(B)} = r_i$.
- If $r_i = k_1$, then $r_i^{(B)} = k_2$.
- If $k_1 < r_i \leq k_2$, then $r_i^{(B)} = r_i - 1$.

An element $r^{(C)}$ in set C is the topological order such that there exist k_1 and k_2 , $1 \leq k_1 < k_2 \leq d$, for any i , $1 \leq i \leq d$, where the i th element of $r^{(C)}$ satisfies the following conditions:

- If $r_i \neq k_1$ and $r_i \neq k_2$, then $r_i^{(C)} = r_i$.
- If $r_i = k_1$, then $r_i^{(C)} = k_2$.
- If $r_i = k_2$, then $r_i^{(C)} = k_1$.

There are $d(d-1)/2$ elements in each of the sets A , B , and C , because there are $d(d-1)/2$ unique pairs of (k_1, k_2) such that $1 \leq k_1 < k_2 \leq d$. However, some element might appear in both sets or all three sets. The neighbors in the algorithm define the searching space in an iteration, which is critical for the TOP algorithm to work well and also distinguishes the TOP algorithm from all other structure learning algorithms.

After the best topological order (and equivalently, the full DAG) is found, the post-processing step is performed to delete the unnecessary edges in the full DAG. This can be regarded as a variable selection problem: for each variable X_i , $i = 1, \dots, d$, assuming that $\text{pa}_{\text{full}}(X_i)$ denotes the parent set of X_i by the full DAG, select $\text{pa}(X_i) \subseteq \text{pa}_{\text{full}}(X_i)$ according to some criterion. For example, compare the p -value of the estimated w_{ji} to a given significance level; that is, X_j is the parent of X_i if and only if the p -value is smaller than the given significance level.

As shown in “[Example 6.1: Simulation Study of DAG Structure Learning](#)” on page 159, the TOP algorithm is fast and accurate even when the number of variables reaches 100.

Minimizing Conditional Variance (MCV) Algorithm

The minimizing conditional variance (MCV) algorithm assumes that the data generating process follows the linear structural causal model with additive errors. Like the TOP algorithm, it uses the sum of square errors as the score function. It further assumes that the topological order $r = (r_1, \dots, r_d)$ corresponds to the best score if $\forall i \in [1, \dots, d], \forall j > i, \text{Var}(X_{r_i} | X_{r_1}, \dots, X_{r_{i-1}}) \leq \text{Var}(X_{r_j} | X_{r_1}, \dots, X_{r_{i-1}})$.

The MCV algorithm consists of two steps:

1. Determine a topological order of the DAG.
2. Find a correct set of edges between the nodes according to the topological order.

The algorithm determines the topological order as follows:

1. Initialize the topological order r .
2. Iteratively use the minimum conditional variance assumption to update the parent list, and use the updated parent list to compute conditional variances.

After the topological order is determined, the postprocessing step is performed as in the TOP algorithm.

The MCV algorithm is as accurate as the TOP algorithm and is much faster, especially when the number of variables is large. As shown in “[Example 6.2: Learning a Large DAG by Using the MCV Algorithm](#)” on page 163, the MCV algorithm can accurately estimate a DAG that has 1,000 nodes in just a few seconds on one CPU.

Causal Discovery with Panel Data

Broadly, PROC CAUSALDISCOVERY works by stepping through all the possible DAGs for a given set of variables, calculating the sum of the squares of the residuals through OLS regression to find the minimum, and removing the insignificant edges to output the final DAG.

When the input data are a panel data set, the procedure supports one-way or two-way fixed effects that it estimates through dummy-variable OLS regression. Fixed-effects dummies represent unobserved unit-level or time-level heterogeneity and are assumed to be exogenous by construction; they have no causal parents within the model. Such variables are therefore designated as root variables and are excluded from the final DAG output.

Missing Values

All observations are read, but only observations without missing values are used in the DAG estimation.

Data Table Output

PROC CAUSALDISCOVERY can create the output data tables listed in Table 6.2. The information in each table is described in the following sections.

OUT= Data Table Generated by the EVALUATE Statement

The output data table contains the following variables:

alpha	values of the significance levels on whose basis the edges in the estimated DAG are determined
nee	number of extra edges
nme	number of missing edges
nre	number of reversed edges
shd	structural Hamming distance, defined as the total number of edges that need to be added, deleted, or reversed to transform the estimated DAG into the true DAG
tpv	true positive rate, defined as the ratio between the number of edges that are correctly identified in the estimated DAG and the total number of edges in the true DAG. It is set to a missing value if the true DAG has no edges.

OUT= Data Table Generated by the LEARN Statement

The DAG estimation details table contains the following variables:

outType	output type. The values are R_I, R_F, EST, STD, TVL, and PVL, where R_I and R_F are, respectively, the initial and final topological order, and EST, STD, TVL, and PVL are, respectively, the estimation value, standard error, t value, and p -value of weights.
rowId	index of the variables
VarName_ i	column for the i th variable, $i = 1, \dots, d$, where d is the number of variables in the VAR statement

OUTDAG= Data Table Generated by the LEARN Statement

The estimated DAG table contains the following variables:

rowId	index of the variables
alpha	values of the significance levels on whose basis the edges in the estimated DAG are determined
VarName_ i	column for the i th variable, $i = 1, \dots, d$, where d is the number of variables in the VAR statement

OUTITER= Data Table Generated by the LEARN Statement

The iteration history table contains the following variables:

iterId	index of the iteration
loss	loss value

OUTPATH= Data Table Generated by the LEARN Statement

The paths of estimated DAGs table contains the following variables:

outType	output type. The values are ADJ and DAG, where ADJ indicates that no significance level has been applied to the weight matrix, and DAG indicates that the significance level α , whose value is listed in the α column, has been used.
rowId	index of the variables
α	values of the significance levels on whose basis the edges in the estimated DAG are determined
variable	name of the variables
parents	parents list
children	children list
pathlhsp	parent at LHS path. For example, if X_1 is the parent of X_3 and X_5 , then the path is $X_1 \Rightarrow X_3 X_5$
pathlhsc	child at LHS path. For example, if X_1 is the child of X_2 and X_4 , then the path is $X_1 \Leftarrow X_2 X_4$

OUT= Data Table Generated by the SIMULATE Statement

The simulated DAG output table contains the following variables:

rowId	index of the variables
VarName_ i	column for the i th variable, $i = 1, \dots, d$, where d is the number of variables in the VAR statement

OUTDATA= Data Table Generated by the SIMULATE Statement

The simulated data output table contains the following variables:

VarName_ i	column for the i th variable, $i = 1, \dots, d$, where d is the number of variables in the VAR statement
--------------	---

OUTORDER= Data Table Generated by the SIMULATE Statement

The topological order output table contains the following variables:

VarName_ i column for the i th variable, $i = 1, \dots, d$, where d is the number of variables in the VAR statement

OUTPARAM= Data Table Generated by the SIMULATE Statement

The parameter values output table contains the following variables:

rowId index of the variables

VarName_ i column for the i th variable, $i = 1, \dots, d$, where d is the number of variables in the VAR statement

ODS Table Names

The CAUSALDISCOVERY procedure assigns a name to each table that it creates. You can use this name to refer to the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in [Table 6.3](#).

Table 6.3 ODS Tables Produced by the CAUSALDISCOVERY Procedure

ODS Table Name	Description	Option
AlgoInfo	Algorithm information	Default
EstAdjMatrices	Estimated adjacency matrices based on significance levels	Default
EstFullAdjMat	Estimated full adjacency matrix	Default
NObs	Observation information	Default
TopoOrders	Initial and final topological orders	Default

Examples: CAUSALDISCOVERY Procedure

Example 6.1: Simulation Study of DAG Structure Learning

This example shows how to use the CAUSALDISCOVERY procedure to study the performance of the TOP algorithm and focuses on DAGs that contain 100 variables. The number of 100-variable DAGs is huge, and far beyond the maximum number that a double-precision number can express. For a data set that contains 100 variables, finding the DAG that is used in the data generating process (DGP) is an almost impossible challenge.

This example checks the accuracy of the TOP algorithm in three cases of different number of edges:

- Sparse: 100-variable, 600-edge DAGs
- Medium: 100-variable, 2,400-edge DAGs
- Full: 100-variable, 4,950-edge DAGs

For each case, 30 trials are examined. In each trial, first the SIMULATE statement simulates a DAG and then generates a 10,000-observation data set based on the simulated DAG; then the LEARN statement estimates the DAG according to the simulated data set; and finally, the EVALUATE statement evaluates the accuracy by comparing the estimated DAG from the LEARN statement to the true simulated DAG from the SIMULATE statement. In the data set simulation, a parameter value that is associated with an edge is uniformly drawn from $(-0.75, -0.4) \cup (0.4, 0.75)$. This specific range is chosen to avoid the numerical issues for the simulated data set, because when the number of edges increases, especially in the full DAG case, the chance of getting a numerically unstable data set also increases. Here, the numerically unstable data set means that the data set that is expressed as a matrix $X = (x_{i,n})_{n=1,\dots,10000}^{i=1,\dots,100}$ is not numerically full rank, although in theory it should be full rank. In the accuracy evaluation, the counts of three possible mistakes in DAG estimation—namely, missing edges, extra edges, and reverse edges—are reported; the sum of these three counts, the so-called structural Hamming distance (SHD), is also reported; and the final reported criterion is the true positive rate (TPR) for checking how many truly existing directed edges are found.

The SAS macro for a trial is as follows:

```
%macro dagTrial(simId,nNodes,nEdges,nObs,tblEval);
  * simulate DAG and data;
  proc causaldiscovery;
    var coll - col&nNodes.;
    simulate outdata=mylib.sdata out=mylib.sdag / nEdges=&nEdges. nObs=&nObs.
      parm(dist=uniform(lb=-0.75 ub=0.75 excllb=-0.4 exclub=0.4))
      seed=%eval(12345678+&simId.);
  run;

  * learn the DAG structure;
  proc causaldiscovery;
    var coll - col&nNodes.;
    learn data=mylib.sdata outdag=mylib.outdag / maxiter=%eval(5*&nNodes)
      alpha=(0.1 0.01 0.0001 0.000001 0.00000001);
  run;
```

```

        out=mylib.oest ;
run;

* evaluate the accuracy of structure learning;
proc causaldiscovery;
    var coll - col&nNodes.;
    evaluate estdag=mylib.outdag truedag=mylib.sdag / out=mylib.oeval;
run;

data mylib.oeval;
    set mylib.oeval; simID=&simId.;
run;

%if &simId.=1 %then %do;
    %let dataread=mylib.oeval;
%end;
%else %do;
    %let dataread= mylib.&tblEval. mylib.oeval;
%end;

data mylib.&tblEval.;
    set &dataread;
run;
%mend;

```

Alternatively, you can specify all three statements (SIMULATE, LEARN, and EVALUATE) in a single procedure call as follows:

```

%macro dagTrial(simId,nNodes,nEdges,nObs,tblEval);

proc causaldiscovery;
    var coll - col&nNodes.;
    * simulate DAG and data;
    simulate out=mylib.sdag outdata=mylib.sdata /
        nEdges=&nEdges. nObs=&nObs. seed=%eval(12345678+&simId.)
        parm(dist=uniform(lb=-0.75 ub=0.75 excllb=-0.4 exclub=0.4));

    * learn the DAG structure;
    learn outdag=mylib.outdag / maxiter=%eval(5*&nNodes)
        alpha=(0.1 0.01 0.0001 0.000001 0.00000001)
        out=mylib.oest ;

    * evaluate the accuracy of structure learning;
    evaluate / out=mylib.oeval;
run;

data mylib.oeval;
    set mylib.oeval; simID=&simId.;
run;

%if &simId.=1 %then %do;
    %let dataread=mylib.oeval;
%end;
%else %do;

```

```

    %let dataread= mylib.&tblEval. mylib.oeval;
%end;

data mylib.&tblEval.;
    set &dataread;
run;
%mend;

```

The following SAS macros loop over all trials and print the final accuracy report:

```

%macro dagPrint(tblEval);
    data &tblEval.; set mylib.&tblEval.; run;
    proc sort data=&tblEval.; by alpha simId; run;
    proc univariate data=&tblEval. NOPRINT;
        var shd nme nee nre tpr; by alpha;
        output out=ostat
            mean=shdMean nmeMean neeMean nreMean tprMean
            std=shdStd nmeStd neeStd nreStd tprStd;
    run;
    data pstat;
        set ostat;
        pType = 'Mean'; shd = shdMean; nme = nmeMean;
        nee = neeMean; nre = nreMean; tpr = tprMean;
        output;
        pType = 'Std.'; shd = shdStd; nme = nmeStd;
        nee = neeStd; nre = nreStd; tpr = tprStd;
        output;
        keep alpha pType shd nme nee nre tpr;
    run;
    proc print data=pstat noobs label;
        format alpha f10.8;
        format shd nme nee nre tpr f7.2;
        label alpha='Significance Level'
            pType='Type' shd='Structural Hamming Distance'
            nme='N Missing Edges' nee='N Extra Edges'
            nre='N Reverse Edges' tpr='True Positive Rate';
    run;
%mend;

%macro dagTrials(sTrial,eTrial,nNodes,nEdges,nObs,tblEval);
    %do simId = &sTrial. %to &eTrial.;
        %put "Simulation &simId.";
        %dagTrial(&simId.,&nNodes.,&nEdges.,&nObs.,&tblEval.);
    %end;
    %dagPrint(&tblEval.);
%mend;

```

First, the 100-variable, 600-edge DAG case is checked using the following code:

```

%let nNodes = 100;
%let nEdges = 600;
%let nObs = 10000;
%let tblEval = evalSparse;
%dagTrials(1,30,&nNodes.,&nEdges.,&nObs.,&tblEval.);

```

The results are shown in [Output 6.1.1](#). Because the proper significance level is chosen (in this case, 10^{-6} or 10^{-8}), the SHD decreases to zero for all 30 trials, which means that the true 100-variable DAGs in all 30 trials are perfectly discovered and the TOP algorithm reaches 100% accuracy. Such results are not seen in any algorithms that are studied by Yu et al. (2021), who also perform the experiments on the 100-variable, 600-edge DAGs.

Output 6.1.1 Accuracy for 100-Variable, 600-Edge DAG Structure Learning

Significance Level	Type	Structural Hamming Distance	N Missing Edges	N Extra Edges	N Reverse Edges	True Positive Rate
0.00000001	Mean	0.00	0.00	0.00	0.00	1.00
0.00000001	Std.	0.00	0.00	0.00	0.00	0.00
0.00000100	Mean	0.00	0.00	0.00	0.00	1.00
0.00000100	Std.	0.00	0.00	0.00	0.00	0.00
0.00010000	Mean	0.37	0.00	0.37	0.00	1.00
0.00010000	Std.	0.49	0.00	0.49	0.00	0.00
0.01000000	Mean	43.60	0.00	43.60	0.00	1.00
0.01000000	Std.	6.09	0.00	6.09	0.00	0.00
0.10000000	Mean	439.10	0.00	439.10	0.00	1.00
0.10000000	Std.	24.65	0.00	24.65	0.00	0.00
1.00000000	Mean	4350.00	0.00	4350.00	0.00	1.00
1.00000000	Std.	0.00	0.00	0.00	0.00	0.00

Second, the number of edges is increased to 2,400. The SAS code is as follows:

```
%let nNodes = 100;
%let nEdges = 2400;
%let nObs = 10000;
%let tblEval = evalMedium;
%dagTrials(1,30,&nNodes.,&nEdges.,&nObs.,&tblEval.);
```

The results are shown in [Output 6.1.2](#). Surprisingly, because the significance level is set to 10^{-6} or 10^{-8} , the accuracy of the TOP algorithm reaches 100% again.

Output 6.1.2 Accuracy for 100-Variable, 2,400-Edge DAG Structure Learning

Significance Level	Type	Structural Hamming Distance	N Missing Edges	N Extra Edges	N Reverse Edges	True Positive Rate
0.00000001	Mean	0.00	0.00	0.00	0.00	1.00
0.00000001	Std.	0.00	0.00	0.00	0.00	0.00
0.00000100	Mean	0.00	0.00	0.00	0.00	1.00
0.00000100	Std.	0.00	0.00	0.00	0.00	0.00
0.00010000	Mean	0.13	0.00	0.13	0.00	1.00
0.00010000	Std.	0.35	0.00	0.35	0.00	0.00
0.01000000	Mean	25.87	0.00	25.87	0.00	1.00
0.01000000	Std.	4.83	0.00	4.83	0.00	0.00
0.10000000	Mean	252.60	0.00	252.60	0.00	1.00
0.10000000	Std.	16.49	0.00	16.49	0.00	0.00
1.00000000	Mean	2550.00	0.00	2550.00	0.00	1.00
1.00000000	Std.	0.00	0.00	0.00	0.00	0.00

Finally, the largest possible number of edges, 4,950 edges, for a 100-variable DAG is tried. The SAS code is as follows:

```
%let nNodes = 100;
%let nEdges = 4950;
%let nObs = 10000;
%let tblEval = evalFull;
%dagTrials(1,30,&nNodes.,&nEdges.,&nObs.,&tblEval.);
```

The results are shown in [Output 6.1.3](#). Because the significance level is chosen as 0.10 or 0.01, the TOP algorithm reaches 100% accuracy again. In this extreme case, even if other significance levels are chosen, the accuracy is still very high.

Output 6.1.3 Accuracy for 100-Variable, 4,950-Edge DAG Structure Learning

Significance Level	Type	Structural Hamming Distance	N Missing Edges	N Extra Edges	N Reverse Edges	True Positive Rate
0.00000001	Mean	1.60	1.60	0.00	0.00	1.00
0.00000001	Std.	1.45	1.45	0.00	0.00	0.00
0.00000100	Mean	0.27	0.27	0.00	0.00	1.00
0.00000100	Std.	0.58	0.58	0.00	0.00	0.00
0.00010000	Mean	0.07	0.07	0.00	0.00	1.00
0.00010000	Std.	0.25	0.25	0.00	0.00	0.00
0.01000000	Mean	0.00	0.00	0.00	0.00	1.00
0.01000000	Std.	0.00	0.00	0.00	0.00	0.00
0.10000000	Mean	0.00	0.00	0.00	0.00	1.00
0.10000000	Std.	0.00	0.00	0.00	0.00	0.00
1.00000000	Mean	0.00	0.00	0.00	0.00	1.00
1.00000000	Std.	0.00	0.00	0.00	0.00	0.00

In all three cases, the TPR is always 100%, which means that in all 90 trials, the true directed edges are always found. When you run the code in this example on a 2,160-CPU grid, each trial takes only a few seconds. Note that the TOP algorithm is not an exact algorithm. Although in this example the TOP algorithm always reaches 100% accuracy for some chosen significance levels, it is not guaranteed that the TOP algorithm can reach 100% accuracy for other data sets. However, this example does show how fast, and especially how accurate, the TOP algorithm is.

Example 6.2: Learning a Large DAG by Using the MCV Algorithm

This example illustrates the accuracy and speed of the MCV algorithm, which is available in the CAUSAL-DISCOVERY procedure, by using a DAG that contains 1,000 variables and 6,000 edges.

In the following SAS code, the SIMULATE statement first simulates the DAG and then generates a 10,000-observation data set based on the simulated DAG. Then the LEARN statement estimates the DAG according to the simulated data set by using the MCV algorithm. Finally, the EVALUATE statement evaluates the accuracy of the estimated DAG.

```

proc causaldiscovery ;
  var x1 - x1000;
  simulate out=mylib.sdag outdata=mylib.outdata / nEdges=6000 nObs=10000
    parm(dist=uniform(lb=-0.75 ub=0.75 excllb=-0.4 exclub=0.4)) seed=123
    outparm = mylib.outparm outorder=mylib.outorder;

  learn outdag=mylib.outdag /
    algorithm=mcv initmethod=data
    alpha=(0.1 0.01 0.0001 0.000001 0.00000001);

  evaluate /out=mylib.oeval;
run;

```

The algorithm information is shown in [Output 6.2.1](#).

Output 6.2.1 Algorithm Information
The CAUSALDISCOVERY Procedure

Algorithm Information	
Algorithm	Minimizing Conditional Variance
Initialization Method	Data
Maximum Iterations	200
Parallel	Yes
Seed	1

The following statements print the accuracy metrics, which are shown in [Output 6.2.2](#):

```

proc print data=mylib.oeval noobs label;
  format alpha f10.8;
  format shd nme nee nre tpr f7.2;
  label alpha='Significance Level'
    shd='Structural Hamming Distance'
    nme='N Missing Edges' nee='N Extra Edges'
    nre='N Reverse Edges' tpr='True Positive Rate';
run;

```

Output 6.2.2 Accuracy of 1,000-Variable, 6,000-Edge DAG Structure Learning

Significance Level	Structural Hamming Distance	N Missing Edges	N Extra Edges	N Reverse Edges	True Positive Rate
0.00000001	0.00	0.00	0.00	0.00	1.00
0.00000100	1.00	0.00	1.00	0.00	1.00
0.00010000	43.00	0.00	43.00	0.00	1.00
0.01000000	5040.00	0.00	5040.00	0.00	1.00
0.10000000	49217.0	0.00	49217.0	0.00	1.00
1.00000000	493500	0.00	493500	0.00	1.00

The MCV algorithm has a 100% accuracy for a significance level of 10^{-8} , and the true positive rate is 100%. The algorithm is very fast, taking only a few seconds when you run the code in this example on one CPU.

Example 6.3: Using Causal Discovery on Panel Data with Fixed Effects

This example illustrates how to use the CAUSALDISCOVERY procedure to learn and evaluate a DAG that is generated from panel data that have individual fixed effects. The covariates of the data are denoted as x , and the group dummies are denoted as gn . The structural Hamming distance (SHD) and true positive rate (TPR) are compared in two cases:

- Case 1: using x variables as nodes of the graph
- Case 2: using x variables as nodes of the graph and gn variables as the root nodes of the graph

The first macro generates the panel data that have individual fixed effects as follows:

```
%macro generate_ds(dsname=);

    /* ngn groups, nT periods, and nNodes covariates */
    /* x(it) is generated by x(i1), ..., x(i, t-1), and gn(i) */
    /* for t=2, ..., t=nT */

    %put "Start generating data set &dsname..";
    %let seed = 54321;      /* Random seed */
    %let mean_shift = 1;    /* Increase mean */
    %let var_scale = 1;     /* Increase variance */

    data mylib.&dsname;
        call streaminit(&seed); /* Initialize RNG once */
        array x[&nNodes];      /* x1 to x&nNodes. */
        keep x1-x&nNodes gn t %do i=1 %to %eval(&ngn); gn&i %end;;

        do gn = 1 to &ngn;
            do t = 1 to &nT;

                /* Create group dummies */
                %do i = 1 %to %eval(&ngn);
                    gn&i = (gn = &i);
                %end;

                /* Generate x1 to x&nNodes. with increasing dependencies */
                do i = 1 to &nNodes;
                    /* Scaled noise and shifted mean */
                    x[i] = &var_scale * rannorm(1) + &mean_shift;

                    /* Add dependency on previous x variables */
                    if i > 1 then do;
                        do j = 1 to (i-1);
                            sign = (-1)**(j); /* Alternates: -1, 1, -1, 1, ... */
                            /* Use sqrt scaling for slower growth */
                            x[i] + (&var_scale * (0.1 * sqrt(j))) * sign * x[j];
                        end;
                    end;

                    /* Add dependency on gn with increasing strength */
```

```

            x[i] + (&var_scale * 0.05 * i) * gn;
        end;

        output;
    end;
end;
run;

%mend;

```

You determine the true DAG from the true topological order by using the IML procedure as follows:

```

%macro create_dag(dagname=, topo_order=);
    proc iml;
        p = &nNodes;
        var_id = T(1:p);

        /* Get sorted indices for topological order */
        call sortndx(idx,&topo_order.,1);
        topo_order2 = var_id[idx,];

        /* Build full W (complete DAG for this order) */
        W = j(p,p,0);

        do j=2 to p;
            do r=1 to j-1;
                W[topo_order2[r],topo_order2[j]] = 1;
            end;
        end;

        x_vars = "x1":"x&nNodes.";
        W2 = T(1:p) || j(p,1,1) || W;
        x_vars2 = "rowId" || 'alpha' || x_vars;

        create &dagname from W2[colname=x_vars2];
        append from W2;
        close &dagname;
    quit;

    data mylib.&dagname;
        set &dagname;
    run;
%mend;

```

The following macro runs PROC CAUSALDISCOVERY for each of the two preceding cases:

```

%macro causaldiscovery(dsname=);

    %put "Start running PROC CAUSALDISCOVERY with only x as nodes
    on data set &dsname..";

    /* Two cases:
    1. x variables are used as nodes
    2. x variables are used as nodes and
       group dummies are used as root nodes
    */

```



```

*/

/* Case 1: Only x variables are used as nodes in the analysis */
* learn the DAG structure;
ods output EstFullAdjMat = mylib.&dsname._PROC_EstFullAdjMat
      EstAdjMatrices = mylib.&dsname._PROC_EstAdjMatrices
      NObs = mylib.&dsname._PROC_NObs
      TopoOrders = mylib.&dsname._PROC_TopoOrders
      AlgoInfo = mylib.&dsname._PROC_AlgoInfo;
proc causaldiscovery;
  var %do i=1 %to (&nNodes); x&i %end;;
  learn data=mylib.&dsname. outdag=mylib.&dsname._PROC_outdag /
    nonparallel maxiter=200
    initmethod=data alpha=(0.1 0.01 0.0001 0.000001 0.00000001)
    outiter=mylib.&dsname._PROC_outiter out=mylib.&dsname._PROC_oest
    outpath=mylib.&dsname._PROC_opath;
run;

%put "Start running PROC CAUSALDISCOVERY with group as root nodes
      on data set &dsname..";
/* Case 2: The group dummies are used as root nodes in the analysis */
* learn the DAG structure;
ods output EstFullAdjMat = mylib.&dsname._PROC_gn_EstFullAdjMat
      EstAdjMatrices = mylib.&dsname._PROC_gn_EstAdjMatrices
      NObs = mylib.&dsname._PROC_gn_NObs
      TopoOrders = mylib.&dsname._PROC_gn_TopoOrders
      AlgoInfo = mylib.&dsname._PROC_gn_AlgoInfo;
proc causaldiscovery;
  var %do i=1 %to (&nNodes); x&i %end;;
  rootvar %do i=1 %to &ngn; gn&i %end;;
  learn data=mylib.&dsname.
    outdag=mylib.&dsname._PROC_gn_outdag /
    nonparallel maxiter=200
    initmethod=data alpha=(0.1 0.01 0.0001 0.000001 0.00000001)
    outiter=mylib.&dsname._PROC_gn_outiter
    out=mylib.&dsname._PROC_gn_oest
    outpath=mylib.&dsname._PROC_gn_opath;
run;
/* note that all the group dummies gn1-gn&ngn. are included
because there is no intercept in the model */

%put "Finished running PROC CAUSALDISCOVERY on data set &dsname..";
%mend;

```

The following macro evaluates the resulting DAGs from both cases against the true DAG:

```

%macro evaluate_dag(dsname=);

  %put "Start evaluating DAGs for data set &dsname..";

  /* Prepare DAGs for evaluations */

  /* Create the true DAG */
  %let &dsname._true_topo = T(1:&nNodes.);

```

```

/* True topological order is natural order by DGP */
%create_dag(dagname=&dsname._true_dag,
  topo_order=&&&dsname._true_topo.);

/* Extract the outdag where alpha=1 when only x variables are
used as nodes */
data mylib.&dsname._PROC_outdag_eval;
  set mylib.&dsname._PROC_outdag(where=(alpha=1));
run;

/* Extract the outdag where alpha=1 when gn are used as root nodes */
data mylib.&dsname._PROC_gn_outdag_eval;
  set mylib.&dsname._PROC_gn_outdag(where=(alpha=1));
run;

/* Extract the topological order for the x variables only, rerank it,
and store in a macro */
proc sql noprint;
  create table temp as
  select monotonic() as id, finalOrder as value
  from mylib.&dsname._PROC_gn_TopoOrders (obs=&nNodes);

  create table ranked as
  select a.value, a.id,
  (select count(*) from temp b where b.value < a.value) + 1 as rank
  from temp a;

  select rank
  into :&dsname._PROC_gn_TopoOrders separated by ' '
  from ranked
  order by id;

  drop table temp, ranked;
quit;

/* Create the estimated adjacency matrix for the x variables only
when running PROC CAUSALDISCOVERY with both x variables and group
dummies included as nodes */
%create_dag(dagname=&dsname._PROC_gn_outdag_eval,
  topo_order=T({&&&dsname._PROC_gn_TopoOrders.}));
/* note: need to transform &&&dsname._PROC_gn_TopoOrders. to a
numeric column vector*/

/* Evaluate the estimated DAG from PROC CAUSALDISCOVERY when
only x variables are used as nodes */
proc causaldiscovery;
  var %do i=1 %to (&nNodes); x&i %end;;
  evaluate estdag = mylib.&dsname._PROC_outdag_eval
    truedag = mylib.&dsname._true_dag /
    out=mylib.&dsname._PROC_eval;
run;

/* Evaluate the estimated DAG from PROC CAUSALDISCOVERY when
group dummies are used as root nodes */

```

```

proc causaldiscovery;
  var %do i=1 %to (&nNodes); x&i %end;;
  evaluate estdag = mylib.&dsname._PROC_gn_outdag_eval
    truedag = mylib.&dsname._true_dag /
    out=mylib.&dsname._PROC_gn_eval;
run;

%mend;

```

Finally, the following macro collects and prints the results:

```

%macro print_outputs(dsname=);

  ods html5 close; /* close any open */
  ods html5 path="~/results_causex3" (url=none)
    file="&dsname._results.html"
    gtitle gfootnote;

  title3 "&dsname: Evaluation of Estimated DAG: Case 1";
  proc print data=mylib.&dsname._PROC_eval noobs;
  run;

  title3 "&dsname: Evaluation of Estimated DAG: Case 2";
  proc print data=mylib.&dsname._PROC_gn_eval noobs;
  run;

  ods html5 close;

%mend;

```

Now that the macros have been defined, the body of the example program is as follows:

```

%let ngn = 10; /* number of groups */
%let nT = 1000; /* number of time periods */
%let dsname = ds1_1_a;
%let nNodes = 20; /* number of nodes */
%let max_iter = 200; /* Set the maximum iterations */

/* Suppress display (but still build ODS OUTPUT data sets) */
ods exclude all;
options nonotes nosource;

%generate_ds(dsname=&dsname.);
%causaldiscovery(dsname=&dsname.);
%evaluate_dag(dsname=&dsname.);

/* Re-enable output for final reporting */
ods exclude none;
options notes source;
%print_outputs(dsname=&dsname.);

proc printto; run;

```

The results from this program are shown in [Output 6.3.1](#). The tables has the following output columns: confidence level (alpha), structural Hamming distance (shd), number of missing edges (nme), number of

extra edges (nee), number of reversed edges (nre), and true positive rate (tpr).

In case 1, because you treat only the x variables as nodes, the estimation of the DAG shows deterioration. If you also include the group dummies in the analysis, the true DAG is found.

Output 6.3.1 Evaluation of Estimated DAG from PROC CAUSALDISCOVERY

ds1_1_a: Evaluation of Estimated DAG: Case 1

alpha	shd	nme	nee	nre	tpr
1	63	0	0	63	0.66842

ds1_1_a: Evaluation of Estimated DAG: Case 2

alpha	shd	nme	nee	nre	tpr
1	1	0	0	1	0.99474

References

- Chen, X., Huang, T., and Chvosta, J. (2022). “Directed Acyclic Graph Machine Learning System.” US Patent 202211443198.
- Chen, X., Kabisa, S. T., Frame, D., Chang, M.-C., Gu, W., Walton, G. E., and Elsheimer, D. B. (2024). “Techniques for Learning Causal Graphs.” US patent pending.
- Chickering, D. M., Heckerman, D., and Meek, C. (2004). “Large-Sample Learning of Bayesian Networks Is NP-Hard.” *Journal of Machine Learning Research* 5:1287–1330.
- Glymour, C., Zhang, K., and Spirtes, P. (2019). “Review of Causal Discovery Methods Based on Graphical Models.” *Frontiers in Genetics* 10:524. <https://doi.org/10.3389/fgene.2019.00524>.
- Kitson, N. K., Constantinou, A. C., Guo, Z., Liu, Y., and Chobtham, K. (2021). “A Survey of Bayesian Network Structure Learning.” ArXiv:2109.11415.
- Murphy, K. (2007). “How to Use the Bayes Net Toolbox.” Accessed September 29, 2021. <http://bayesnet.github.io/bnt/docs/usage.html>.
- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco: Morgan Kaufmann.
- Peters, J., Janzing, D., and Schölkopf, B. (2017). *Elements of Causal Inference: Foundations and Learning Algorithms*. Cambridge, MA: MIT Press.
- Yu, Y., Gao, T., Yin, N., and Ji, Q. (2021). “DAGs with No Curl: An Efficient DAG Structure Learning Approach.” ML Research Press.

Chapter 7

CCDM Procedure

Contents

Overview: CCDM Procedure	172
Using CAS Sessions and CAS Engine Librefs	174
Getting Started: CCDM Procedure	175
Estimating a Simple Compound Distribution Model	175
Analyzing the Effect of Parameter Estimate Uncertainty on the Compound Distribution	180
Scenario Analysis	183
Syntax: CCDM Procedure	193
Functional Summary	194
PROC CCDM Statement	196
BY Statement	206
COUNTMODEL Statement	206
DISPLAY Statement	207
DISPLAYOUT Statement	208
EXTERNALCOUNTS Statement	209
OUTPUT Statement	210
OUTSUM Statement	211
SEVERITYMODEL Statement	214
SIMULATEDSYMBOL Statement	215
Programming Statements	216
Details: CCDM Procedure	216
Specifying Scenario Data in the DATA= Data Table	216
Simulation Processes for Collective Risk Simulation Mode	217
Simulation Processes for Pure Premium and Custom Simulation Modes	225
Simulation of Adjusted Compound Distribution Sample	229
Parameter Perturbation Analysis	239
Fixed Parameter-Value Distributions	240
Descriptive Statistics	244
Input Specification	246
Output Data Tables	247
Displayed Output	250
ODS Graphics	251
Examples: CCDM Procedure	253
Example 7.1: Estimating the Probability Distribution of Insurance Payments	253
Example 7.2: Using Externally Simulated Count Data	258
Example 7.3: Scenario Analysis with Rich Regression Effects and BY Groups	268
Example 7.4: Illustrating Pure Premium and Custom Simulation Modes	278
References	284

Overview: CCDM Procedure

In many loss modeling applications, you analyze the loss events by modeling the severity (magnitude) of loss and the frequency (count) of loss separately. The primary goal of creating these models is to estimate the aggregate loss—that is, the total loss that occurs over a period of time for which the frequency model is applicable. For example, an insurance company might want to assess the expected and worst-case losses for a particular business line, such as automobile insurance, over an entire year by using the models for the number of losses in a year and the severity of each loss. A bank might want to assess the value-at-risk (VaR), a measure of the worst-case loss, for a portfolio of assets by using the frequency and severity models for each asset type.

Severity of loss and frequency of loss are random variables, so the aggregate loss is also a random variable. Instead of preparing a point estimate of the expected aggregate loss, it is better to estimate its probability distribution, because this enables you to infer various aspects of the aggregate loss, such as measures of location, scale (variability), and shape, in addition to percentiles. For example, the VaR that banks or insurance companies use to compute regulatory capital requirements is usually the estimate of the 97.5th or 99th percentile from the aggregate loss distribution.

Let N represent the frequency random variable for the number of loss events that occur in the time period of interest. Let X represent the severity random variable for the magnitude of one loss event. Then, the aggregate loss S is defined as

$$S = \sum_{j=1}^N X_j$$

The goal is to estimate the probability distribution of S . Let $F_X(x)$ denote the cumulative distribution function (CDF) of X , let $F_X^{*n}(x)$ denote the n -fold convolution of the CDF of X , and let $\Pr(N = n)$ denote the probability of seeing n losses as per the frequency distribution. The CDF of S is theoretically computable as

$$F_S(s) = \sum_{n=0}^{\infty} \Pr(N = n) \cdot F_X^{*n}(x)$$

This probability distribution model of S , characterized by the CDF $F_S(s)$, is referred to as a *compound distribution model* (CDM). The CCDM procedure computes an estimate of the CDM by using the distribution models of X and N .

Direct computation of F_S is usually a difficult task because of the need to compute the n -fold convolution. Klugman, Panjer, and Willmot (1998, Ch. 4) suggest some relatively efficient recursion and inversion methods for certain combinations of severity and frequency distributions. However, those methods assume that distributions of N and X are fixed and all X s are identically distributed. When the distributions of X and N are conditional on external factors (regressors), each set of regressor values results in a different distribution. So you must repeat the recursion and inversion methods for each combination of regressor values, and this repetition makes these methods prohibitively expensive. PROC CCDM instead estimates the compound distribution by using a Monte Carlo simulation method, which can use all available computational resources to generate a sufficiently large, representative sample of the compound distribution while accommodating the dependence of distributions of X and N on external factors. Conceptually, the simulation method works as follows:

1. Use the specified frequency model to draw a value N , which represents the number of loss events.

2. Use the specified severity model to draw N values, each of which represents the magnitude of loss for each of the N loss events.
3. Add the N severity values from step 2 to compute the aggregate loss S as

$$S = \sum_{j=1}^N X_j$$

This forms one sample point of the CDM.

Steps 1 through 3 are repeated M times, where M is a number that you specify, to obtain the representative sample of the CDM. PROC CCDM analyzes this sample to compute empirical estimates of various summary statistics of the compound distribution such as the mean, variance, skewness, and kurtosis in addition to percentiles such as the median, the 95th percentile, the 99th percentile, and so on. You can also use PROC CCDM to write the entire simulated sample to an output data table.

The simulation process gets more complicated when the frequency and severity models contain regression effects. The CDM is then conditional on the values of regressors. The simulation process essentially becomes a scenario analysis, because you need to specify the expected values of the regressors that together represent the scenario for which you want to estimate the CDM. PROC CCDM enables you to specify an input data table that contains the scenario. If you are modeling a group of entities together (such as a portfolio of multiple assets or a group of insurance policies), and each entity has a different set of characteristics, then the scenario consists of more than one observation, and each observation corresponds to a different entity. PROC CCDM enables you to specify such a group scenario in the input data table and creates a realistic simulation of loss events that each entity can generate.

PROC CCDM accepts the severity model of X as estimated by the SEVSELECT procedure, and it accepts the frequency model of N as estimated by the CNTSELECT procedure. Both the SEVSELECT and CNTSELECT procedures are part of SAS Econometrics software. Both procedures allow models of X and N to be conditional on external factors (regressors). In particular, you can model the severity distribution such that its scale parameter depends on severity regressors, and you can model the frequency distribution such that its mean depends on frequency regressors. The frequency model can also be a zero-inflated model. PROC CCDM uses the estimates of model parameters and the values of severity and frequency regressors to estimate the compound distribution model.

PROC CCDM also enables you to specify externally simulated counts. This is useful if you have an empirical frequency model or if you estimate the frequency model by using a method other than using PROC CNTSELECT. To specify an external count model, you create a data table that contains M replications of counts that are randomly drawn from your frequency model. For each of the replications, in step 1 of the simulation, PROC CCDM uses the count N that is associated with that replication in the count data table. If the severity model contains regression effects, then you can specify the scenario to analyze by specifying the severity regressor data for each of the M replications in the same count data table.

If the estimates of the parameters of your severity and frequency models have uncertainty associated with them, and they usually do, then you can use PROC CCDM to conduct parameter perturbation analysis to assess the effect of the uncertainty in parameter estimates on the estimates of CDM. If you request that P perturbed samples be generated, then the parameter set is perturbed P times, and each time PROC CCDM makes a random draw from either the univariate normal distribution of each parameter or the multivariate normal distribution over all parameters. For each of the P perturbed parameter sets, a full compound distribution sample is simulated and summarized. This process yields P estimates for each summary statistic

and percentile, which are then used to provide you with estimates of the location and variability of each summary statistic and percentile.

You can also use PROC CCDM to compute the distribution of an aggregate *adjusted* loss. For example, in insurance applications, you might want to compute the distribution of the amount paid in a particular time period after applying adjustments such as the deductible and the policy limit to each individual loss. PROC CCDM enables you to specify SAS programming statements to adjust each severity value. If X_j^d represents the d th adjusted severity value, then PROC CCDM computes S^d , the d th aggregate adjusted loss, as

$$S^d = \sum_{j=1}^N X_j^d$$

You can define and analyze as many adjusted losses as you want in one PROC CCDM step. All the analyses that PROC CCDM conducts for the aggregate unadjusted loss, including scenario analysis and parameter perturbation analysis, are also conducted for each aggregate adjusted loss. This gives you a comprehensive picture of the compound distribution model for each adjusted loss.

PROC CCDM is the next-generation version of PROC HPCDM. It requires SAS Cloud Analytic Services (CAS) in order to run. Because PROC CCDM is a next-generation high-performance analytical procedure, it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the

CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: CCDM Procedure

This section outlines the use of the CCDM procedure to fit compound distribution models. The examples are intended as a gentle introduction to some of the features of the procedure.

Estimating a Simple Compound Distribution Model

This example illustrates the simplest use of PROC CCDM. Assume that you are an insurance company that has used its historical data about the number of losses per year and the severity of each loss to determine that the Poisson distribution is the best distribution for the loss frequency and that the gamma distribution is the best distribution for the severity of each loss. Now, you want to estimate the distribution of an aggregate loss to determine the worst-case loss that your policyholders can incur in a year. In other words, you want to estimate the compound distribution of $S = \sum_{i=1}^N X_i$, where the loss frequency, N , follows the fitted Poisson distribution and the severity of each loss event, X_i , follows the fitted gamma distribution.

To illustrate, let the historical count and severity data be stored in the data sets Work.ClaimCount and Work.ClaimSev, respectively. The following two DATA steps simulate a frequency sample from a Poisson distribution and a severity sample from the gamma distribution, respectively:

```
/* Simulate data for an intercept-only Poisson count model */
data claimcount(keep=numLosses);
  call streaminit(12345);
  label numLosses='Number of Loss Events in a Year';
  Lambda = 2;
  do n = 1 to 500;
    numLosses = rand('POISSON', Lambda);
    output;
  end;
run;

/* Simulate data for a gamma severity model */
data claimsev(keep=lossValue);
  call streaminit(67890);
  label lossValue='Severity of a Loss Event';
  Theta = 1000;
  Alpha = 2;
  do n = 1 to 500;
```

```

        lossValue = quantile('Gamma', rand('UNIFORM'), Alpha, Theta);
        output;
    end;
run;

```

You can load the data sets `Work.ClaimCount` and `Work.ClaimSev` into data tables in your CAS session by using your CAS engine libref with the following DATA steps:

```

/* Load the data */
data mylib.claimcount;
    set claimcount;
run;
data mylib.claimsev;
    set claimsev;
run;

```

These statements assume that your CAS engine libref is named `mylib`, as in the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 174, but you can substitute any appropriately defined CAS engine libref.

To create model parameter estimates in a format that PROC CCDM can use, you need to use the following PROC CNTSELECT and PROC SEVSELECT steps to fit and store the parameter estimates of the frequency and severity models:

```

/* Fit an intercept-only Poisson count model and
   write estimates to an item store */
proc cntselect data=mylib.claimcount store=mylib.countStorePoisson;
    model numLosses= / dist=poisson;
run;

/* Fit severity models and write estimates to a data table */
proc sevselect data=mylib.claimsev criterion=aicc
    outest=mylib.sevest covout;
    loss lossValue;
    dist _predefined_;
run;

```

The `STORE=` option in the PROC CNTSELECT statement saves the count model information, including the parameter estimates, in the item store `mylib.CountStorePoisson`. An item store contains the model information in a binary format that cannot be modified after it is created. You can examine the contents of an item store that is created by a PROC CNTSELECT step by using the `VIEWSTORE` statement in another PROC CNTSELECT step. (For more information, see Chapter 10, “[CNTSELECT Procedure](#).”)

The `OUTEST=` option in the PROC SEVSELECT statement stores the estimates of all the fitted severity models in the data table `mylib.SevEst`. You can verify that the best severity model that the PROC SEVSELECT step chooses is the gamma distribution model.

You can now submit the following PROC CCDM step to simulate an aggregate loss sample of size 10,000 by specifying the count model’s item store in the `COUNTSTORE=` option and the severity model’s data table of estimates in the `SEVERITYEST=` option:

```

/* Simulate and estimate Poisson-gamma compound distribution model */
proc ccdm countstore=mylib.countStorePoisson severityest=mylib.sevest
    seed=13579 nreplicates=10000 plots=(edf density)
    print=(summarystatistics percentiles);
    severitymodel gamma;

```

```

output out=mylib.aggregateLossSample samplevar=aggloss;
outsum out=mylib.aggregateLossSummary mean stddev skewness kurtosis
      p01 p05 p95 p995=var pctlpts=90 97.5;
run;

```

The SEVERITYMODEL statement requests that an aggregate sample be generated by compounding only the gamma distribution and the frequency distribution. Specifying the SEED= value helps you get an identical sample each time you execute this step, provided that you use the same execution environment. In the single-machine mode of execution, the execution environment is the combination of the operating environment and the number of threads that are used for execution. In distributed computing mode, the execution environment is the combination of the operating environment, the number of worker nodes, and the number of threads that are used for execution on each worker node.

Upon completion, PROC CCDM creates the two output data tables that you specify in the OUT= options in the OUTPUT and OUTSUM statements. The data table mylib.AggregateLossSample contains 10,000 observations such that the value of the variable AggLoss in each observation represents one possible aggregate loss value that you can expect to see in one year. Together, the set of the 10,000 values of AggLoss represents one sample of compound distribution. PROC CCDM uses this sample to compute the empirical estimates of various summary statistics and percentiles of the compound distribution. The data table mylib.AggregateLossSummary contains the estimates of mean, standard deviation, skewness, and kurtosis that you specify in the OUTSUM statement. It also contains the estimates of the 1st, 5th, 90th, 95th, 97.5th, and 99.5th percentiles that you specify in the OUTSUM statement. The value-at-risk (VaR) is an aggregate loss value such that there is a very low probability that an observed aggregate loss value exceeds the VaR. One of the commonly used probability levels to define VaR is 0.005, which makes the 99.5th percentile an empirical estimate of VaR. Hence, the OUTSUM statement of this example stores the 99.5th percentile in a variable named VaR. VaR is one of the widely used measures of worst-case risk.

Some of the default output and some of the output that you have requested by specifying the PRINT= option are shown in [Figure 7.1](#).

Figure 7.1 Information, Summary Statistics, and Percentiles of the Poisson-Gamma Compound Distribution

The CCDM Procedure

Severity Model: Gamma
Count Model: Poisson

Compound Distribution Information	
Severity Model	Gamma Distribution
Count Model	Poisson Model in Item Store COUNTSTOREPOISSON

The CCDM Procedure

Severity Model: Gamma
Count Model: Poisson

Sample Summary Statistics			
Mean	4025.3	Median	3360.3
Standard Deviation	3443.7	Interquartile Range	4569.8
Variance	11859027.2	Minimum	0
Skewness	1.14279	Maximum	25106.0
Kurtosis	1.71709	Sample Size	10000

Figure 7.1 *continued*

Sample Percentiles	
Percentile	Value
1	0
5	0
25	1345.4
50	3360.3
75	5915.1
90	8691.7
95	10602.8
97.5	12294.1
99	14720.1
99.5	16055.7
Percentile Method = 5	

The “Sample Summary Statistics” table indicates that for the given parameter estimates of the Poisson frequency and gamma severity models, you can expect to see a mean aggregate loss and a median aggregate loss in a year. The “Sample Percentiles” table indicates that there is a 0.5% chance that the aggregate loss exceeds the 99.5th percentile, which is the VaR estimate, and a 2.5% chance that the aggregate loss exceeds the 97.5th percentile. These summary statistics and percentile estimates provide a quantitative picture of the compound distribution.

You can also visually analyze the compound distribution by examining the plots that PROC CCDM creates when you specify the PLOTS= option. The first plot in [Figure 7.2](#) shows the empirical distribution function (EDF), which is a nonparametric estimate of the cumulative distribution function (CDF). The second plot shows the histogram and the kernel density estimate, which are nonparametric estimates of the probability density function (PDF).

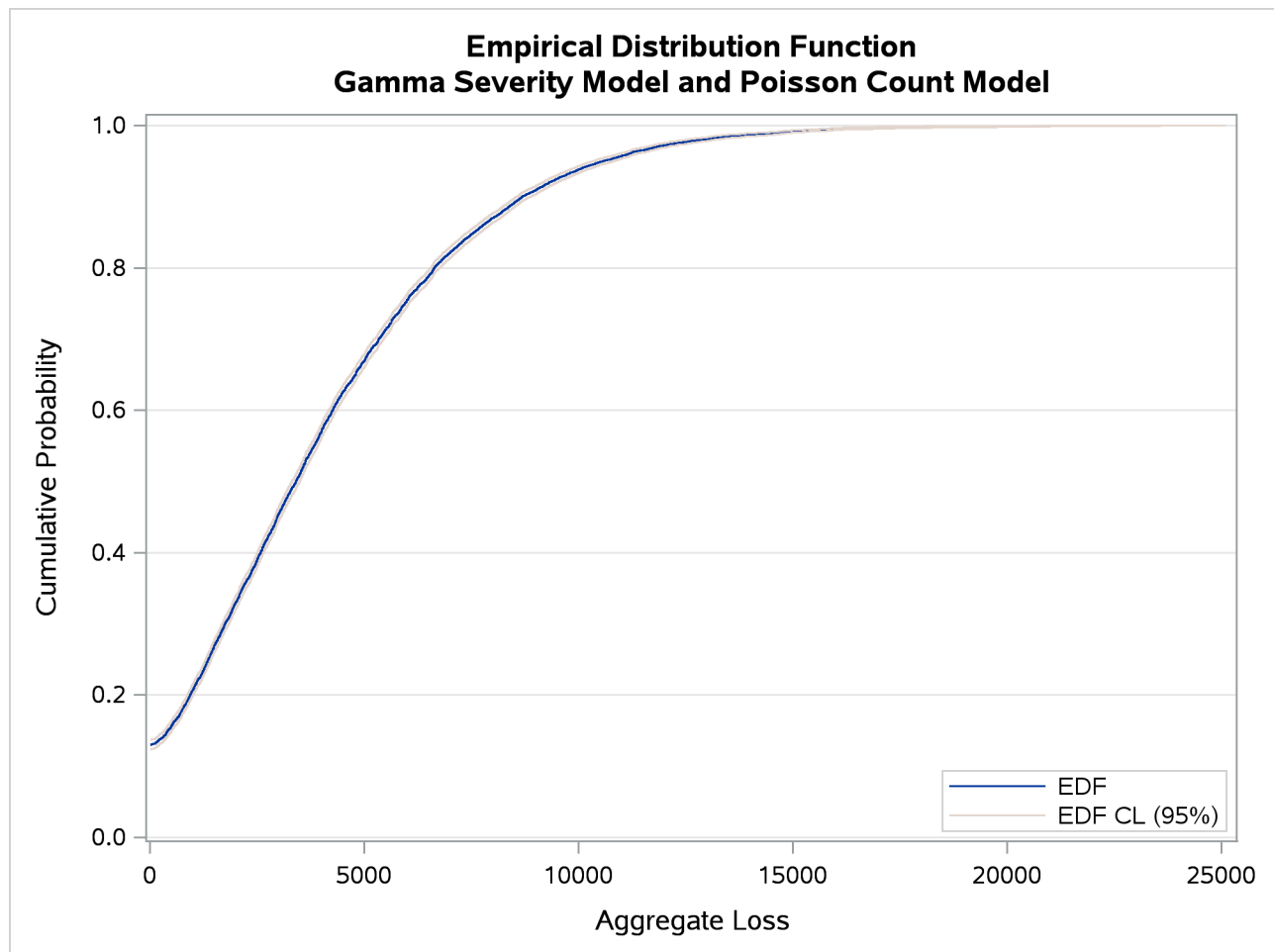
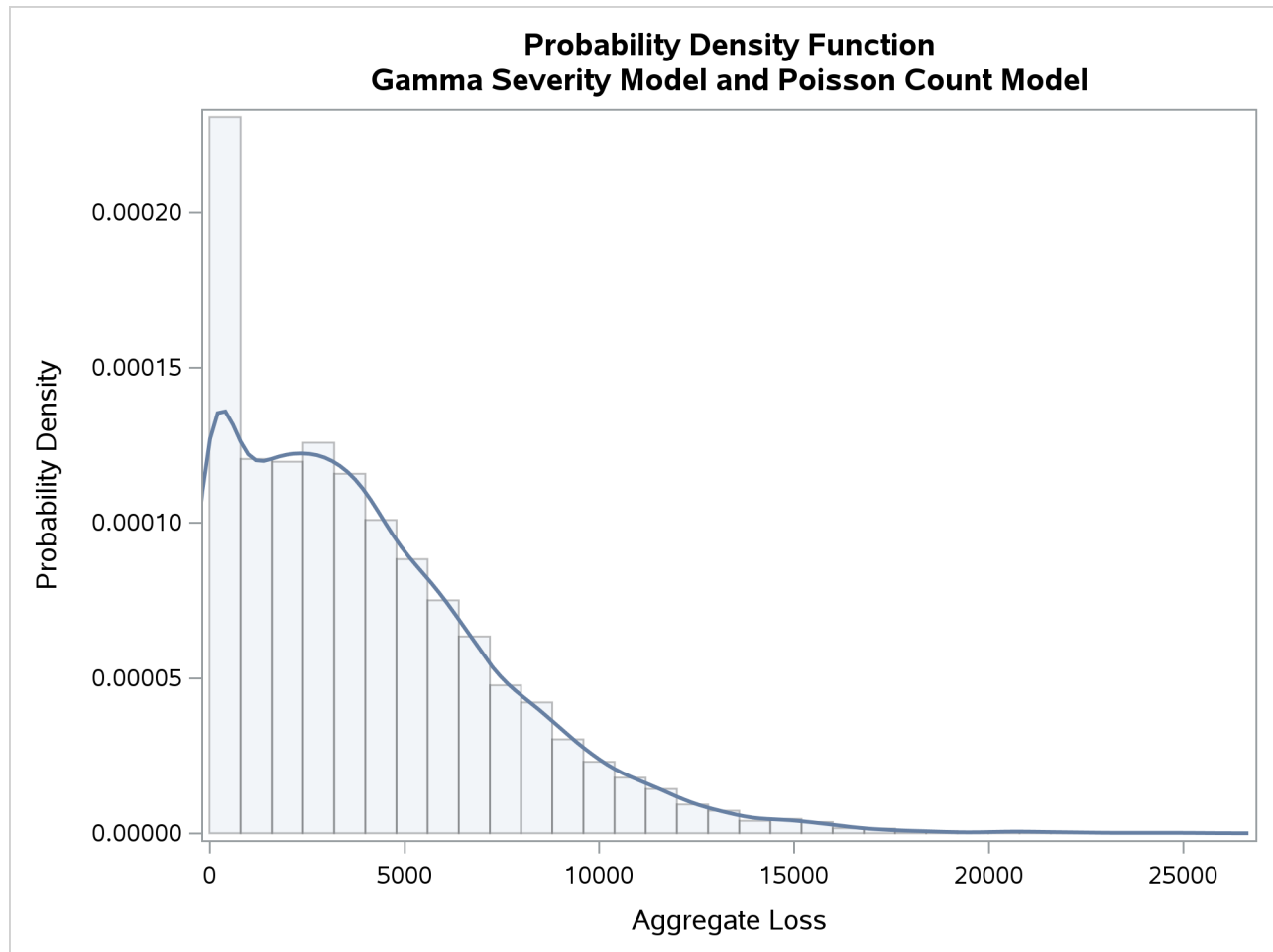
Figure 7.2 Nonparametric CDF and PDF Plots of the Poisson-Gamma Compound Distribution

Figure 7.2 *continued*

The plots confirm the right skew that is indicated by the estimate of skewness in Figure 7.1 and a relatively fat tail, which is indicated by comparing the maximum and the 99.5th percentiles in Figure 7.1.

Analyzing the Effect of Parameter Estimate Uncertainty on the Compound Distribution

Continuing with the previous example, note that you have fitted the frequency and severity models by using the historical data. Even if you choose the best-fitting models, the true underlying models are not known exactly. This fact is reflected in the uncertainty that is associated with the estimates of your model parameters. Any compound distribution estimate that you compute by using these uncertain parameter estimates is inherently uncertain. You can request that PROC CCDM perform parameter perturbation analysis; this analysis assesses the effect of the uncertainty in parameter estimates on the estimates of the compound distribution by simulating multiple samples, each of which uses parameters that are randomly perturbed from their mean estimates.

The following PROC CCDM step adds the NPerturbedSamples= option to the PROC CCDM statement to perform perturbation analysis and the PRINT=PERTURBSUMMARY option to display a summary of the perturbation analysis:

```
/* Perform parameter perturbation analysis of
   the Poisson-gamma compound distribution model */
proc ccdm countstore=mylib.countStorePoisson severityest=mylib.sevest
  seed=13579 nreplicates=10000 nperturbedsamples=30
  print (only)=(perturbsummary);
  severitymodel gamma;
  output out=mylib.aggregateLossSample samplevar=aggloss;
  outsum out=mylib.aggregateLossSummary mean stddev skewness kurtosis
    p01 p05 p95 p995=var pctlpts=90 97.5;
run;
```

The data table mylib.AggregateLossSummary contains the specified summary statistics and percentiles for all 30 perturbed samples. You can identify a perturbed sample by the value of the variable `_DRAWID_`. Figure 7.3 shows the first few observations of the data table mylib.AggregateLossSummary. For the first observation, the value of `_DRAWID_` is 0, which represents an unperturbed sample—that is, the aggregate sample that is simulated without perturbing the parameters from their means.

Figure 7.3 Summary Statistics and Percentiles of the Perturbed Samples

<code>_SEVERITYMODEL_</code>	<code>_COUNTMODEL_</code>	<code>_DRAWID_</code>	<code>_SAMPLEVAR_</code>	<code>N</code>	<code>MEAN</code>	<code>STDDEV</code>
Gamma	Poisson	0	aggloss	10000	4025.31	3443.69
Gamma	Poisson	1	aggloss	10000	4125.45	3429.26
Gamma	Poisson	2	aggloss	10000	4287.77	3637.42
Gamma	Poisson	3	aggloss	10000	4334.63	3565.39
Gamma	Poisson	4	aggloss	10000	3809.63	3265.58
Gamma	Poisson	5	aggloss	10000	4018.32	3380.26
Gamma	Poisson	6	aggloss	10000	4234.88	3533.23
Gamma	Poisson	7	aggloss	10000	4123.72	3540.83
Gamma	Poisson	8	aggloss	10000	3911.35	3353.88
Gamma	Poisson	9	aggloss	10000	4150.71	3475.63
Gamma	Poisson	10	aggloss	10000	3851.87	3411.48

<code>SKEWNESS</code>	<code>KURTOSIS</code>	<code>P01</code>	<code>P05</code>	<code>P90</code>	<code>P95</code>	<code>P97_5</code>	<code>var</code>
1.14279	1.71709	0	0	8691.72	10602.80	12294.10	16055.68
1.09793	1.47736	0	0	8801.94	10756.17	12454.72	16455.91
1.16310	1.82222	0	0	9212.21	11203.74	13220.55	17607.38
1.12635	1.62618	0	0	9132.32	11185.31	13027.37	17154.58
1.11780	1.46614	0	0	8255.58	10031.06	11854.98	15442.36
1.14690	1.67508	0	0	8621.91	10506.04	12206.77	16143.93
1.09093	1.41442	0	0	9117.79	11094.86	12727.30	16289.66
1.14779	1.59946	0	0	8891.16	10988.96	12829.11	16696.45
1.17612	1.74692	0	0	8496.19	10465.81	12206.77	15474.85
1.10561	1.63954	0	0	8892.25	10809.21	12555.65	15983.76
1.22101	1.92094	0	0	8411.36	10455.43	12313.91	16456.85

The PRINT=PERTURBSUMMARY option in the preceding PROC CCDM step produces the “Sample Perturbation Analysis” and “Sample Percentile Perturbation Analysis” tables shown in Figure 7.4. The tables

show the mean aggregate loss and the standard error of the mean. If you want to use the VaR estimate to determine the amount of reserves that you need to maintain to cover the worst-case loss, then you should consider not only the mean estimate of the 99.5th percentile, but also the standard error to account for the effect of uncertainty in your frequency and severity parameter estimates.

Figure 7.4 Summary of Perturbation Analysis of the Poisson-Gamma Compound Distribution

The CCDM Procedure		
Severity Model: Gamma		
Count Model: Poisson		
Sample Perturbation Analysis		
Statistic	Estimate	Standard Error
Mean	4062.7	165.88338
Standard Deviation	3439.7	107.78857
Variance	11843103	748676.1
Skewness	1.12773	0.04342
Kurtosis	1.58638	0.20840
Number of Perturbed Samples = 30		
Size of Each Sample = 10000		
Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	0	0
5	0	0
25	1410.1	110.09651
50	3386.8	164.53612
75	5961.2	232.68234
90	8751.3	290.64535
95	10669.9	365.97177
97.5	12439.3	416.31525
99	14656.1	530.76247
99.5	16242.1	608.33789
Number of Perturbed Samples = 30		
Size of Each Sample = 10000		

Scenario Analysis

The distributions of loss frequency and loss severity often depend on exogenous variables (regressors). For example, the number of losses and the severity of each loss that an automobile insurance policyholder incurs might depend on the characteristics of both the policyholder and the vehicle. When you fit frequency and severity models, you need to account for the effects of such regressors on the probability distributions of the counts and severity. The CNTSELECT procedure enables you to model regression effects on the mean of the count distribution, and the SEVSELECT procedure enables you to model regression effects on the scale parameter of the severity distribution. When you use these models to estimate the compound distribution model of the aggregate loss, you need to specify a set of values for all the regressors, which represents the state of the world for which the simulation is conducted. This is referred to as the what-if or scenario analysis.

Suppose that you, as an automobile insurance company, have postulated that the distribution of the loss event frequency depends on five regressors (external factors): age of the policyholder, gender, type of car, miles driven annually, and policyholder's education level. Further, the distribution of the severity of each loss depends on three regressors: type of car, safety rating of the car, and annual household income of the policyholder (which can be thought of as a proxy for the luxury level of the car). Note that the frequency model regressors and severity model regressors can be different, as illustrated in this example.

Let these regressors be recorded, respectively, in the variables Age (scaled by a factor of 1/50), Gender (1: female, 2: male), CarType (1: sedan, 2: sport utility vehicle), AnnualMiles (scaled by a factor of 1/5,000), Education (1: high school graduate, 2: college graduate, 3: advanced degree holder), CarSafety (scaled to be between 0 and 1, the safest being 1), and Income (scaled by a factor of 1/100,000).

To illustrate, the following DATA steps simulate and record the historical data about the number of losses that various policyholders incur in a year in the variable NumLoss of the data table mylib.LossCounts, and the severity of each loss in the variable LossAmount of the data table mylib.Losses:

```
/* Simulate data for losses that several policyholders incur in a year */
data losses(keep=policyholderId age gender carType annualMiles
             education carSafety income nolos lossamount);
  call streaminit(12345);
  array cx{5} age gender carType annualMiles education;
  array cbeta{6} _TEMPORARY_ (1 -0.75 1 0.6 -1 -0.25);

  array sx{3} carType carSafety income;
  array sbeta{4} _TEMPORARY_ (3.5 1.5 -0.8 0.6);

  alpha = 1/3; theta = 1/alpha;
  Sigma = 1;
  do policyholderId=1 to 5000;
    /* simulate policyholder and vehicle attributes */
    age = MAX(int(rand('NORMAL', 35, 15)),16)/50;

    if (rand('UNIFORM') < 0.5) then gender = 1; * female;
    else gender = 2; * male;

    if (rand('UNIFORM') < 0.7) then carType = 1; * sedan;
    else carType = 2; * SUV;

    annualMiles = MAX(1000, int(rand('NORMAL', 12000, 5000)))/5000;
```

```

educationLevel = rand('UNIFORM');
if (educationLevel < 0.5) then education = 1; *high school graduate;
else if (educationLevel < 0.85) then education = 2; *college graduate;
else education = 3; *advanced degree;

carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */

income = MAX(15000,int(rand('NORMAL', education*30000, 50000)))/100000;

/* simulate number of losses incurred by this policyholder */
cxbeta = cbeta(1);
do i=1 to dim(cx);
    cxbeta = cxbeta + cx(i) * cbeta(i+1);
end;
Mu = exp(cxbeta);
p = theta/(Mu+theta);
numloss = rand('NEGB',p,theta);

/* simulate severity of each loss */
if (numloss > 0) then do;
    noloss = 0;
    do iloss=1 to numloss;
        Mu = sbeta(1);
        do i=1 to dim(sx);
            Mu = Mu + sx(i) * sbeta(i+1);
        end;
        lossamount = exp(Mu) * rand('LOGNORMAL')**Sigma;
        output;
    end;
end;
else do;
    noloss = 1;
    lossamount = .;
    output;
end;
end;
run;

/* Aggregate number of annual loss events for each policyholder */
data losscounts(keep=age gender carType annualMiles education numloss);
set losses;
by policyholderId;
retain numloss 0;
if (noloss ne 1) then
    numloss = numloss + 1;
if (last.policyholderId) then do;
    output;
    numloss = 0;
end;
run;

/* Load the data */
data mylib.losses;

```

```

    set losses;
run;
data mylib.losscounts;
    set losscounts;
run;

```

Note that the last two DATA steps in the previous program load the local SAS data sets Work.LossCounts and Work.Losses into the data tables mylib.LossCounts and mylib.Losses, respectively, in your CAS session that is associated with the mylib CAS engine libref.

The following PROC CNTSELECT step fits the count regression model and stores the fitted model information in the item store mylib.CountregModel:

```

/* Fit negative binomial frequency model for the number of losses */
proc cntselect data=mylib.losscounts store=mylib.countregmodel;
    model numloss = age gender carType annualMiles education / dist=negbin;
run;

```

You can examine the parameter estimates of the count model that are stored in the item store mylib.CountregModel by submitting the following statements:

```

/* Examine the parameter estimates for the model in the item store */
proc cntselect data=mylib.losscounts;
    viewstore / instore=mylib.countregmodel finalestimates;
run;

```

The “Parameter Estimates” table that is displayed by the SHOW statement is shown in [Figure 7.5](#).

Figure 7.5 Parameter Estimates of the Count Regression Model

The CNTSELECT Procedure

Model fit results restored from item store COUNTREGMODEL:

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	0.910479	0.090515	10.06	<.0001
age	1	-0.626803	0.058547	-10.71	<.0001
gender	1	1.025034	0.032099	31.93	<.0001
carType	1	0.615165	0.031153	19.75	<.0001
annualMiles	1	-1.010276	0.017512	-57.69	<.0001
education	1	-0.280246	0.021677	-12.93	<.0001
_Alpha	1	0.318403	0.020090	15.85	<.0001

The following PROC SEVSELECT step fits the severity scale regression models for all the common distributions that are predefined in the procedure:

```

/* Fit severity models for the magnitude of losses */
proc sevselect data=mylib.losses outest=mylib.sevregest print=all;
    loss lossamount;
    scalemodel carType carSafety income;
    dist _predef_;
    nloptions maxiter=100;
run;

```

The comparison of fit statistics of various scale regression models is shown in Figure 7.6. The scale regression model based on the lognormal distribution is deemed the best-fitting model according to the likelihood-based statistics, whereas the scale regression model based on the Pareto distribution is deemed the best-fitting model according to the statistics based on the empirical distribution function (EDF).

Figure 7.6 Severity Model Comparison

The SEVSELECT Procedure

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Burr	127231	127243	127243	127286	4.66462	75.26361	9.17877
Exp	128431	128439	128439	128467	3.61201	61.22673	4.17008
Gamma	128324	128334	128334	128370	4.44757	92.64608	8.25474
Igauss	127434	127444	127444	127480	3.65517	71.24503	5.95287
Logn	127062*	127072*	127072*	127107*	4.13167	71.45566	7.20527
Pareto	128166	128176	128176	128211	3.24411*	37.34514*	2.41164*
Gpd	128166	128176	128176	128211	3.24411	37.34527	2.41165
Weibull	128429	128439	128439	128475	3.65639	64.22930	4.54181

Asterisk (*) denotes the best model in the column.

Now, you are ready to analyze the distribution of the aggregate loss that can be expected from a specific policyholder. First, you need to encode and scale the policyholder's information into the appropriate regressor variables of a data table. This table is called the scenario data table. Note that you need to follow the same encoding and scaling method for the regressor variables in the scenario data table as you use for the regressor variables in the input data tables of the SEVSELECT and CNTSELECT procedures. The following DATA steps create the scenario data table `mylib.SinglePolicy`, which contains a scenario that consists of a 59-year-old male policyholder who has an advanced degree, earns 159,870, and drives a sedan that has a very high safety rating about 11,474 miles annually:

```
/* Generate the scenario data table for single policyholder */
data singlePolicy(keep=age gender carType annualMiles
                  education carSafety income);
    age = 59/50;           * age scaled by 50;
    gender = 2;           * male;
    carType = 1;          * sedan;
    annualMiles = 11474/5000; * miles scaled by 5,000;
    education = 3;        * advanced degree;
    carSafety = 0.99532;   * car safety in the range (0,1);
    income = 159870/100000; * annual income scaled by 100,000;
    output;
run;

/* Load the data */
data mylib.singlePolicy;
    set singlePolicy;
run;
```

The following PROC CCDM step analyzes the compound distribution of the aggregate loss that the policyholder in the scenario data table `mylib.SinglePolicy` incurs in a year by using the frequency model from the item store `mylib.CountregModel` and the two best severity models, lognormal and Pareto, from the data

table mylib.SevRegEst:

```
/* Simulate the aggregate loss distribution for the scenario
   with single policyholder */
proc ccdm data=mylib.singlePolicy nreplicates=10000 seed=13579 print=all
    countstore=mylib.countregmodel severityest=mylib.sevregist;
    severitymodel logn pareto;
    outsum out=mylib.onepolicysum mean stddev skew kurtosis median
    pctlpts=97.5 to 99.5 by 1;
run;
```

The results from the preceding PROC CCDM step are shown in [Figure 7.7](#).

When you use a severity scale regression model, the “Compound Distribution Information” table displays the severity scale regression effects that PROC CCDM uses. PROC CCDM detects the severity regression effects automatically by examining the metadata that PROC SEVSELECT stores in the SEVERITYEST= data table and verifies the existence of the individual regressors in the DATA= data table.

Figure 7.7 Scenario Analysis Results for One Policyholder with Lognormal Severity Model

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Compound Distribution Information	
Severity Model	Lognormal Distribution
Scale Model Effects	carSafety carType income
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Sample Summary Statistics			
Mean	208.61064	Median	0
Standard Deviation	417.46735	Interquartile Range	249.36197
Variance	174279.0	Minimum	0
Skewness	4.33531	Maximum	6395.5
Kurtosis	30.68677	Sample Size	10000

Figure 7.7 *continued*

Sample Percentiles	
Percentile	Value
1	0
5	0
25	0
50	0
75	249.36197
95	971.66336
97.5	1358.9
98.5	1675.7
99	1964.1
99.5	2396.1
Percentile Method = 5	

The “Sample Summary Statistics” and “Sample Percentiles” tables in [Figure 7.7](#) show estimates of the aggregate loss distribution for the lognormal severity model. The percentiles table shows that the distribution is highly skewed to the right; this is also confirmed by the skewness estimate. The median estimate of 0 can be interpreted in two ways. One way is to conclude that the policyholder will not incur any loss in 50% of the years during which he or she is insured. The other way is to conclude that 50% of policyholders who have the characteristics of this policyholder will not incur any loss in a given year. However, there is a 2.5% chance that the policyholder will incur a loss that exceeds the 97.5th percentile in any given year and a 0.5% chance that the policyholder will incur a loss that exceeds the 99.5th percentile in any given year.

If the aggregate loss sample is simulated by using the Pareto severity model, then the results are as shown in [Figure 7.8](#). These estimates are very close to the values that the lognormal severity model predicts.

Figure 7.8 Scenario Analysis Results for One Policyholder with Pareto Severity Model

The CCDM Procedure

Severity Model: Pareto
Count Model: NegBin(p=2)

Compound Distribution Information	
Severity Model	Pareto Distribution
Scale Model Effects	carSafety carType income
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL

The CCDM Procedure

Severity Model: Pareto
Count Model: NegBin(p=2)

Sample Summary Statistics			
Mean	206.71313	Median	0
Standard Deviation	390.78039	Interquartile Range	260.70822
Variance	152709.3	Minimum	0
Skewness	3.24907	Maximum	4625.0
Kurtosis	15.26508	Sample Size	10000

Figure 7.8 *continued*

Sample Percentiles	
Percentile	Value
1	0
5	0
25	0
50	0
75	260.70822
95	999.56196
97.5	1360.5
98.5	1604.1
99	1777.6
99.5	2195.9
Percentile Method = 5	

The scenario that you just analyzed contains only one policyholder. You can expand the scenario to include multiple policyholders. The following DATA step simulates the data table `mylib.GroupOfPolicies`, which records information about five different policyholders, as shown in [Figure 7.9](#):

```
/* Generate the scenario data table for multiple policyholders */
data groupOfPolicies(keep=policyholderId age gender carType annualMiles
                    education carSafety income);
  call streaminit(67897);

  do policyholderId=1 to 5;
    age = MAX(int(rand('NORMAL', 35, 15)),16)/50;

    if (rand('UNIFORM') < 0.5) then gender = 1; * female;
    else gender = 2; * male;

    if (rand('UNIFORM') < 0.7) then carType = 1; * sedan;
    else carType = 2; * SUV;

    annualMiles = MAX(1000, int(rand('NORMAL', 12000, 5000)))/5000;

    educationLevel = rand('UNIFORM');
    if (educationLevel < 0.5) then education = 1; *high school graduate;
    else if (educationLevel < 0.85) then education = 2; *college graduate;
    else education = 3; *advanced degree;

    carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */

    income = MAX(15000,int(rand('NORMAL', education*30000, 50000)))/100000;

    output;
  end;
run;

/* Load the data */
data mylib.groupOfPolicies;
```

```
set groupOfPolicies;
run;
```

Figure 7.9 Scenario Analysis Data for Multiple Policyholders

policyholderId	age	gender	carType	annualMiles	education	carSafety	income
1	1.18	2	1	2.2948	3	0.99532	1.59870
2	0.66	2	1	2.6718	2	0.86412	0.84459
3	0.64	2	2	1.9528	1	0.86478	0.50177
4	0.46	1	2	2.6402	2	0.27062	1.18870
5	0.62	1	1	1.7294	1	0.32830	0.37694

The following PROC CCDM step conducts a scenario analysis for the aggregate loss that is incurred by all five policyholders in the data table mylib.GroupOfPolicies together in one year:

```
/* Simulate the aggregate loss distribution for the scenario
   of multiple policyholders */
proc ccdm data=mylib.groupOfPolicies nreplicates=10000 seed=13579 print=all
    countstore=mylib.countregmodel severityest=mylib.sevregest
    plots=(conditionaldensity(rightq=0.95)) nperturbedSamples=50;
    severitymodel logn pareto;
    outsum out=mylib.multipolicysum mean stddev skew kurtosis median
    pctlpts=97.5 to 99.5 by 1;
run;
```

The preceding PROC CCDM step conducts perturbation analysis by simulating 50 perturbed samples. The perturbation summary results for the lognormal severity model are shown in [Figure 7.10](#), and the results for the Pareto severity model are shown in [Figure 7.11](#).

Figure 7.10 Perturbation Analysis of Losses from Multiple Policyholders with Lognormal Severity Model

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Compound Distribution Information	
Severity Model	Lognormal Distribution
Scale Model Effects	carSafety carType income
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL

Sample Perturbation Analysis		
Statistic	Estimate	Standard Error
Mean	5318.6	175.49858
Standard Deviation	4179.3	141.45815
Variance	17486679	1178659.7
Skewness	2.09037	0.32112
Kurtosis	10.92294	7.21369
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

Figure 7.10 *continued*

Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	199.34899	27.70726
5	745.77631	53.38547
25	2385.1	104.32994
50	4321.0	160.79384
75	7149.7	238.12755
95	13146.5	422.60230
95	13146.5	422.60230
97.5	15820.5	517.56323
98.5	17880.0	660.76788
99	19623.3	740.16275
99.5	22674.1	1032.8
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

If the severity of each loss follows the fitted Pareto distribution, then you can expect an average loss and a worst-case loss as shown in [Figure 7.11](#).

The numbers for lognormal and Pareto are well within one or two standard errors of each other, which indicates that the aggregate loss distribution is less sensitive to the choice of these two severity distributions in this particular example. You can use the results from either of them.

Figure 7.11 Perturbation Analysis of Losses from Multiple Policyholders with Pareto Severity Model

The CCDM Procedure

Severity Model: Pareto
Count Model: NegBin(p=2)

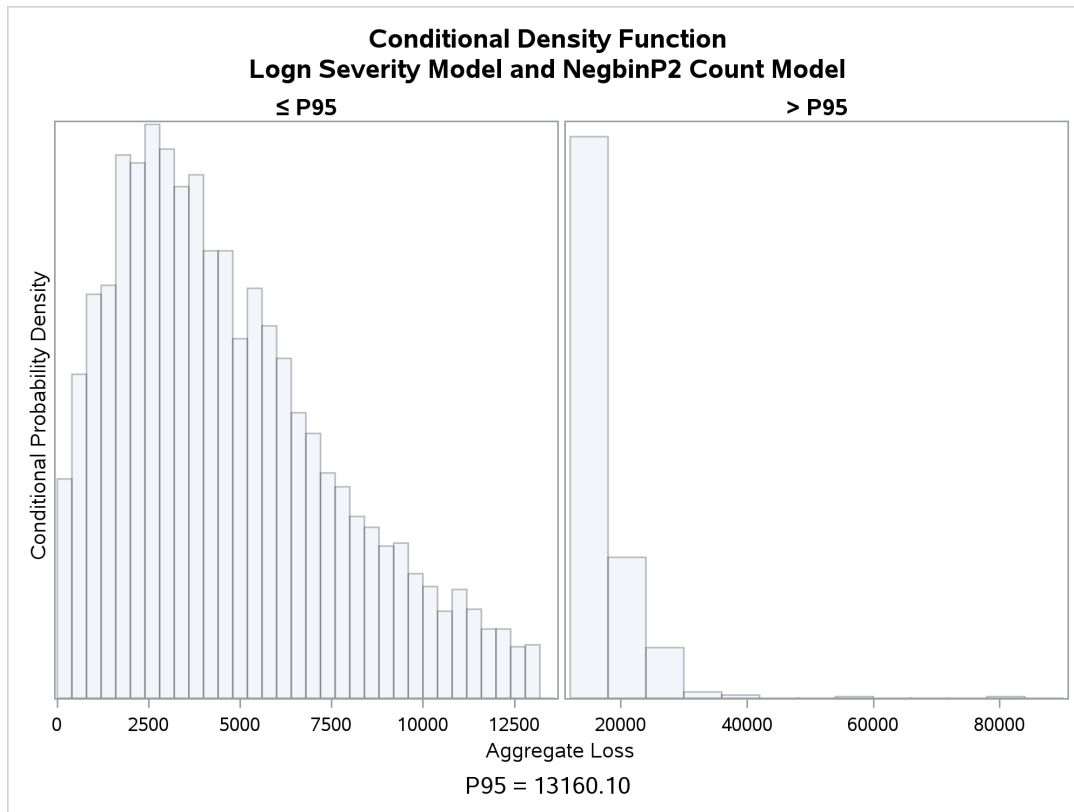
Compound Distribution Information	
Severity Model	Pareto Distribution
Scale Model Effects	carSafety carType income
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL

Sample Perturbation Analysis		
Statistic	Estimate	Standard Error
Mean	5251.4	169.35598
Standard Deviation	3915.5	121.76718
Variance	15346184	945964.3
Skewness	1.49189	0.08940
Kurtosis	3.79266	0.85099
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

Figure 7.11 *continued*

Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	162.52002	26.88218
5	700.88526	52.09737
25	2386.3	102.12837
50	4365.2	158.73203
75	7166.9	229.30068
95	12752.8	385.34531
95	12752.8	385.34531
97.5	15052.7	512.07368
98.5	16745.5	585.49172
99	18118.1	656.42766
99.5	20512.6	819.42806
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

The PLOTS=CONDITIONALDENSITY option that is used in the preceding PROC CCDM step creates the conditional density plots for the body and right-tail regions of the density function of the aggregate loss. The plots for the aggregate loss sample that is generated by using the lognormal severity model are shown in Figure 7.12. The plot on the left side is the plot of $\Pr(Y|Y \leq 95\text{th percentile})$, where the limit is as specified by the RIGHTQ=0.95 suboption of the PLOTS=CONDITIONALDENSITY option. The plot on the right side is the plot of $\Pr(Y|Y > 95\text{th percentile})$, which helps you visualize the right-tail region of the density function. You can also request the plot of the left tail by specifying the LEFTQ= suboption if you want to explore the details of the left tail region. The conditional density plots are always produced by using the unperturbed sample.

Figure 7.12 Conditional Density Plots for the Aggregate Loss of Multiple Policyholders

Syntax: CCDM Procedure

The following statements are available in the CCDM procedure:

```
PROC CCDM options ;
  BY variable-list ;
  SEVERITYMODEL severity-model-list < / definition-option > ;
  EXTERNALCOUNTS COUNT=frequency-variable < ID=replication-id-variable > ;
  COUNTMODEL distribution-name(parameter-value < (value-option) > < , > ... parameter-value
    < (value-option) > ) ;
  SIMULATEDSYMBOL symbol ~ distribution-name(parameter-value-list) < ... symbol ~ distribution-
    name(parameter-value-list) > ;
  OUTPUT OUT=libref.data-table < variable-name-options > < / out-option > ;
  OUTSUM OUT=libref.data-table      statistic-keyword=variable-name >      < ... statistic-
    keyword=variable-name > > < outsum-options > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
Programming statements ;
```

Functional Summary

Table 7.1 summarizes the statements and options available in the CCDM procedure.

Table 7.1 PROC CCDM Functional Summary

Description	Statement	Option
Statements		
Specifies the names of severity distribution models	SEVERITYMODEL	
Specifies externally simulated count data	EXTERNALCOUNTS	
Specifies a count distribution that has fixed parameter values	COUNTMODEL	
Specifies simulated symbols and their probability distributions	SIMULATEDSYMBOL	
Specifies where and how the full simulated samples are written	OUTPUT	
Specifies where and how the summary statistics of simulated samples are written	OUTSUM	
Specifies the ODS tables to display (an alternative to the PRINT= option)	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	
Specifies programming statements that define an objective function	Programming statements	
Data Table Options		
Specifies the input data table	PROC CCDM	DATA=
Specifies the output data table for the individual severity draws	PROC CCDM	OUTDRAW=
Specifies the output data table for the full simulated samples	OUTPUT	OUT=
Specifies the output data table for the summary statistics	OUTSUM	OUT=
Model Input Options		
Specifies the variable that contains externally simulated counts	EXTERNALCOUNTS	COUNT=
Specifies the item store that contains the frequency (count) model	PROC CCDM	COUNTSTORE=
Specifies the replicate identifier variable for external counts	EXTERNALCOUNTS	ID=
Specifies the input data table for parameter estimates of the severity models	PROC CCDM	SEVERITYEST=
Specifies the item store for parameter estimates of the severity models	PROC CCDM	SEVERITYSTORE=

Table 7.1 *continued*

Description	Statement	Option
Simulation Options		
Specifies the adjusted severity symbols in the programming statements	PROC CCDM	ADJUSTEDSEVERITY=
Specifies that the covariance estimates be ignored for perturbation analysis	PROC CCDM	IGNORECOV
Specifies the left-truncation threshold for truncating a severity distribution	PROC CCDM	LEFTTRUNCATION=
Specifies the upper limit on the count for each sample point	PROC CCDM	MAXCOUNTDRAW=
Specifies the number of parameter-perturbed samples to be simulated	PROC CCDM	NPERTURBEDSAMPLES=
Specifies a number that controls the size of the simulated sample	PROC CCDM	NREPLICATES=
Specifies how the perturbed samples are generated	PROC CCDM	PERTURBMODE=
Specifies the right-truncation limit for truncating a severity distribution	PROC CCDM	RIGHTTRUNCATION=
Specifies a seed for the internal pseudorandom number generator	PROC CCDM	SEED=
Specifies the type of aggregate loss you want to simulate	PROC CCDM	SIMULATIONMODE=
Output Preparation Options		
Specifies the variables for the aggregate adjusted loss samples	OUTPUT	ADJSAMPLEVAR=
Specifies that a backward compatible output style be used for the case of a single adjusted severity symbol	PROC CCDM	ONEADJSEVCOMPAT
Specifies the names of the variables for percentiles	OUTSUM	PCTLNAME=
Specifies the decimal precision to form default percentile variable names	OUTSUM	PCTLNDEC=
Specifies percentiles to compute and report	OUTSUM	PCTLPTS=
Specifies the method to compute the percentiles	PROC CCDM	PCTLDEF=
Specifies that all perturbed samples be written to the OUT= data table	OUTPUT	PERTURBOUT
Specifies the variable for the aggregate loss sample	OUTPUT	SAMPLEVAR=
Specifies that zero-valued aggregate losses be removed from the aggregate loss sample	PROC CCDM	TRUNCATEZEROS
Specifies the denominator for computing second- and higher-order moments	PROC CCDM	VARDEF=
Displayed Output Options		
Suppresses all displayed output	PROC CCDM	NOPRINT

Table 7.1 *continued*

Description	Statement	Option
Specifies which graphical output to create	PROC CCDM	PLOTS=
Specifies which displayed output to produce	PROC CCDM	PRINT=

PROC CCDM Statement

PROC CCDM *options* ;

The PROC CCDM statement invokes the procedure. You can specify the following *options*, which are listed in alphabetical order.

ADJUSTEDSEVERITY=*symbol-name*

ADJSEV=*symbol-name*

ADJUSTEDSEVERITY=(*symbol-name* <... *symbol-name* >)

ADJSEV=(*symbol-name* <... *symbol-name* >)

names one or more symbols that represent an adjusted severity value in the SAS programming statements that you specify. If you specify more than one *symbol-name*, then separate them with spaces and enclose them in parentheses. Each *symbol-name* is a SAS name that conforms to the naming conventions of a SAS variable. For more information, see the section “[Programming Statements](#)” on page 216.

COUNTSTORE=*libref.data-table*

names the input data table that contains all the information about the frequency (count) model in the form of an item store. The CNTSELECT procedure generates this item store data table when you use the STORE= option.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 174. The *libref* must be identical to the *libref* that you specify in the DATA= option.

The exogenous variables in the frequency model, if any, are deduced from this item store. The DATA= data table must contain all those variables.

If you specify a BY statement in the PROC CNTSELECT step that creates the COUNTSTORE= item store, then you must specify an identical BY statement in the PROC CCDM step.

You must specify this option if you do not specify the COUNTMODEL or EXTERNALCOUNTS statement. This option is ignored if you specify the EXTERNALCOUNTS statement, because PROC CCDM does not need to simulate frequency counts internally when you specify externally simulated counts.

DATA=libref.data-table

names the input data table for PROC CCDM to use. *libref.data-table* is a two-level name, where

<i>libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>libref</i> , see the section “Using CAS Sessions and CAS Engine Librefs” on page 174.
<i>data-table</i>	specifies the name of the input data table.

The DATA= data table specifies information about the scenario for which you want to estimate the aggregate loss distribution. It is expected to contain the values of regression variables in the frequency or severity model and severity adjustment variables that you use in the programming statements.

IGNORECOV**IGNOREPARMCOVARIANCE**

specifies that the covariance estimates be ignored for perturbation analysis. If you specify this option, then even if the covariance estimates of severity or frequency model parameters are valid, they are not used for perturbation analysis, and only standard errors are used. This enables you to force independence among parameters. However, perturbation analysis based on covariance estimates is more accurate than perturbation analysis based on the standard errors alone.

For more information about parameter perturbation analysis, see the section [“Parameter Perturbation Analysis”](#) on page 239.

LEFTTRUNCATION=number**LTRUNC=number**

specifies the left-truncation threshold for truncating a severity distribution.

Specifying this option, and optionally the **RIGHTTRUNCATION=** option, enables you to simulate from a truncated severity distribution. For more information, see the section [“Making a Random Draw from a Truncated Severity Distribution”](#) on page 225.

MAXCOUNTDRAW=number**MAXCOUNT=number**

specifies an upper limit on the number of loss events (count) that is used for simulating one aggregate loss sample point. If *number* is equal to $N_{\max}$, then any count greater than $N_{\max}$ is assumed to be equal to $N_{\max}$, and only $N_{\max}$ severity draws are made to compute one point in the aggregate loss sample.

If you specify this option and also specify the COUNTSTORE= item store or the COUNTMODEL statement, then the limit is applied to each count that PROC CCDM randomly draws from the count distribution. Any count draw larger than *number* is replaced by *number*.

If you specify this option and also specify the EXTERNALCOUNTS statement, then the limit is applied to each observation in the DATA= data table, and any value of the COUNT= variable larger than *number* is replaced by *number*.

If you do not specify this option, then a default value of 1,000 is used.

If you specify a *number* significantly larger than 1,000, then PROC CCDM might take a very long time to complete the simulation, especially when some counts are closer to the limit.

NOPRINT

turns off all displayed output. If you specify this option, then PROC CCDM ignores any value that you specify for the PRINT= option.

NPERTURBEDSAMPLES=number**NPERTURB=number**

performs parameter perturbation analysis, where *number* determines how many perturbed parameter sets are generated. A separate full sample is simulated for each set of perturbed parameter values. The summary statistics and percentiles are computed for each such perturbed sample, and their values are aggregated across the samples to compute the mean and standard deviation of each summary statistic and percentile.

The parameter perturbation analysis makes random draws of parameter values from a multivariate normal distribution if the covariance estimates of the parameters are available. For the multivariate normal distribution of severity model parameters, PROC CCDM attempts to read the covariance estimates from the SEVERITYEST= data table or the SEVERITYSTORE= item store. For the multivariate normal distribution of count model parameters, PROC CCDM attempts to read the covariance estimates from the item store that is specified in the COUNTSTORE= option. If covariance estimates are not available or valid, or if you specify the IGNORECOV option, then for each parameter, PROC CCDM makes a random draw from the univariate normal distribution whose mean and standard deviation are equal to that parameter's point estimate and standard error, respectively. The standard errors for severity model parameters are read from either the SEVERITYEST= data table or the SEVERITYSTORE= item store. The standard errors for count model parameters are either read from the COUNTSTORE= item store or you must specify them in the COUNTMODEL statement. If neither covariance nor standard error estimates are available for any severity or frequency model parameter, then perturbation analysis is not conducted.

If you specify the PRINT=ALL or PRINT=PERTURBSUMMARY option, then a summary of the perturbation analysis is printed for the core summary statistics and the percentiles of the aggregate loss distribution. If you specify the OUTSUM statement, then the requested summary statistics are written to the OUTSUM= data table for each perturbed sample. You can also optionally request that each perturbed sample be written in its entirety to the OUT= data table by specifying the PERTURBOUT option in the OUTPUT statement.

For more information about parameter perturbation analysis, see the section “[Parameter Perturbation Analysis](#)” on page 239.

NREPLICATES=number**NREP=number**

specifies a *number* that controls the size of the compound distribution sample that PROC CCDM simulates. The *number* is interpreted differently based on whether you specify the EXTERNALCOUNTS statement.

If you do not specify the EXTERNALCOUNTS statement, then the sample size is equal to *number*. If you do not specify this option, then a default value of 100,000 is used.

If you specify the EXTERNALCOUNTS statement, then the number of replicates that you specify in the DATA= data table is multiplied by *number* to get the total size of the compound distribution sample. If you do not specify this option, then a default value of 1 is used.

ONEADJSEVCOMPAT

specifies that the output of the case of single adjusted severity symbol be prepared by using the same method as used by the versions of the CCDM procedure prior to SAS Econometrics 8.5. In particular, specifying this option causes the following output-related changes, where *sym* denotes the name of the single adjusted severity symbol that you specify in the [ADJUSTEDSEVERITY=](#) option:

- If you specify the OUTPUT statement, then the default name of the adjusted severity variable in the OUTSAMPLE= table is `_AGGADJSEV_` instead of *sym*.
- If you specify the OUTSUM statement, then the default value in the `_SAMPLEVAR_` column of the OUTSUM= table is `_AGGADJSEV_` instead of *sym* for the rows that correspond to the aggregate adjusted severity.
- The ODS path for the adjusted severity group's output table or plot uses `AggregateAdjustedLoss` instead of *sym*.
- The label for the adjusted severity group in the table of contents of a SAS client is "Aggregate Adjusted Loss" instead of "Aggregate Adjusted Loss *sym*."
- The title of the SummaryStatistics ODS table is "Adjusted Sample Summary Statistics" instead of "Summary Statistics for *sym*."
- The title of the PerturbedSummary ODS table is "Adjusted Sample Perturbation Analysis" instead of "Sample Perturbation Analysis for *sym*."
- The title of the Percentiles ODS table is "Adjusted Sample Percentiles" instead of "Percentiles for *sym*."
- The title of the PerturbedPctlSummary ODS table is "Adjusted Sample Percentile Perturbation Analysis" instead of "Percentile Perturbation Analysis for *sym*."
- The titles of plots use just "Adjusted Sample" instead of "Adjusted Sample *sym*."
- The X-axis variable label in plots is just "Aggregate Adjusted Loss" instead of "Aggregate Adjusted Loss *sym*."

When you do not specify this option, the style of the output for the case of a single adjusted severity symbol is no different from the style of the output for the case of more than one adjusted severity symbol.

OUTDRAW=*libref.data-table*

specifies the output data table to contain the individual severity random draws. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section "Using CAS Sessions and CAS Engine Librefs" on page 174. The *libref* must be identical to the *libref* that you specify in the [DATA=](#) option.

For more information about the variables in the OUTDRAW= data table, see the section "[OUTDRAW= Data Table](#)" on page 247.

PCTLDEF=*percentile-method*

specifies the method of computing the percentiles of the compound distribution. The *percentile-method* can be 1, 2, 3, 4, or 5. By default, PCTLDEF=5. For more information, see the description of the PCTLDEF= option in the UNIVARIATE procedure in the [Base SAS Procedures Guide: Statistical Procedures](#).

PERTURBMODE=*mode*

PERTURBMETHOD=*mode*

specifies the method to use for generating perturbed samples.

This option has no effect if you do not specify the **NPERTURBEDSAMPLES=** option or if you specify the **EXTERNALCOUNTS** statement.

Let P and W denote the values of the **NPERTURBEDSAMPLES=** option and the number of worker nodes in the CAS session that the procedure is running in, respectively. Also, let N denote the value of the **NREPLICATES=** option.

You can specify one of the following *modes*:

AUTO | SYNC | 1

generates perturbed samples in either a distributed or partitioned manner, depending on the values of P and W , as follows:

- If $P < W$, then each perturbed sample is distributed across all W workers, such that each worker generates a subsample of approximately N/W points of each perturbed aggregate loss sample. If you specify the **PERTURBOUT** option in the **OUTPUT** statement, then observations that have the same **_DRAWID_** variable value are distributed across all worker nodes. The controller node needs to gather the subsamples from all the worker nodes in order to compute the percentiles, and perturbed samples are generated sequentially for each severity model within each BY group.
- If $P \geq W$, then perturbed samples are generated in a partitioned manner such that each worker node generates approximately P/W perturbed samples, each of size N . If you specify the **PERTURBOUT** option in the **OUTPUT** statement, then observations that have the same **_DRAWID_** variable value are available on only one of the worker nodes. There is no need to communicate any sample data to the controller node, because each worker node has enough data to compute the percentiles of each perturbed sample. The controller node gathers only the summary statistics and percentiles of perturbed samples from all worker nodes. Further, each worker node can simulate its assigned P/W perturbed samples independently of other worker nodes. Together, these two aspects make the perturbation analysis more efficient.

DIST | SYNCDIST | 2

always generates perturbed samples in a distributed manner, irrespective of the values of P and W . This option's behavior is equivalent to the behavior of the **AUTO** option in the $P < W$ case.

It is recommended that you specify the **AUTO mode**. By default, **PERTURBMETHOD=DIST** to ensure that the current release of PROC CCDM produces, by default, the same perturbation results as releases prior to SAS Econometrics 8.4.

PLOTS <(global-plot-options)> =*plot-request-option*

PLOTS <(global-plot-options)> =(*plot-request-option* . . . *plot-request-option*)

specifies the desired graphical output.

By default, the CCDM procedure produces no graphical output.

When you request plots by using this option, PROC CCDM creates a plot data table on the server. This table contains an estimate of the empirical distribution function (EDF) for a carefully selected subsample of the simulated aggregate loss sample. PROC CCDM brings the plot data table back to the client machine where your SAS session is running and uses it to create all the plots that you specify.

You can specify the following *global-plot-options*:

ONLY

turns off the default graphical output and creates only the requested plots.

NOBS=number

specifies the approximate number of observations in the plot data table. The larger the *number*, the more accurate are the plots, but also the larger the size of the data that need to be brought back to the client machine. A larger plot data table not only requires more time and resources to prepare but can also cost you additional money if the data movement on the client-server connection is metered.

If you specify a nonzero *number*, then it must be greater than 10. If you do not specify this option or if you specify NOBS=0, then a default value of 5,000 is used.

NADJSEVTOPLOT=k

NADJSEV=k

specifies the number of aggregate adjusted loss samples for which to create the requested plots. PROC CCDM creates plots for the first *k* adjusted severity symbols that you specify in the [ADJUSTEDSEVERITY=](#) option.

This option is ignored if you do not specify the [ADJUSTEDSEVERITY=](#) option and the programming statements to adjust the simulated severity values.

If you specify *m* symbols in the [ADJUSTEDSEVERITY=](#) option and if $k > m$, then PROC CCDM creates the requested plots for all adjusted severity symbols.

Note that PROC CCDM prepares a separate plot data table for each of the *k* aggregate adjusted loss samples, so the amount of data that PROC CCDM needs to bring back to the client machine increases linearly with the value of *k*.

By default, PROC CCDM does not create plots for aggregate adjusted loss samples—that is, NADJSEVTOPLOT=0 by default.

You control which plots to create for each aggregate loss sample by specifying the *plot-request-options*. If you specify more than one *plot-request-option*, then separate them with spaces and enclose them in parentheses. You can specify the following *plot-request-options*:

ALL

displays all the graphical output.

CONDITIONALDENSITY (*conditional-density-plot-options*)

CONDPDF (*conditional-density-plot-options*)

creates a group of plots of the conditional density functions estimates. The group contains at most three plots, each conditional on the value of the aggregate loss being in one of the three regions that are defined by the quantiles that you specify in the following *conditional-density-plot-options*:

LEFTQ=number

specifies the quantile in the range (0,1) that marks the end of the left-tail region. If you specify a value of l for *number*, then the left-tail region is defined as the set of values that are less than or equal to q_l , where q_l is the l th quantile. For the left-tail region, nonparametric estimates of the conditional probability density function $f_S^l(s) = \Pr[S = s | S \leq q_l]$ are plotted. The value of q_l is estimated by the 100/ l th percentile of the simulated compound distribution sample.

If you do not specify this option or you specify a missing value for this option, then the left-tail region is not plotted.

RIGHTQ=number

specifies the quantile in the range (0,1) that marks the beginning of the right-tail region. If you specify a value of r for *number*, then the right-tail region is defined as the set of values that are greater than q_r , where q_r is the r th quantile. For the right-tail region, nonparametric estimates of the conditional probability density function $f_S^r(s) = \Pr[S = s | S > q_r]$ are plotted. The value of q_r is estimated by the 100/ r th percentile of the simulated compound distribution sample.

If you do not specify this option or you specify a missing value for this option, then the right-tail region is not plotted.

You must specify a nonmissing value for at least one of the preceding two options. For the region between the LEFTQ= and RIGHTQ= quantiles, which is referred to as the central or body region, nonparametric estimates of the conditional probability density function $f_S^c(s) = \Pr[S = s | q_l < S \leq q_r]$ are plotted. If you do not specify a LEFTQ= value, then q_l is assumed to be 0. If you do not specify a RIGHTQ= value, then q_r is assumed to be ∞ .

DENSITY

creates a plot of the nonparametric estimates of the probability density function (in particular, histogram and kernel density estimates) of the compound distribution.

EDF <(edf-plot-option)>

creates a plot of the nonparametric estimates of the cumulative distribution function of the compound distribution.

You can request that the confidence interval be plotted by specifying the following *edf-plot-option*:

ALPHA=number

specifies the confidence level in the (0,1) range that is used for computing the confidence intervals for the EDF estimates. If you specify a value of α for *number*, then the upper and lower confidence limits for the confidence level of $100(1 - \alpha)$ are plotted.

NONE

displays none of the graphical output. If you specify this option, then it overrides all other plot request options. The default graphical output is also suppressed.

PRINT <(*global-display-option*)> =*display-option*

PRINT <(*global-display-option*)> =(display-option ... display-option)

specifies the desired displayed output. If you specify more than one *display-option*, then separate them with spaces and enclose them in parentheses.

You can specify the following *global-display-option*:

ONLY

turns off the default displayed output and displays only the requested output.

You can specify the following *display-options*:

ALL

displays all the output.

NONE

displays no output. If you specify this option, then it overrides all other display options. The default displayed output is also suppressed.

PERCENTILES

displays the percentiles of the compound distribution sample. This includes all the predefined percentiles and percentiles that you request in the OUTSUM statement.

PERTURBSUMMARY

displays the mean and standard deviation of the summary statistics and percentiles that are taken across all *P* samples that the procedure produces by perturbing the model parameters, where *P* is the value of the NPERTURBEDSAMPLES= option that you specify in the PROC CCDM statement.

SUMMARYSTATISTICS | SUMSTAT

displays the summary statistics of the compound distribution sample.

If you do not specify the PRINT= option or the ONLY *global-display-option*, then the procedure behaves as if you have specified PRINT=(SUMMARYSTATISTICS).

RIGHTTRUNCATION=*number*

RTRUNC=*number*

specifies the right-truncation limit for truncating a severity distribution.

Specifying this option, and optionally the [LEFTTRUNCATION](#)= option, enables you to simulate from a truncated severity distribution. For more information, see the section “[Making a Random Draw from a Truncated Severity Distribution](#)” on page 225.

SEED=*number*

specifies the integer to use as the seed in generating the pseudorandom numbers that are used for simulating severity and frequency values.

If you do not specify the seed, or if *number* is negative or 0, then PROC CCDM uses the time of day from the computer’s clock as the seed.

SEVERITYEST=libref.data-table

names the input data table that contains the parameter estimates for the severity models. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 174. The *libref* must be identical to the *libref* that you specify in the [DATA=](#) option. This data table must be an [OUTEST=](#) data table that the [SEVSELECT](#) procedure produces.

The names of the regression variables in the scale regression models, if any, are deduced from this data table. If the severity models are scale regression models, then the [DATA=](#) data table must contain all the regressors in the scale regression models.

To ensure that PROC CCDM correctly matches the values of regressors and the values of regression parameter estimates, you need to ensure that names of the regressors in the [DATA=](#) data table match the names of the regressors that you specify in the [SCALEMODEL](#) statement of the PROC [SEVSELECT](#) step that fits the severity models.

If you specify a [BY](#) statement in the PROC [SEVSELECT](#) step that creates the [SEVERITYEST=](#) data table, then you must specify an identical [BY](#) statement in the PROC CCDM step.

SEVERITYSTORE=libref.data-table**SEVSTORE=libref.data-table**

specifies the input data table that contains the context and estimates of the severity models in the form of an item store. Specifying the [OUTSTORE=](#) option in a PROC [SEVSELECT](#) step creates this item store data table.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 174. The *libref* must be identical to the *libref* that you specify in the [DATA=](#) option.

If your severity models contain classification or interaction effects, then you need to use this option instead of the [SEVERITYEST=](#) option to specify the estimates of the severity models. If you specify this option, you cannot specify the [SEVERITYEST=](#) option.

If you specify a [BY](#) statement in the PROC [SEVSELECT](#) step that creates the [SEVERITYSTORE=](#) item store, then you must specify an identical [BY](#) statement in the PROC CCDM step.

SIMULATIONMODE=mode

specifies the mode of simulating aggregate loss. You can specify one of the following *modes*:

COLLECTIVERISK**CR**

specifies the collective risk mode. Aggregate loss is defined as $S = \sum_{i=1}^N X_i$, where X_i denotes a continuous random variable that represents the severity of one loss event and N denotes the discrete random variable that represents the number of loss events you expect to see in a particular time period. The severity random variables $\{X_i\}$ that are associated with all loss events are assumed to be iid. For each random frequency draw N , PROC CCDM makes N random draws from the severity distribution and adds those severity draws to compute one sample point in the aggregate loss sample.

If you specify this mode (or choose it by default) and if you specify the frequency distribution by using the [COUNTMODEL](#) statement, then the count distribution must be a discrete distribution.

CUSTOM**CSTM**

specifies the custom mode. In this mode, you use programming statements to combine the severity and frequency values according to your business logic in order to compute the aggregate loss. If you specify this mode, then you must also use the `ADJUSTEDSEVERITY=` option to specify an aggregate loss symbol that your programming statements compute. For each random frequency draw N , PROC CCDM makes one random draw, X , from the severity distribution and sets the keyword symbols `_FREQ_` and `_SEV_` to N and X , respectively. It then executes your programming statements to compute one sample point in the aggregate loss sample.

This mode is especially useful if you also use the `SIMULATEDSYMBOL` statement to specify some stochastic variables that you have externally estimated to follow certain parametric probability distributions. For example, the following statements simulate a sample of the total paid loss `paidLoss` as $P = NEX + C$, where N denotes the average frequency of loss events per exposure and follows a lognormal distribution with $\mu = 3$ and $\sigma = 0.5$, E denotes the estimate of the total exposure in a particular time period and follows a normal distribution $\mathcal{N}(1000, 75)$, X denotes the average severity of a loss event, and C denotes the random change in losses from one time period to next and follows a normal distribution $\mathcal{N}(500, 10)$:

```
proc ccdm severitystore=mylib.sevstore simulationmode=custom
    adjustedseverity=paidLoss;
    countmodel lognormal(3,0.5);
    simulatedsymbol E ~ normal(1000, 75) C ~ normal(500, 10);
    paidLoss = _freq_ * E * _sev_ + C;
run;
```

PUREPREMIUM**PP**

specifies the pure premium mode. Aggregate loss is defined as $S = NX$, where N denotes the average frequency of loss events you expect to see in a particular time period and X denotes a continuous random variable that represents the average severity of the loss events in that time period. For each random frequency draw N , PROC CCDM makes one random draw, X , from the severity distribution and multiplies N and X to compute one sample point in the aggregate loss sample.

By default, `SIMULATIONMODE=COLLECTIVERISK`.

TRUNCATEZEROS

truncates (removes) zero-valued aggregate loss values from the aggregate loss sample.

The value of the aggregate loss is 0 when the sum of randomly drawn counts for all observations is 0 for the internally simulated counts or when the total count for a replication is 0 in the external counts data table.

If you omit this option, then PROC CCDM keeps the zero-valued aggregate losses in the output sample.

VARDEF=divisor

specifies the divisor to use in the calculation of variance, standard deviation, kurtosis, and skewness of the compound distribution sample. If the sample size is N , then you can specify one of the following values for the *divisor*:

DF

sets the divisor for variance to $N - 1$. This also changes the definitions of skewness and kurtosis.

N

sets the divisor to N .

By default, VARDEF=DF. For more information, see the section “Descriptive Statistics” on page 244.

BY Statement

BY *variable-list* ;

You can use the BY statement in the CCDM procedure to process the input data table in groups of observations defined by the BY variables.

If you specify the BY statement, then unlike procedures that do not use a CAS server, PROC CCDM does not need the input data table to be sorted in the order of the BY variables. As a consequence, the NOTSORTED and DESCENDING options have no effect, and PROC CCDM ignores them.

COUNTMODEL Statement

COUNTMODEL *distribution-name*(*parameter-value* < (*value-option*)>< , > ... *parameter-value* < (*value-option*)>) ;

The COUNTMODEL statement enables you to specify a parametric count model that has fixed parameter values. You must specify this statement if you do not specify the COUNTSTORE= option or the EXTERNALCOUNTS statement. On the other hand, PROC CCDM ignores this statement if you specify either the COUNTSTORE= option or the EXTERNALCOUNTS statement.

You must specify the following:

distribution-name

identifies your desired distribution. For a list of supported distributions and their acceptable names, see Table 7.2 or Table 7.3 in the section “Fixed Parameter-Value Distributions” on page 240. Each distribution accepts a fixed number of parameters in a specific order.

parameter-values

specifies the values of the distribution parameters. You must specify as many *parameter-values* as the expected number of parameters for the distribution that corresponds to *distribution-name*. PROC CCDM associates your specified values to parameters in the order the parameters are listed from top to bottom in the Parameters column of Table 7.2 or Table 7.3. The Parameters column also specifies the acceptable bounds, if any, for each parameter. Your specified *parameter-value* must be within those bounds.

You can optionally specify a standard error for each parameter by enclosing the following *value-option* in parentheses immediately after a parameter's value:

STDERROR=*number*

SE=*number*

number

specifies the standard error of the parameter. If you specify a value v and standard error s for a parameter, then PROC CCDDM assumes that the parameter follows a normal distribution $\mathcal{N}(v, s)$.

You can separate the parameter value specifications by using either a comma or a space.

The following example COUNTMODEL statement specifies that the counts follow the standard negative binomial distribution with parameters $p = 0.7$ and $n = 5$:

```
countmodel negbin(0.7 5);
```

If you request perturbation analysis by specifying a positive value for the **NPURTURBEDSAMPLES=** option and the parameters of your count distribution are stochastic, then it is recommended that you specify the standard error for each parameter. The following COUNTMODEL statement specifies that the counts follow the Weibull distribution with parameters $\theta = 50$ and $\tau = 0.75$ such that the standard errors for the θ and τ parameters are 3.5 and 0.25, respectively:

```
countmodel weibull(50(se=3.5), 0.75(0.25));
```

With this specification, when PROC CCDDM conducts the perturbation analysis, it sets the perturbed values of θ and τ to randomly drawn quantiles from the normal distributions $\mathcal{N}(50, 3.5)$ and $\mathcal{N}(0.75, 0.25)$, respectively.

PROC CCDDM permits a continuous count distribution (such as the Weibull distribution in the preceding example) only if you use the **SIMULATIONMODE=** option to specify a simulation mode other than the collective risk mode. For example, in the pure premium simulation mode (**SIMULATIONMODE=PP**), the count might represent an average claim frequency that an insurance company observes for a type of policy in a particular time period. Such an average frequency can have fractional values that are best modeled by a continuous distribution.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[Displayed Output](#)” on page 250. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* </ options> ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[Displayed Output](#)” on page 250. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

EXTERNALCOUNTS Statement

EXTERNALCOUNTS **COUNT**=*frequency-variable* <**ID**=*replication-id-variable*> ;

The EXTERNALCOUNTS statement enables you to specify externally simulated frequency counts. By default, PROC CCDM internally simulates the number of loss events by using the frequency model from the COUNTSTORE= item store or the COUNTMODEL statement. However, if you specify the EXTERNALCOUNTS statement, then PROC CCDM uses the counts that you specify in the DATA= data table and simulates only the severity values internally.

If you specify more than one EXTERNALCOUNTS statement, only the first one is used.

You must specify the following option in the EXTERNALCOUNTS statement:

COUNT=*count-variable*

specifies the variable that contains the simulated counts. This variable must be present in the DATA= data table.

You can also specify the following option in the EXTERNALCOUNTS statement:

ID=*replication-id-variable*

specifies the variable that contains the replicate identifier. This variable must be present in the DATA= data table.

When you specify this option, you must partition the observations of the DATA= data table by the BY variables, if any, and the ID= variable. One way to partition the table is to run the following DATA step when you upload a local SAS data set to a CAS table on the CAS server, where *BY-variable-list* is the list of BY variables that is identical to the list in the BY statement:

```
data mylib.counts (partition=(<BY-variable-list> <replication-id-variable>));
  set inlib.counts;
run;
```

If the DATA= data table is already loaded onto the CAS server and not partitioned by the required set of variables, then one way to partition the table is to use PROC CAS to run the **partition** action.

If you do not specify the BY statement, then the DATA= data table must be partitioned by just the ID= variable.

If you do not specify the ID= option, then PROC CCDM assumes that each observation represents one replication. In other words, the observation number serves as the default replication identifier.

The simulation process of using the external counts to generate the compound distribution sample is described in the section “[Simulation with External Counts](#)” on page 221.

OUTPUT Statement

OUTPUT OUT=*libref.data-table* < *variable-name-option* > < / *out-option* > ;

The OUTPUT statement enables you to specify the data table to output the generated compound distribution sample.

If you specify more than one OUTPUT statement, only the first one is used.

You must specify the output data table by using the following option:

OUT=*libref.data-table*

OUTSAMPLE=*libref.data-table*

specifies the output data table to contain the simulated compound distribution sample. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 174. The *libref* must be identical to the *libref* that you specify in the DATA= option. If you specify programming statements to adjust individual severity values, then this data table contains both unadjusted and adjusted samples.

You can specify the following *variable-name-options* to control the names of the variables that are created in the OUT= data table:

ADJSAMPLEVAR=*variable-name*

ADJSAMPLEVAR=(*variable-name* < ... *variable-name* >)

specifies the names of the variables to contain the adjusted compound distribution sample in the OUT= data table. If you specify more than one *variable-name*, then separate them with spaces and enclose them in parentheses.

This option is ignored if you do not specify the **ADJUSTEDSEVERITY=** option and the programming statements to adjust the simulated severity values.

If you specify *k* names in the ADJSAMPLEVAR= option and if you specify *m* symbols in the ADJUSTEDSEVERITY= option such that $m > k$, then the *k* names are used for the first *k* symbols. The variable names for the remaining $m - k$ symbols are the same as the corresponding symbol names. If $m < k$, then the extra $k - m$ names are ignored.

If you do not specify the ADJSAMPLEVAR= option, if you specify only one adjusted severity symbol in the ADJUSTEDSEVERITY= option, and if you specify the **ONEADJSEVCOMPAT** option, then “_AGGADJSEV_” is used by default.

SAMPLEVAR=*variable-name*

specifies the name of the variable to contain the simulated sample in the OUT= data table. If you do not specify this option, then “_AGGSEV_” is used by default.

Further, you can request that the perturbed samples be written to the OUT= data table by specifying the following *out-option*:

PERTURBOUT

writes all the perturbed samples to the OUT= data table. Each perturbed sample is identified by the value of the _DRAWID_ variable in the OUT= data table. A value of 0 indicates an unperturbed sample.

Separate compound distribution samples are generated for each combination of specified severity and frequency models. The _SEVERITYMODEL_ and _COUNTMODEL_ columns in the OUT= data table identify the severity and frequency models, respectively, that are used to generate the sample in the SAMPLEVAR= variable and ADJSAMPLEVAR= variables.

For more information about the variables in the OUTSAMPLE= data table, see the section “[OUTSAMPLE= Data Table](#)” on page 248.

OUTSUM Statement

OUTSUM **OUT=***libref.data-table* *statistic-keyword*<=*variable-name*> <... *statistic-keyword*<=*variable-name*>> <*outsum-options*> ;

The OUTSUM statement enables you to specify the data table in which PROC CCDM writes the summary statistics of the compound distribution samples.

If you specify more than one OUTSUM statement, only the first one is used.

You must specify the output data table by using the following option:

OUT=*libref.data-table*

OUTSUM=*libref.data-table*

specifies the output data table that contains the summary statistics of each of the simulated compound distribution samples. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 174. The *libref* must be identical to the *libref* that you specify in the DATA= option.

You can control the summary statistics that appear in this data table by specifying different *statistic-keywords* and *outsum-options*.

You can request that one or more predefined statistics of the compound distribution sample be written to the OUTSUM= data table. For each specification of the form *statistic-keyword*<=*variable-name*>, the statistic that is specified by the *statistic-keyword* is written to a variable named *variable-name*. If you do not specify the *variable-name*, then the statistic is written to a variable named *statistic-keyword*. You can specify the following *statistic-keywords*:

KURTOSIS

KURT

specifies the kurtosis of the compound distribution sample.

MEAN

specifies the mean of the compound distribution sample.

P01

specifies the 1st percentile of the compound distribution sample.

P05

specifies the 5th percentile of the compound distribution sample.

P10

specifies the 10th percentile of the compound distribution sample.

P25

Q1

specifies the lower or 1st quartile (the 25th percentile) of the compound distribution sample.

P50

MEDIAN

Q2

specifies the median (the 50th percentile) of the compound distribution sample.

P75

Q3

specifies the upper or 3rd quartile (the 75th percentile) of the compound distribution sample.

P90

specifies the 90th percentile of the compound distribution sample.

P95

specifies the 95th percentile of the compound distribution sample.

P99

specifies the 99th percentile of the compound distribution sample.

P99_5

P995

specifies the 99.5th percentile of the compound distribution sample.

QRANGE

specifies the interquartile range ($Q3-Q1$) of the compound distribution sample.

SKEWNESS

SKEW

specifies the skewness of the compound distribution sample.

STDDEV**STD**

specifies the standard deviation of the compound distribution sample.

All percentiles are computed by using the method that you specify in the **PCTLDEF=** option in the PROC CCDF statement. You can also request additional percentiles to be reported in the OUTSUM= data table by specifying the following *outsum-options*:

PCTLPTS=percentile-list

specifies one or more percentiles that you want to be computed and written to the OUTSUM= data table. This option is useful if you need to request percentiles that are not available in the preceding list of *statistic-keyword* values. Each percentile value must belong to the (0,100) open interval. The *percentile-list* is a comma-separated list of numbers. You can also use a list notation of the form “< number1 > to < number2 > by < increment >”. For example, the following two options are equivalent:

```
pctlpts=10, 20, 99.6, 99.7, 99.8, 99.9
pctlpts=10, 20, 99.6 to 99.9 by 0.1
```

You can specify the name of the variable for a particular percentile value in the PCTLNAME= option.

PCTLNAME=percentile-variable-name-list

specifies the names of the variables that contain the estimates of the percentiles that you request by using the PCTLPTS= option.

If you specify the PCTLNAME= option, you must specify it as the last option in the OUTSUM statement.

If you do not specify the PCTLNAME= option, then each percentile value t in the list of values in the PCTLPTS= option is written to the variable named “Pt,” where the decimal point in t , if any, is replaced by an underscore.

The *percentile-variable-name-list* is a space-separated list of names. You can also use a shortcut notation of <prefix> m –<prefix> n for two integers m and n ($m < n$) to generate the following list of names: <prefix> m , <prefix> $m + 1$, ..., and <prefix> n . For example, the following two options are equivalent:

```
pctlname=p1 p2 pc5 pc6 pc7 pc8 pc9 pc10
pctlname=p1 p2 pc5-pc10
```

The name in j th position of the expanded name list of the PCTLNAME= option is used to create a variable for a percentile value in the j th position of the expanded value list of the PCTLPTS= option. If you specify k_n names in the PCTLNAME= option and k_v percentile values in the PCTLPTS= option, and if $k_n < k_v$, then the first k_n percentiles are written to the variables that you specify. The remaining $k_v - k_n$ percentiles are written to the variables that have the name of the form P_t , where t is the text representation of the percentile value that is formed by retaining at most PCTLNDEC= digits after the decimal point and replacing the decimal point with an underscore (‘_’). For example, assume that you specify the following options:

```
pctlpts=10, 20, 99.3 to 99.5 by 0.1, 99.995
pctlname=pten ptwenty ninenine3-ninenine5
```

Then PROC CCDDM writes the 10th and 20th percentiles to the variables `pten` and `ptwenty`, respectively; the 99.3rd through 99.5th percentiles to the variables `ninenine3`, `ninenine4`, and `ninenine5`, respectively; and the remaining 99.995th percentile to the variable `P99_995`.

If a percentile value in the `PCTLPTS=` option matches a percentile value that is implied by one of the predefined percentile statistics and you specify the corresponding *statistic-keyword*, then the variable name that is implied by the *statistic-keyword*≤*variable-name* specification takes precedence over the name that you specify in the `PCTLNAME=` option. For example, assume that you specify the predefined percentile statistic of `P95`, as in the following `OUTSUM` statement:

```
outsum out=mypctls p95=ninetyfifth
      pctlpts=95 to 99 by 1 pctlname=pct95-pct99;
```

Then the 95th percentile is written to the variable `ninetyfifth` instead of the variable `pct95` that the `PCTLNAME=` option implies.

PCTLNDEC=integer-value

specifies the maximum number of decimal places to use while creating the names of the variables for the percentile values in the `PCTLPTS=` option. By default, `PCTLNDEC=3`. For example, for a percentile value of 99.9995, PROC CCDDM creates a variable named `P99_999` by default, but if you specify `PCTLNDEC=4`, then the variable is named `P99_9995`.

The `PCTLNDEC=` option is used only for percentile values for which you do not specify a name in the `PCTLNAME=` option.

Note that all variable names in the `OUTSUM=` data table have a limit of 32 characters. If a name exceeds that limit, then it is truncated to the first 32 characters. For more information about the variables in the `OUTSUM=` data table, see the section “[OUTSUM= Data Table](#)” on page 249.

SEVERITYMODEL Statement

```
SEVERITYMODEL severity-model-list < / definition-option > ;
```

The `SEVERITYMODEL` statement specifies one or more severity distribution models that you want to use in simulating a compound distribution sample. The *severity-model-list* is a space-separated list of names of severity models that you would use with PROC SEVSELECT’s `DIST` statement. The `SEVERITYEST=` data table or the `SEVERITYSTORE=` item store must contain all the severity models in the list. If you specify the `SEVERITYEST=` data table and you specify a name that does not appear in the `_MODEL_` column of the `SEVERITYEST=` data table, then that name is ignored. Similarly, if you specify the `SEVERITYSTORE=` item store and a severity model by that name does not appear in the item store, then that name is ignored.

You can also specify the following *definition-option* (after a slash) in the `SEVERITYMODEL` statement:

DEFINITIONSTABLE="CAS-table-name" <(CASLIB="caslib")>

SEVDEF="CAS-table-name" <(CASLIB="caslib")>

specifies a data table on the CAS server that contains the severity distribution function definitions. If you omit the CASLIB= option, then PROC CCDM assumes that the table is in the caslib that is active in the current CAS session. To specify a different caslib, use the CASLIB= option.

If you omit the DEFINITIONSTABLE= option, then PROC CCDM loads the severity function definitions from the library Sashelp.Svrtdist and the libraries that you specify in the CMPLIB= SAS system option, and it creates a temporary CAS table named cdm_svrtdist in the current CAS session. However, this process requires additional time and effort that you can avoid by specifying the DEFINITIONSTABLE= option. You can use the CAS table that the SEVSELECT procedure creates when it fits the severity models. For more information, see the OUTFUNCDEF= option in the DIST statement of the SEVSELECT procedure.

If you specify more than one SEVERITYMODEL statement, only the first one is used.

If you do not specify a SEVERITYMODEL statement, then this is equivalent to specifying all the severity models that appear in the SEVERITYEST= data table or the SEVERITYSTORE= item store.

A compound distribution sample is generated for each severity model by compounding that severity model with the frequency model that you specify in the COUNTSTORE= item store or the COUNTMODEL statement, or the external frequency model that is encoded by the COUNT= variable that you specify in the EXTERNALCOUNTS statement.

SIMULATEDSYMBOL Statement

SIMULATEDSYMBOL *symbol* ~ *distribution-name*(*parameter-value-list*) <... *symbol* ~ *distribution-name*(*parameter-value-list*)> ;

The SIMULATEDSYMBOL statement specifies one or more symbols and their associated probability distributions that you want PROC CCDM to simulate. You must use these symbols in the programming statements that define one or more of the ADJUSTEDSEVERITY= symbols. Hence, this statement is ignored if you do not specify the ADJUSTEDSEVERITY= option.

Each simulated *symbol* is a random variable that follows a specific parametric probability distribution, which you specify by using the *distribution-name* followed by a *parameter-value-list* that is enclosed in parentheses. The list of supported distributions is shown in Table 7.2 and Table 7.3 in the section “Fixed Parameter-Value Distributions” on page 240. The *distribution-name* is one of the names that appear in the Names column of those tables, and *parameter-value-list* is a comma- or space-separated list of numbers that represent the values of the parameters in the order in which the parameters appear top to bottom in the Parameters column for the specified *distribution-name* in those tables. The SIMULATEDSYMBOL statement can contain multiple such symbol specifications separated by a space.

The simulated symbols are useful in applications where your adjusted severity depends on some random variables that you have estimated externally. For example, if you are simulating the pure premium by specifying the SIMULATIONMODE=PP option, and if your frequency is modeled as the average number of claims per exposure, then to estimate the loss that you expect to pay for multiple exposures, you need to multiply the pure premium by the exposure. Typically, exposure is a random variable that you estimate externally by using a regression or time series forecasting technique. The SIMULATEDSYMBOL statement enables you to easily incorporate such stochastic variables in the simulation process. For example, if exposure

denotes the exposure and if you have estimated it to follow a normal distribution $\mathcal{N}(\mu = 100, \sigma = 7.5)$, then you can specify the following statements to instruct PROC CCDM to estimate the distribution of the paid loss (paidLoss) that accounts for the randomness of exposure:

```
proc ccdm severitystore=mylib.sevstore countstore=mylib.cntstore
    simulationmode=pp adjustedseverity=paidLoss;
    simulatedsymbol exposure ~ normal(100,7.5);
    paidLoss = _sev_ * exposure;
run;
```

Programming Statements

In PROC CCDM, you can use a series of programming statements that use variables in the DATA= data table to define multiple adjustments to an individual severity value. Each adjusted severity value is aggregated to form a separate adjusted compound distribution sample.

The programming statements are executed for each simulated individual severity value. The observation of the input data table that is used to evaluate the programming statements is determined by the simulation process that is described in the section “[Simulation Processes for Collective Risk Simulation Mode](#)” on page 217.

For more information, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 229.

Details: CCDM Procedure

Specifying Scenario Data in the DATA= Data Table

A scenario represents a state of the world for which you want to estimate the distribution of aggregate losses. The state consists of one or more entities that generate the loss events. For example, an entity might be an individual who has an insurance policy or an organization that has a workers’ compensation policy. Each entity has some characteristics of its own and some external factors that affect the frequency with which it generates the losses and the severity of each loss. For example, characteristics of an individual automobile insurance policyholder can include various demographics of the individual and various features of the automobile. Characteristics of an organization that has a workers’ compensation policy can be the number of employees, revenue, ratio of temporary to permanent employees, and so on. The organization can also be affected by external macroeconomic factors such as GDP and unemployment rate of the country where it operates and factors that affect its industry. You need to quantify and specify all these characteristics as external factors (regressors) when you fit severity and frequency models.

You should specify all the information about a scenario in the DATA= data table that you specify in the PROC CCDM statement. Each observation in the DATA= data table encodes the characteristics of an entity. For proper simulation of severities, you must specify in the DATA= data table all the characteristics that you use as regressors in the severity scale regression models. When you use the COUNTSTORE= option to specify the frequency model, you must specify in the DATA= data table all the characteristics that you use as regressors in the frequency model in order to properly simulate the counts. All the regressors are expected to

have nonmissing values. If any of the regressors have a missing value in an observation, then that observation is ignored.

The information in the DATA= data table is interpreted as follows, based on whether you specify the EXTERNALCOUNTS statement:

- If you do not specify the EXTERNALCOUNTS statement, then all the observations in the data table form a scenario. The observations are used together to compute one random draw from the compound distribution. The total number of draws is equal to the value that you specify in the NREPLICATES= option. The simulation process is described in the section “[Simulation with Regressors and No External Counts](#)” on page 218 and illustrated using an example in the section “[Illustration of Aggregate Loss Simulation Process](#)” on page 219.
- If you specify the EXTERNALCOUNTS statement, then the DATA= data table is expected to contain multiple replications (draws) of the frequency counts that you simulate externally for a scenario. The DATA= data table must contain the COUNT= variable that you specify in the EXTERNALCOUNTS statement. The replications are identified by the observation number or the ID= variable that you specify in the EXTERNALCOUNTS statement. For each observation in a given replication, the COUNT= variable is expected to contain the count of losses that are generated by the entity associated with that observation. All the observations of a given replication are used together to compute one random draw from the compound distribution. The size of the compound distribution sample is equal to the number of distinct replications that you specify in the DATA= data table, multiplied by the value that you specify in the NREPLICATES= option. The simulation process is described in the section “[Simulation with External Counts](#)” on page 221 and illustrated using an example in the section “[Illustration of the Simulation Process with External Counts](#)” on page 221.

In both cases, an observation can also contain severity adjustment variables that you can use to adjust the severity of the losses generated by that entity, based on some policy rules. For more information about simulating the adjusted compound distribution sample, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 229.

If you specify severity and frequency models that contain no regression effects, and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then you do not need to specify the DATA= data table. This case corresponds to a fixed scenario that is represented entirely by the distribution parameters of the models.

Simulation Processes for Collective Risk Simulation Mode

This section describes the simulation processes for the collective risk mode, which is the simulation mode that PROC CCDM uses when you do not specify the SIMULATIONMODE= option or when you specify SIMULATIONMODE=CR. For other simulation modes, see the section “[Simulation Processes for Pure Premium and Custom Simulation Modes](#)” on page 225.

PROC CCDM selects a simulation process based on whether you specify external counts or you request that PROC CCDM simulate the counts and on whether or not the severity or frequency models contain regression effects. The following sections describe the processes for the different possibilities.

Simulation with No Regressors and No External Counts

If you specify severity and frequency models that contain no regression effects and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then PROC CCDM uses the simulation process that this section describes.

The process is described for one severity distribution, *dist*. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution.

The following steps are repeated M times to generate a compound distribution sample of size M , where M is the value of the NREPLICATES= option:

1. Use the frequency model that you specify in the COUNTSTORE= item store or the COUNTMODEL statement to draw a value N from the count distribution, where N denotes the number of loss events that are expected to occur in the time period that is being simulated. Adjust N to conform to the upper limit by setting it equal to $\min(N, N_{\max})$, where $N_{\max}$ is the value of the MAXCOUNTDRAW= option.
2. Use the parameter estimates of the severity distribution *dist* (which are read from the SEVERITYEST= data table or the SEVERITYSTORE= item store that you specify) to draw N severity values, X_j ($j = 1, \dots, N$). (For more information about how PROC CCDM draws an individual severity value, see the section “Making a Random Draw from a Severity Distribution” on page 224.)
3. Add the N severity values that are drawn in step 2 to compute one sample point, S , of the compound distribution sample as

$$S = \sum_{j=1}^N X_j$$

Although it is more common to fit a frequency model that contains regressors, the CNTSELECT procedure enables you to fit a frequency model that does not contain regressors. If you do not specify any regressors in the MODEL statement of PROC CNTSELECT, then it fits a model that contains only an intercept.

Simulation with Regressors and No External Counts

If the severity or frequency models contain regression effects and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then you must specify a DATA= data table to provide values of the regression variables, which together represent a scenario for which you want to simulate the CDM. In this case, PROC CCDM uses the simulation process that this section describes.

The process is described for one severity distribution. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution.

Note that you are performing scenario analysis when regression effects are present. Let K denote the number of observations that form the scenario. This is the number of observations either in the current BY group or in the entire DATA= data table if you do not specify the BY statement. If $K > 1$, then you are modeling the scenario for a group of entities. If $K = 1$, then you are modeling the scenario for one entity.

The following steps are repeated M times to generate a compound distribution sample of size M , where M is the value of the NREPLICATES= option:

1. For each observation k ($k = 1, \dots, K$), draw a count N_k from the frequency model, where N_k denotes the number of loss events that are expected to be generated by entity k in the time period that is being simulated. If you specify the COUNTSTORE= item store, then draw N_k from the frequency model whose parameters are determined by the frequency regressors in observation k . If you specify the COUNTMODEL statement, the frequency model does not depend on the regressors, so draw N_k by using the information in the COUNTMODEL statement. Adjust N_k to conform to the upper limit by setting it equal to $\min(N_k, N_{\max})$, where $N_{\max}$ is the value of the MAXCOUNTDRAW= option.
2. Add counts from all observations to compute $N = \sum_{k=1}^K N_k$, which is the total number of loss events that are expected to occur in the time period that is being simulated.
3. Make N random draws from the severity distribution and add them to generate one sample point of the compound distribution sample. If you specify a scale regression model for the severity distribution, then use the values of severity regressors in the k th observation to compute the scale parameter of the severity distribution and make N_k severity draws from that distribution. (For more information about how PROC CCDM makes a random severity draw, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 224.)

If you specify the BY statement, then a separate sample of size M is created for each BY group in the DATA= data table.

Illustration of Aggregate Loss Simulation Process

As an illustration of the simulation process, consider a very simple example of analyzing the distribution of an aggregate loss that is incurred by a set of policyholders of an automobile insurance company in a period of one year. It is postulated that the frequency and severity distributions depend on three variables: Age, Gender (1: female, 2: male), and CarType (1: sedan, 2: sport utility vehicle). So these variables are used as regressors while you fit the count model and severity scale regression model by using the CNTSELECT and SEVSELECT procedures, respectively. Now, consider that you want to use the fitted frequency and severity models to estimate the distribution of the aggregate loss that is incurred by a set of five policyholders. The characteristics of the five policyholders are encoded in a data table named mylib.Scenario that has the following contents:

Obs	age	gender	carType
1	30	2	1
2	25	1	2
3	45	2	2
4	33	1	1
5	50	1	1

The column Obs contains the observation number. It is shown only for the purpose of illustration. It does not need to be present in the data table. The following PROC CCDM step simulates the scenario in the data table mylib.Scenario:

```
proc ccdm data=mylib.scenario
    severityest=<severity parameter estimates data table>
    countstore=<count model store> nreplicates=<sample size>;
    severitymodel <severity distribution name(s)>;
run;
```

The following process generates a sample from the aggregate loss distribution for the scenario in the data table mylib.Scenario:

1. Use the values Age=30, Gender=2, and CarType=1 in the first observation to draw a count from the count distribution. Let that count be 2. Repeat the process for the remaining four observations. Let the counts be as shown in the Count column in the following table:

Obs	age	gender	carType	count
1	30	2	1	2
2	25	1	2	1
3	45	2	2	2
4	33	1	1	3
5	50	1	1	0

The Count column is shown for illustration only; it is not added as a variable to the DATA= data table.

2. The simulated counts from all the observations are added together to get a value of $N = 8$. This means that for this particular sample point, you expect a total of eight loss events in a year from these five policyholders.
3. For the first observation, the scale parameter of the severity distribution is computed by using the values Age=30, Gender=2, and CarType=1. That value of the scale parameter is used together with estimates of the other parameters from the SEVERITYEST= data table to make two draws from the severity distribution. Each draw simulates the magnitude of the loss that is expected from the first policyholder. The process is repeated for the remaining four policyholders. The fifth policyholder does not generate any loss event for this particular sample point, so no severity draws are made by using the fifth observation. Let the severity draws, rounded to integers for convenience, be as shown in the _SEV_ column in the following table:

Obs	age	gender	carType	count	_sev_		
1	30	2	1	2	350	2100	
2	25	1	2	1	4500		
3	45	2	2	2	700	4300	
4	33	1	1	3	600	1500	950
5	50	1	1	0			

The _SEV_ column is shown for illustration only; it is not added as a variable to the DATA= data table.

PROC CCDM adds the severity values of the eight draws to compute an aggregate loss value of 15,000. After recording this amount in the sample, the process returns to step 1 to compute the next point in the aggregate loss sample. For example, in the second iteration, the count distribution of each policyholder might generate one loss event, for a total of five loss events, and the five severity draws from the severity distributions that govern each policyholder might add up to 5,000. Then, the value of 5,000 is recorded as the second point in the aggregate loss sample. The process continues until M aggregate loss sample points are simulated, where the M is the value that you specify in the NREPLICATES= option.

Simulation with External Counts

If you specify externally simulated counts by using the `EXTERNALCOUNTS` statement, then each replication in the input data table represents the loss events that are generated by an entity. An entity can be an individual or organization for which you want to estimate the compound distribution. If an entity has any characteristics that are used as external factors (regressors) in developing the severity scale regression model, then you must specify the values of those factors in the `DATA=` data table. If you specify the `ID=` variable, then multiple observations for the same replication ID represent different entities in a group for which you are simulating the CDM.

This section describes the simulation process that `PROC CCDM` uses in the presence of externally simulated counts.

The process is described for one severity distribution. If you specify multiple severity distributions in the `SEVERITYMODEL` statement, then the process is repeated for each specified distribution.

Let there be M distinct replications in the current `BY` group of the `DATA=` data table or in the entire `DATA=` data table if you do not specify the `BY` statement. A replication is identified by either the observation number or the value of the `ID=` variable that you specify in the `EXTERNALCOUNTS` statement.

For each of the M values of the replication identifier, the following steps are executed R times, where R is the value of the `NREPLICATES=` option:

1. If there are K ($K \geq 1$) observations for the current value of the replication identifier, then for each observation k ($k = 1, \dots, K$), set N_k equal to the value of the `COUNT=` variable in that observation. Adjust N_k to conform to the upper limit by setting it equal to $\min(N_k, N_{\max})$, where $N_{\max}$ is the value of the `MAXCOUNTDRAW=` option.
2. Compute the total number of losses N across all observations of the replication as $N = \sum_{k=1}^K N_k$.
3. Make N random draws from the severity distribution and add them to generate one sample point of the compound distribution sample. If you specify a scale regression model for the severity distribution, then use the values of severity regressors in the k th observation to compute the scale parameter of the severity distribution and make N_k severity draws from that distribution. (For more information about how `PROC CCDM` makes a random severity draw, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 224.)

This process generates a compound distribution sample of size $M \times R$. If you specify the `BY` statement, then a separate sample of size $M \times R$ is created for each `BY` group in the `DATA=` data table.

Illustration of the Simulation Process with External Counts

To illustrate the simulation process, consider the following simple example. In this example, your severity model does not contain any regressors. An example that uses a severity scale regression model is illustrated later. Assume that you have made 10 random draws from an external count model and recorded them in the variable `ExtCount` of a data table named `mylib.Counts1`, as follows:

Obs	extCount
1	3
2	2
3	0
4	1

5	3
6	4
7	1
8	2
9	0
10	5

Because the data table does not contain an ID= variable, the observation number that is shown in the Obs column acts as the replicate identifier. The following PROC CCDM step simulates an aggregate loss sample by using the data table mylib.Counts1:

```
proc ccdm data=mylib.counts1 nreplicates=5
    severityest=<severity parameter estimates data table>;
    severitymodel <severity distribution name(s)>;
    externalcounts count=extCount;
run;
```

The simulation process works as follows:

1. For the first replication, which is associated with the first observation, three severity values are drawn from the severity distribution by using the parameter estimates that you specify in the SEVERITYEST= data table. If the severity values are 150, 500, and 320, then their sum of 970 is recorded as the first point of the aggregate loss sample. Because the value of the NREPLICATES= option is 5, this process of drawing three severity values and adding them together to form a point of the aggregate loss sample is repeated four more times to generate a total of five sample points that correspond to the first observation.
2. For the second replication, two severity values are drawn from the severity distribution. If the severity values are 450 and 100, then their sum of 550 is recorded as a point of the aggregate loss sample. This process of drawing two severity values and adding them together to form a point of the aggregate loss sample is repeated four more times to generate a total of five sample points that correspond to the second observation.
3. The process continues until all the replications, which are observations in this case, are exhausted.

The process results in an aggregate loss sample of size 50, which is equal to the number of replications in the data table (10) multiplied by the value of the NREPLICATES= option (5).

Now, consider an example in which the severity models in the SEVERITYEST= data table are scale regression models. In this case, the severity distribution that is used for drawing the severity value is determined by the values of regressors in the observation that is being processed. Consider that you want to simulate the aggregate loss that one policyholder incurs and you have recorded, in the variable ExtCount, the results of 10 random draws from an external count model. The DATA= data table has the following contents:

Obs	age	gender	carType	extCount
1	30	2	1	5
2	30	2	1	2
3	30	2	1	0
4	30	2	1	1
5	30	2	1	3
6	30	2	1	4
7	30	2	1	1

8	30	2	1	2
9	30	2	1	0
10	30	2	1	5

The simulation process in this case is the same as the process in the previous case of no regressors, except that the severity distribution that PROC CCDM uses for drawing the severity values has a scale parameter that is determined by the values of the regressors *Age*, *Gender*, and *CarType* in the observation that is being processed. In this particular example, all observations have the same value for all regressors, indicating that you are modeling a scenario in which the characteristics of the policyholder do not change during the time for which you have simulated the number of events. You can also model a scenario in which the characteristics of the policyholder change by recording those changes in the values of the appropriate regressors.

Extending this example further, consider that you want to analyze the distribution of the aggregate loss that is incurred by a group of policyholders, as in the example in the section “[Illustration of Aggregate Loss Simulation Process](#)” on page 219. Let the data table *mylib.Counts2* record multiple replications of the number of losses that each policyholder might generate. The contents of the data table *mylib.Counts2* are as follows:

Obs	replicateId	age	gender	carType	extCount
1	1	30	2	1	2
2	1	25	1	2	1
3	1	45	2	2	3
4	1	33	1	1	5
5	1	50	1	1	1
6	2	30	2	1	3
7	2	25	1	2	2
8	1	45	2	2	0
9	2	33	1	1	4
10	2	50	1	1	1

The variable *ReplicateId* records the identifier for the replication. Each replication contains multiple observations, such that each observation represents one of the policyholders that you are analyzing. For simplicity, only the first two replications are shown here.

The following PROC CCDM step simulates an aggregate loss sample by using the data table *mylib.Counts2*:

```
proc ccdm data=mylib.counts2 nreplicates=3
    severityest=<severity parameter estimates data table>;
    severitymodel <severity distribution name(s)>;
    externalcounts count=extCount id=replicateId;
    output out=aggloss samplevar=totalLoss;
run;
```

When you specify an *ID=* variable in the *EXTERNALCOUNTS* statement, the *DATA=* data table must be partitioned by the *ID=* variable as noted in the description of the *ID=* option.

The simulation process works as follows:

1. First, the five observations of the first replication (*ReplicateId*=1) are analyzed. For the first observation (*Obs*=1), the scale parameter of the severity distribution is computed by using the values *Age*=30, *Gender*=2, and *CarType*=1. That value of the scale parameter is used together with estimates of the other parameters from the *SEVERITYEST=* data table to make two draws from the severity distribution. Next, the regressor values of the second observation are used to compute the scale parameter of the severity distribution, which is used to make one severity draw. The process continues such that the

regressor values in the third, fourth, and fifth observations are used to select which severity distribution to make three, five, and one draws from, respectively. Let the severity values that are drawn from the observations of this replication be as shown in the `_SEV_` column in the following table, where the `_SEV_` column is shown for illustration only; it is not added as a variable to the `DATA=` data table.

Obs	replicateId	age	gender	carType	extCount	_sev_
1	1	30	2	1	2	700 500
2	1	25	1	2	1	5000
3	1	45	2	2	3	900 1400 300
4	1	33	1	1	5	350 2000 150 800 600
5	1	50	1	1	1	250

The values of all 12 severity draws are added together to compute and record the value of 12,950 as the first point of the aggregate loss sample. Because you specify `NREPLICATES=3` in the `PROC CCDM` step, this process of making 12 severity draws from the respective observations is repeated two more times to generate a total of three sample points for the first replication.

2. The five observations of the second replication (`ReplicateId=2`) are analyzed next to draw three, two, four, and one severity values from the severity distributions, with scale parameters that are determined by the regressor values in the sixth, seventh, ninth, and tenth observations, respectively. The 10 severity values are added together to form a point of the aggregate loss sample. This process of making 10 severity draws from the respective observations is repeated two more times to generate a total of three sample points for the second replication.

If your data table `mylib.Counts2` contains 10,000 distinct values of `ReplicateId`, then 30,000 observations are written to the data table `mylib.AgglLoss` that you specify in the `OUTPUT` statement of the preceding `PROC CCDM` step. Because you specify `SAMPLEVAR=TotalLoss` in the `OUTPUT` statement, the aggregate loss sample is available in the `TotalLoss` column of the data table `mylib.AgglLoss`.

Making a Random Draw from a Severity Distribution

`PROC CCDM` uses a simple process to make a random draw from the severity distribution. It first makes a draw from the uniform distribution. If this value is denoted by p , then the corresponding severity draw is computed as $x = F^{-1}(p)$, where F denotes the cumulative distribution function (CDF) of the severity distribution.

For a severity distribution *dist*, `PROC CCDM` computes the inverse of the CDF (that is, F^{-1}) by evaluating the `dist_QUANTILE` function. If `PROC CCDM` does not find the definition of the `dist_QUANTILE` function in the `DEFINITIONSTABLE=` table or in the libraries that you specify in the `CMPLIB=` system option, it uses a numerical inversion algorithm to invert the CDF function at p . This algorithm yields an approximate inverse and can be slow because it makes multiple calls to the `dist_CDF` or `dist_LOGCDF` function of the distribution. So it is recommended that you define the `dist_QUANTILE` function whenever possible for any custom severity distribution that you define for use with `PROC CCDM`. For more information about defining custom severity distributions, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811 in Chapter 26, “[SEVSELECT Procedure](#).” The `dist_QUANTILE` function is already defined for all severity distributions in the library `Sashelp.Svrtdist`.

To evaluate the `dist_QUANTILE` or `dist_CDF` function, `PROC CCDM` uses the distribution parameters that it reads from the `SEVERITYEST=` data table or the `SEVERITYSTORE=` item store. If the severity model

contains regression effects, then PROC CCDM computes the scale parameter of the severity distribution by using the base value of the scale parameter and the values of the regression effects.

Making a Random Draw from a Truncated Severity Distribution

PROC CCDM assumes that the severity distribution that you specify in the `SEVERITYEST=` data table or the `SEVERITYSTORE=` item store is an untruncated distribution—that is, a distribution that has the full support. For severity distributions that have a scale parameter, full support is $[0, \infty)$, and for severity distributions that have a log-scale parameter, full support is $(-\infty, \infty)$. If you want PROC CCDM to make draws from a truncated distribution, which is especially useful in insurance applications, then you can specify the `LEFTTRUNCATION=` or `RIGHTTRUNCATION=` options in the PROC CCDM statement.

Let L and R denote the left-truncation threshold and right-truncation limit that you specify in the `LEFTTRUNCATION=` and `RIGHTTRUNCATION=` options, respectively. Let F and $F_{L,R}$ denote the cumulative distribution functions (CDF) of the untruncated and truncated severity distributions, respectively. Not specifying the `LEFTTRUNCATION=` option is equivalent to specifying $L = F^{-1}(0)$ —that is, $F(L) = 0$. Not specifying the `RIGHTTRUNCATION=` option is equivalent to specifying $R = F^{-1}(1)$ —that is, $F(R) = 1$.

PROC CCDM makes a random draw from the truncated severity distribution by using the following steps:

1. Draw a value p_t from the uniform distribution. This is the CDF of the truncated distribution.
2. Compute p , which represents the CDF of the untruncated distribution, as

$$p = p_t(F(R) - F(L)) + F(L)$$

3. Compute the severity value x by inverting the CDF of the untruncated distribution as $x = F^{-1}(p)$. The value of x that is computed in this manner is essentially a random draw from $F_{L,R}$ —that is, $x = F_{L,R}^{-1}(p_t)$. The inversion of the untruncated CDF is described in the section “[Making a Random Draw from a Severity Distribution](#)” on page 224.

Simulation Processes for Pure Premium and Custom Simulation Modes

This section describes the simulation processes for pure premium mode (when you specify `SIMULATIONMODE=PP`) and custom mode (when you specify `SIMULATIONMODE=CUSTOM`).

Like the simulation processes for the collective risk mode, which are described in the section “[Simulation Processes for Collective Risk Simulation Mode](#)” on page 217, the simulation processes for pure premium and custom modes also depend on whether you specify external counts or you request that PROC CCDM simulate the counts and on whether or not the severity or frequency models contain regression effects. The following sections describe the processes for the different possibilities.

Simulation with No Regressors and No External Counts

If you specify severity and frequency models that contain no regression effects and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then PROC CCDM uses the simulation process that this section describes.

The process is described for one severity distribution, *dist*. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution. Let D denote the number of adjusted severity symbols that you specify in the ADJUSTEDSEVERITY= option.

The following steps are repeated M times to generate a compound distribution sample of size M , where M is the value of the NREPLICATES= option:

1. Use the frequency model that you specify in the COUNTSTORE= item store or the COUNTMODEL statement to draw a value N from the count distribution.
2. Use the parameter estimates of the severity distribution *dist* (which are read from the SEVERITYEST= data table or the SEVERITYSTORE= item store that you specify) to draw a severity value X . (For more information about how PROC CCDM draws an individual severity value, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 224.)
3. Compute one sample point of different aggregate loss samples as follows:
 - For SIMULATIONMODE=CUSTOM, do not compute the unadjusted aggregate loss sample point. Because D must be nonzero for this mode, compute D sample points S^d of the D adjusted aggregate loss samples as follows:
 - a) Set the _FREQ_ and _SEV_ placeholder symbols equal to N and X , respectively. (For more information about the placeholder symbols, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 229.)
 - b) Execute your severity adjustment program.
 - c) Retrieve the adjusted severity values X_k^d ($d = 1, \dots, D$) that your adjustment program computes.
 - d) Set the sample point S^d of the d th adjusted aggregate loss sample equal to X_k^d ($d = 1, \dots, D$).
 - For SIMULATIONMODE=PUREPREMIUM, multiply the frequency and severity values to compute the sample point S of the unadjusted aggregate loss sample as $S = NX$. If $D > 0$, then follow the same steps as the custom mode to compute D sample points S^d of the D adjusted aggregate loss samples, except for the following two differences:
 - Do not fill the _FREQ_ placeholder symbol, because it is valid only in the custom mode.
 - Compute S^d as $S^d = NX^d$ ($d = 1, \dots, D$).

Simulation with Regressors and No External Counts

If the severity or frequency models contain regression effects and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then you must specify a DATA= data table to provide values of the regression variables, which together represent a scenario for which you want to simulate the CDM. In this case, PROC CCDM uses the simulation process that this section describes.

The process is described for one severity distribution. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution. Let D denote the number of adjusted severity symbols that you specify in the ADJUSTEDSEVERITY= option.

Note that you are performing scenario analysis when regression effects are present. Let K denote the number of observations that form the scenario. This is the number of observations either in the current BY group or in the entire DATA= data table if you do not specify the BY statement. If $K > 1$, then you are modeling the scenario for a group of entities. If $K = 1$, then you are modeling the scenario for one entity.

The following steps are repeated M times to generate a compound distribution sample of size M , where M is the value of the NREPLICATES= option:

1. For each observation k ($k = 1, \dots, K$), draw a value N_k from the frequency model. If you specify the COUNTSTORE= item store, then the parameters of the frequency model are determined by the frequency regressors in observation k . If you specify the COUNTMODEL statement, the frequency model does not depend on the regressors, so N_k is drawn by using the information that you specify in the COUNTMODEL statement.
2. For each observation k ($k = 1, \dots, K$), draw a severity value X_k from the severity model. If you specify a scale regression model for the severity distribution, then the scale parameter of the severity distribution is determined by the values of the severity regressors in the k th observation. (For more information about how PROC CCDM makes a random severity draw, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 224.)
3. Compute one sample point of different aggregate loss samples as follows:
 - For SIMULATIONMODE=CUSTOM, do not compute the unadjusted aggregate loss sample point. Because D must be nonzero for this mode, compute D sample points S^d of the D adjusted aggregate loss samples as follows:
 - a) Execute your severity adjustment program K times, once per observation. To execute the program for the k th observation, set the _FREQ_ and _SEV_ placeholder symbols equal to N_k and X_k , respectively. (For more information about the placeholder symbols, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 229.)
 - b) Retrieve the adjusted severity values X_k^d ($d = 1, \dots, D$) that your adjustment program computes.
 - c) After all K observations are processed, compute the sample point S^d of the d th adjusted aggregate loss sample as $S^d = \sum_{k=1}^K X_k^d$ ($d = 1, \dots, D$).
 - For SIMULATIONMODE=PUREPREMIUM, compute the sample point S of the unadjusted aggregate loss sample as $S = \sum_{k=1}^K N_k X_k$. If $D > 0$, then follow the same steps as the custom mode to compute D sample points S^d of the D adjusted aggregate loss samples, except for the following two differences:
 - Do not fill the _FREQ_ placeholder symbol, because it is valid only in the custom mode.

- Compute S^d as $S^d = \sum_{k=1}^K N_k X_k^d$ ($d = 1, \dots, D$).

If you specify the BY statement, then a separate sample of size M is created for each BY group in the DATA= data table.

Simulation with External Counts

If you specify externally simulated counts by using the EXTERNALCOUNTS statement, then each replication in the input data table represents the loss events that are generated by an entity. An entity can be an individual or organization for which you want to estimate the compound distribution. If an entity has any characteristics that are used as external factors (regressors) in developing the severity scale regression model, then you must specify the values of those factors in the DATA= data table. If you specify the ID= variable, then multiple observations for the same replication ID represent different entities in a group for which you are simulating the CDM.

This section describes the simulation process that PROC CCDM uses in the presence of externally simulated counts.

The process is described for one severity distribution. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution. Let D denote the number of adjusted severity symbols that you specify in the ADJUSTEDSEVERITY= option.

Suppose there are M distinct replications in the current BY group of the DATA= data table or in the entire DATA= data table if you do not specify the BY statement. A replication is identified either by the observation number or by the value of the ID= variable that you specify in the EXTERNALCOUNTS statement.

For each of the M values of the replication identifier, the following steps are executed R times, where R is the value of the NREPLICATES= option:

1. Read the value of the COUNT= variable, N_k , for each observation k ($k = 1, \dots, K$), where K denotes the number of observations in the current replication—that is, the number of observations in which the replication identifier has the same value.
2. For each observation k ($k = 1, \dots, K$), draw a severity value X_k from the severity model. If you specify a scale regression model for the severity distribution, then the scale parameter of the severity distribution is determined by the values of the severity regressors in the k th observation. (For more information about how PROC CCDM makes a random severity draw, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 224.)
3. Compute one sample point of different aggregate loss samples as follows:
 - For SIMULATIONMODE=CUSTOM, do not compute the unadjusted aggregate loss sample point. Because D must be nonzero for this mode, compute D sample points S^d of the D adjusted aggregate loss samples as follows:
 - a) Execute your severity adjustment program K times, once per observation. To execute the program for the k th observation, set the _FREQ_ and _SEV_ placeholder symbols equal to N_k and X_k , respectively. (For more information about the placeholder symbols, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 229.)
 - b) Retrieve the adjusted severity values X_k^d ($d = 1, \dots, D$) that your adjustment program computes.

- c) After all K observations are processed, compute the sample point S^d of the d th adjusted aggregate loss sample as $S^d = \sum_{k=1}^K X_k^d$ ($d = 1, \dots, D$).
- For SIMULATIONMODE=PUREPREMIUM, compute the sample point S of the unadjusted aggregate loss sample as $S = \sum_{k=1}^K N_k X_k$. If $D > 0$, then follow the same steps as the custom mode to compute D sample points S^d of the D adjusted aggregate loss samples, except for the following two differences:
 - Do not fill the _FREQ_ placeholder symbol, because it is valid only in the custom mode.
 - Compute S^d as $S^d = \sum_{k=1}^K N_k X_k^d$ ($d = 1, \dots, D$).

This process generates a compound distribution sample of size $M \times R$. If you specify the BY statement, then a separate sample of size $M \times R$ is created for each BY group in the DATA= data table.

Simulation of Adjusted Compound Distribution Sample

If you specify a list of D adjusted severity symbols in the ADJUSTEDSEVERITY= option and specify programming statements that compute those symbols, then a separate compound distribution sample is generated for each adjusted severity symbol. The set of all programming statements that you specify in the PROC CCDM step is called the severity adjustment program.

Formally, the severity adjustment program is expected to implement D adjustment functions for all D adjusted severity symbols. The adjustment function for the d th symbol is denoted by f^d ($d = 1, \dots, D$). The function f^d uses the unadjusted severity value, X_j along with other values, some of which depend on the simulation mode, to compute and return an adjusted severity value, X_j^d . The information that is available to compute X_j^d depends on the simulation mode as follows:

- For the collective risk simulation mode, if N denotes the number of loss events that are simulated for the current replication of the simulation process, then for the severity draw, X_j , of the j th loss event ($j = 1, \dots, N$), the adjusted severity value for the d th symbol is

$$X_j^d = f^d(X_j, S_{j-1}, S_{j-1}^d, \Psi_j)$$

where $S_{j-1} = \sum_{l=1}^{j-1} X_l$ is the aggregate unadjusted loss, $S_{j-1}^d = \sum_{l=1}^{j-1} X_l^d$ is the aggregate value of the d th adjusted severity symbol before the j th severity draw that generates X_j , and Ψ_j denotes the vector of values of the stochastic symbols that you specify in the SIMULATEDSYMBOL statement. The initial values of both S and S^d are set to 0—that is, $S_0 = 0$ and $S_0^d = 0$ ($d = 1, \dots, D$). Prior to executing your adjustment program, PROC CCDM makes a random draw for each stochastic symbol by using that symbol's probability distribution and parameter values. The aggregate adjusted loss for the replication is S_N^d , which is denoted by S^d for simplicity and is defined as

$$S^d = \sum_{j=1}^N X_j^d$$

- For pure premium and custom simulation modes, there is only one severity draw for every frequency draw, so the index j is always equal to 1.

- For pure premium mode, the adjusted severity value depends only on the unadjusted severity value and stochastic symbols as $X_1^d = f^d(X_1, \Psi_1)$ and the aggregate adjusted loss for the replication is $S^d = NX_1^d$.
- For custom simulation mode, you are free to combine the values of frequency, severity, and stochastic symbols in any manner to compute the adjusted severity value as $X_1^d = f^d(N, X_1, \Psi_1)$ and PROC CCDM sets the aggregate adjusted loss equal to the adjusted severity value as $S^d = X_1^d$.

In your severity adjustment program, you can use the following keywords as placeholder symbols for the input arguments of the function f^d :

_ADJSEVSUM d _

_CUMADJSEV d _

indicates the placeholder for S_{j-1}^d , the sum of adjusted severity values for the d th symbol that are computed by your program before X_j is generated and adjusted. PROC CCDM supplies this value to your program. For $d = 1$, you can use the symbol _ADJSEVSUM_ (alias _CUMADJSEV_).

This placeholder is always meaningful for **SIMULATIONMODE=COLLECTIVERISK**, but for **SIMULATIONMODE=PUREPREMIUM** and **SIMULATIONMODE=CUSTOM**, it is meaningful only for a multi-entity scenario, which is described in the section “[Aggregate Adjusted Loss Simulation for a Multi-entity Scenario](#)” on page 231.

FREQ

indicates the placeholder for frequency N . PROC CCDM supplies this value to your program.

You can use this placeholder only when you specify **SIMULATIONMODE=CUSTOM**.

SEV

indicates the placeholder for X_j , the unadjusted severity value. PROC CCDM generates this value as described in the section “[Simulation with No Regressors and No External Counts](#)” on page 218 (step 2) or the section “[Simulation with Regressors and No External Counts](#)” on page 218 (step 3). PROC CCDM supplies this value to your program.

You can use this placeholder for all simulation modes.

SEVSUM

CUMSEV

indicates the placeholder for S_{j-1} , the sum of unadjusted severity values that PROC CCDM generates before X_j is generated. PROC CCDM supplies this value to your program.

This placeholder is always meaningful for **SIMULATIONMODE=COLLECTIVERISK**, but for **SIMULATIONMODE=PUREPREMIUM** and **SIMULATIONMODE=CUSTOM**, it is meaningful only for a multi-entity scenario, which is described in the section “[Aggregate Adjusted Loss Simulation for a Multi-entity Scenario](#)” on page 231.

Your severity adjustment program must assign the output of the d th function f^d to the d th symbol that you specify in the **ADJUSTEDSEVERITY=** option in the PROC CCDM statement. PROC CCDM uses the final assigned value of the d th symbol as the value of X_j^d .

You can use most DATA step statements and functions in your program. The DATA step file and the data table I/O statements (for example, INPUT, FILE, SET, and MERGE) are not available. However, some

functionality of the PUT statement is supported. For more information, see the section “PROC FCMP and DATA Step Differences” in *Base SAS Procedures Guide*.

The simulation process that generates the aggregate adjusted loss sample is identical to the process that is described in the section “Simulation with Regressors and No External Counts” on page 218 or the section “Simulation with External Counts” on page 221, except that after making each of the N severity draws, PROC CCDM executes your severity adjustment program to compute the adjusted severity symbols (X_j^d). All the N adjusted severity values are added together to compute S^d , which forms one point of the d th aggregate adjusted loss sample. The process is illustrated using an example in the section “Illustration of the Aggregate Adjusted Loss Simulation Process” on page 233.

Using Severity Adjustment Variables

If you do not specify the DATA= data table, then your ability to adjust the severity value is limited, because you can use only the current severity draw, sums of unadjusted and adjusted severity draws that are made before the current draw, and some constant numbers to encode your adjustment policy. That is sufficient if you want to estimate the distribution of aggregate adjusted loss for only one entity. However, if you are simulating a scenario that contains more than one entity, then it might be more useful if the adjustment policy depends on factors that are specific to each entity that you are simulating. To do that, you must specify the DATA= data table and encode such factors as *adjustment variables* in the DATA= data table. If A denotes the set of values of the adjustment variables, then each adjustment function f^d accepts the set A as one of its inputs, in addition to the other input values that it accepts according to the simulation mode. PROC CCDM reads the values of adjustment variables from the DATA= data table and supplies the set of those values (A) to your severity adjustment program. For an invocation of each f^d with an unadjusted severity value of X_j , the values in set A are read from the same observation that is used to simulate X_j .

All adjustment variables that you use in your program must be present in the DATA= data table. You must not use any keyword for a placeholder symbol as a name of any variable in the DATA= data table, whether the variable is a severity adjustment variable or a regressor in the frequency or severity model. Further, the following restrictions apply to the adjustment variables in your severity adjustment program:

- You can use only numeric-valued variables. This restriction also implies that you cannot use SAS functions or call routines that require character-valued arguments, unless you pass those arguments as constant (literal) strings or characters.
- You cannot use functions that create lagged versions of a variable. If you need lagged versions, then you can use a DATA step before the PROC CCDM step to add those versions to the input data table.

The use of adjustment variables is illustrated using an example in the section “Illustration of the Aggregate Adjusted Loss Simulation Process” on page 233.

Aggregate Adjusted Loss Simulation for a Multi-entity Scenario

If you are simulating a scenario that consists of multiple entities, then you can use some additional pieces of information in your severity adjustment program. Let the scenario consist of K entities, and let N_k denote the number of loss events that are incurred by the k th entity ($k = 1, \dots, K$) in the current iteration of the simulation process. The simulation mode determines the processing of multiple entities as follows:

- For the collective risk simulation mode, each value of N_k is adjusted to conform to the upper limit that is imposed by the value of the MAXCOUNTDRAW= option. The total number of severity draws that

need to be made is $N = \sum_{k=1}^K N_k$. The d th aggregate adjusted loss is now defined as

$$S^d = \sum_{k=1}^K \sum_{j=1}^{N_k} X_{k,j}^d$$

where $X_{k,j}^d$ is an adjusted severity value of the d th symbol for the k th entity in the j th draw ($j = 1, \dots, N_k$). The form of the adjustment function f^d that computes $X_{k,j}^d$ is

$$X_{k,j}^d = f^d(X_{k,j}, S_{k,j-1}, S_{k,j-1}^d, S_{n-1}, S_{n-1}^d, \Psi_{k,j}, A_k)$$

where $X_{k,j}$ is the value of the j th draw of unadjusted severity for the k th entity; $S_{k,j-1} = \sum_{l=1}^{j-1} X_{k,l}$ and $S_{k,j-1}^d = \sum_{l=1}^{j-1} X_{k,l}^d$ are the aggregate unadjusted loss and the aggregate adjusted loss, respectively, for the k th entity before $X_{k,j}$ is generated; $\Psi_{k,j}$ denotes the vector of values of the j th draw of stochastic symbols that you specify in the **SIMULATEDSYMBOL** statement for the k th entity; and A_k is the set of values of the severity adjustment variables that PROC CCDM reads from the DATA= data table for the k th entity.

The index n ($n = 1, \dots, N$) keeps track of the total number of severity draws, across all entities, that are made before $X_{k,j}$ is generated. So $S_{n-1} = \sum_{m=1}^{k-1} \sum_{l=1}^{N_m} X_{m,l} + \sum_{l=1}^{j-1} X_{k,l}$ and $S_{n-1}^d = \sum_{m=1}^{k-1} \sum_{l=1}^{N_m} X_{m,l}^d + \sum_{l=1}^{j-1} X_{k,l}^d$ are the aggregate unadjusted loss and aggregate adjusted loss, respectively, for all the entities that are processed before $X_{k,j}$ is generated. Note that S_{n-1} and S_{n-1}^d include the $j - 1$ draws that are made for the k th entity before $X_{k,j}$ is generated.

The initial values of all types of aggregate losses are set to 0. In other words, $S_0 = 0$; for all values of d , $S_0^d = 0$; for all values of k , $S_{k,0} = 0$; and for all combinations of k and d , $S_{k,0}^d = 0$.

In your severity adjustment program, you can use the following two additional placeholder keywords:

SEVSUMFOROBS

CUMSEVFOROBS

indicates the placeholder for $S_{k,j-1}$, which is the total loss that the k th entity incurs before the current loss event. PROC CCDM supplies this value to your program.

ADJSEVSUMFOROBSd

CUMADJSEVFOROBSd

indicates the placeholder for $S_{k,j-1}^d$, which is the total adjusted loss for the d th symbol that the k th entity incurs before the current loss event. PROC CCDM supplies this value to your program. For $d = 1$, you can use the symbol **_ADJSEVSUMFOROBS_** (alias **_CUMADJSEVFOROBS_**).

The previously described placeholder symbols **_SEVSUM_** and **_ADJSEVSUM_** represent S_{n-1} and S_{n-1}^d , respectively. If you have only one entity in the scenario ($K = 1$), then the values of **_SEVSUMFOROBS_** and **_ADJSEVSUMFOROBSd_** are identical to the values of **_SEVSUM_** and **_ADJSEVSUMd_**, respectively.

- For pure premium and custom simulation modes, there is one frequency draw per entity and only one severity draw for every frequency draw, so the index j is always equal to 1. PROC CCDM draws the frequency value N_k for each entity k and uses it according to the simulation mode as follows:

- For the pure premium simulation mode, the form of the adjustment function f^d that computes $X_{k,1}^d$ is

$$X_{k,1}^d = f^d(X_{k,1}, S_{n-1}, S_{n-1}^d, \Psi_{k,1}, A_k)$$

and PROC CCDDM computes the aggregate adjusted loss as

$$S^d = \sum_{k=1}^K N_k X_{k,1}^d$$

- For the custom simulation mode, you are free to combine the values of frequency, severity, stochastic symbols, and the adjustment variables in any manner to compute the adjusted severity value $X_{k,1}^d$. The form of the adjustment function f^d that computes $X_{k,1}^d$ is

$$X_{k,1}^d = f^d(N_k, X_{k,1}, S_{n-1}, S_{n-1}^d, \Psi_{k,1}, A_k)$$

and PROC CCDDM computes the aggregate adjusted loss as

$$S^d = \sum_{k=1}^K X_{k,1}^d$$

For both the pure premium and custom simulation modes, your adjustment program can use the previously described placeholder symbols `_SEVSUM_` and `_ADJSEVSUM_` to represent S_{n-1} and S_{n-1}^d , respectively.

Your severity adjustment program must implement all D adjustment functions and assign the output of the d th function f^d to the d th adjusted severity symbol that you specify in the `ADJUSTEDSEVERITY=` option. PROC CCDDM executes the severity adjustment program for each j th severity value that it draws for each k th entity, and it uses the final value of the d th symbol as the value of $X_{k,j}^d$.

There is one caveat when a scenario consists of more than one entity ($K > 1$) and when you use any of the symbols for cumulative severity values (`_SEVSUM_`, `_ADJSEVSUMd_`, `_SEVSUMFOROBS_`, or `_ADJSEVSUMFOROBSd_`) in your severity adjustment program. In this case, to make the simulation realistic, it is important to randomize the order of N severity draws across K entities. For more information, see the section “Randomizing the Order of Severity Draws across Observations of a Scenario” on page 236.

Illustration of the Aggregate Adjusted Loss Simulation Process

This section continues the example in the section “Simulation with Regressors and No External Counts” on page 218 to illustrate the simulation of aggregate adjusted loss for the collective risk simulation mode.

Recall that the earlier example simulates a scenario that consists of five policyholders. Assume that you want to compute the distribution of the aggregate amount paid to all the policyholders in a year, where the payment for each loss is determined by a deductible and a per-payment limit. To begin with, you must record the deductible and limit information in the input `DATA=` data table. The following table shows the `DATA=` data table from the earlier example, extended to include two variables, `Deductible` and `Limit`:

Obs	age	gender	carType	deductible	limit
1	30	2	1	250	5000
2	25	1	2	500	3000
3	45	2	2	100	2000
4	33	1	1	500	5000
5	50	1	1	200	2000

The variables Deductible and Limit are referred to as severity adjustment variables, because you need to use them to compute the adjusted severity. Let AmountPaid represent the value of adjusted severity that you are interested in. Further, let the following SAS programming statements encode your logic of computing the value of AmountPaid:

```
amountPaid = MAX(_sev_ - deductible, 0);
amountPaid = MIN(amountPaid, MAX(limit - _adjsevsumforobs_, 0));
```

PROC CCDM supplies your program with values of the placeholder symbols `_SEV_` and `_ADJSEVSUMFOROBS_`, which represent the value of the current unadjusted severity draw and the sum of adjusted severity values from the previous draws, respectively, for the observation that is being processed. The use of `_ADJSEVSUMFOROBS_` helps you ensure that the payment that is made to a particular policyholder in a year does not exceed the limit that is recorded in the variable Limit.

To simulate a sample for the aggregate of AmountPaid, you need to submit a PROC CCDM step whose structure is like the following:

```
proc ccdm data=<data table name> adjustedseverity=amountPaid
    severityest=<severity parameter estimates data table>
    countstore=<count model store>;
    severitymodel <severity distribution name(s)>;

    amountPaid = MAX(_sev_ - deductible, 0);
    amountPaid = MIN(amountPaid, MAX(limit - _adjsevsumforobs_, 0));
run;
```

The simulation process of one replication that generates one point of the aggregate loss sample and the corresponding point of the aggregate adjusted loss sample is as follows:

1. Use the values Age=30, Gender=2, and CarType=1 in the first observation to draw a count from the count distribution. Let that count be 3. Repeat the process for the remaining four observations. Let the counts be as shown in the Count column in the following table:

Obs	age	gender	carType	deductible	limit	count
1	30	2	1	250	5000	2
2	25	1	2	500	3000	1
3	45	2	2	100	2000	2
4	33	1	1	500	5000	3
5	50	1	1	200	2000	0

The Count column is shown for illustration only; it is not added as a variable to the DATA= data table.

2. The simulated counts from all the observations are added together to get a value of $N = 8$. This means that for this particular replication, you expect a total of eight loss events in a year from these five policyholders.

3. For the first observation, the scale parameter of the severity distribution is computed by using the values Age=30, Gender=2, and CarType=1. That value of the scale parameter is used together with estimates of the other parameters from the SEVERITYEST= data table to make two draws from the severity distribution. The process is repeated for the remaining four policyholders. The fifth policyholder does not generate any loss event for this particular replication, so no severity draws are made by using the fifth observation. Let the severity draws, rounded to integers for convenience, be as shown in the `_SEV_` column in the following table:

Obs	age	gender	carType	deductible	limit	count	<code>_sev_</code>		
1	30	2	1	250	5000	2	350	2100	
2	25	1	2	500	3000	1	4500		
3	45	2	2	100	2000	2	700	4300	
4	33	1	1	200	5000	3	600	1500	950
5	50	1	1	200	2000	0			

The `_SEV_` column is shown for illustration only; it is not added as a variable to the DATA= data table. The sample point for the aggregate unadjusted loss is computed by adding together the severity values of eight draws, which gives an aggregate loss value of 15,000. The aggregate unadjusted loss is also referred to as the ground-up loss.

For each severity draw, your severity adjustment program is executed to compute the adjusted severity, which is the value of AmountPaid in this case. For the draws in the preceding table, the values of AmountPaid are as follows:

Obs	deductible	limit	<code>_sev_</code>	<code>_adjsevsumforobs_</code>	amountPaid
1	250	5000	350	0	100
1	250	5000	2100	100	1850
2	500	3000	4500	0	3000
3	100	2000	700	0	600
3	100	2000	4300	600	1400
4	200	5000	600	0	400
4	200	5000	1500	400	1300
4	200	5000	950	1700	750

The adjusted severity values are added together to compute the cumulative payment value of 9,400, which forms the first sample point for the aggregate adjusted loss.

After recording the aggregate unadjusted and aggregate adjusted loss values in their respective samples, the process returns to step 1 to compute the next sample point unless the specified number of sample points have been simulated.

In this particular example, you can verify that the order in which the eight loss events are simulated does not affect the aggregate adjusted loss. As a simple example, consider the following order of draws, which is different from the consecutive order that the preceding table uses:

Obs	deductible	limit	<code>_sev_</code>	<code>_adjsevsumforobs_</code>	amountPaid
4	200	5000	600	0	400
3	100	2000	4300	0	2000
1	250	5000	350	0	100
3	100	2000	700	2000	0
4	200	5000	950	400	750

1	250	5000	2100	100	1850
2	500	3000	4500	0	3000
4	200	5000	1500	1150	1300

Although the payments that are made for individual loss events differ, the aggregate adjusted loss is still 9,400.

However, in general, when you use a cumulative severity value such as `_ADJSEVSUMFOROBS_` in your program, the order in which the draws are processed affects the final value of aggregate adjusted loss. For more information, see the sections “[Randomizing the Order of Severity Draws across Observations of a Scenario](#)” on page 236 and “[Illustration of the Need to Randomize the Order of Severity Draws](#)” on page 237.

Randomizing the Order of Severity Draws across Observations of a Scenario

If you specify a scenario that consists of more than one entity, then it is assumed that each entity generates its loss events independently from the other entities. In other words, the time at which the loss event of one entity is generated or recorded is independent of the time at which the loss event of another entity is generated or recorded. To honor this assumption of independence among entities, the order in which those entities are processed must be randomized across K entities such that no entity is preferred over another. The K entities are represented by K observations of the scenario in the `DATA=` data table. If you specify external counts, the K observations correspond to the observations that have the same replication identifier value. If you do not specify the external counts, then the K observations correspond to all the observations in the `BY` group, or to all the observations in the entire `DATA=` data table if you do not specify the `BY` statement.

The randomization process depends on the simulation mode:

- For the collective risk simulation mode, if entity k generates N_k loss events, where N_k is adjusted to conform to the upper limit that is imposed by the value of the `MAXCOUNTDRAW=` option, then the total number of loss events for a group of K entities is $N = \sum_{k=1}^K N_k$. To simulate the aggregate loss for this group, N severity draws are made and aggregated to compute one point of the compound distribution sample. To honor the assumption of independence among entities, the order of those N severity draws must be randomized across K entities such that no entity is preferred over another.

PROC CCDM implements the randomization process over N severity draws as follows: First, it chooses one of the K observations at random, say the k th observation, and draws one severity value ($X_{k,j}$) from the severity distribution that is implied by that observation. It then supplies the severity value, along with the values of the severity adjustment variables (A_k) and random draws of simulated symbols ($\Psi_{k,j}$), to your severity adjustment program to compute the adjusted severity values ($X_{k,j}^d$) and updates the respective aggregate adjusted losses (S^d). Next, it chooses another observation at random, draws one severity value from its implied severity distribution, and repeats the analysis of computing $X_{k,j}^d$ and updating S^d . In each step, PROC CCDM increments by 1 the total number of events that are simulated for the selected observation k . When it simulates all N_k events for an observation k , it retires that observation and continues the process with the remaining observations until it makes a total of N severity draws.

- For pure premium and custom simulation modes, there is one frequency and severity draw per entity. To honor the assumption of independence among entities, PROC CCDM randomizes the order in which it processes the entities as follows: First, it chooses one of the K observations at random, say the k th observation. It draws one frequency value and one severity value from the count and severity

distributions, respectively, that are implied by the k th observation. It then supplies those frequency (N_k) and severity ($X_{k,1}$) values, along with the values of the severity adjustment variables (A_k) and random draws of simulated symbols ($\Psi_{k,1}$), to your severity adjustment program to compute the adjusted severity values ($X_{k,1}^d$) and updates the respective aggregate adjusted losses (S^d). Next, PROC CCDM retires the processed observation, chooses another observation at random from the remaining $K - 1$ observations, and repeats the analysis of computing $X_{k,1}^d$ and updating S^d . The process continues until all K observations are exhausted.

The randomization process is especially important in the context of simulating an adjusted compound distribution sample when your severity adjustment program uses the aggregate adjusted severity to adjust the next severity value. For an illustration of the need to randomize in such cases for the collective risk simulation mode, see the next section.

Illustration of the Need to Randomize the Order of Severity Draws

This section uses the example in the section “[Illustration of the Aggregate Adjusted Loss Simulation Process](#)” on page 233, but with the following PROC CCDM step:

```
proc ccdm data=<data table name> adjustedseverity=amountPaid
    severityest=<severity parameter estimates data table>
    countstore=<count model store>;
    severitymodel <severity distribution name(s)>;

    if (_adjsevsum_ > 15000) then
        amountPaid = 0;
    else do;
        penaltyFactor = MIN(3, 15000/(15000 - _adjsevsum_));
        amountPaid = MAX(0, _sev_ - deductible * penaltyFactor);
    end;
run;
```

The severity adjustment statements in the preceding code compute the value of AmountPaid by using the collective risk simulation mode and the following provisions in the insurance policy:

- There is a limit of 15,000 on the total amount that can be paid in a year to the group of policyholders that is being simulated. The amount of payment for each loss event depends on the total amount of payments before that loss event.
- The penalty for incurring more losses is imposed in the form of an increased deductible. In particular, the deductible is increased by the ratio of the maximum cumulative payment (15,000) to the amount that remains available to pay for future losses in the year. The factor by which the deductible can be raised has a limit of three.

This example illustrates only step 3 of the simulation process, where randomization is done. It assumes that step 2 of the simulation process is identical to step 2 of the example in the section “[Illustration of the Aggregate Adjusted Loss Simulation Process](#)” on page 233. At the beginning of step 3, let the severity draws from all the observations be as shown in the `_SEV_` column in the following table:

Obs	age	gender	carType	deductible	count	_sev_		
1	30	2	1	250	2	350	2100	
2	25	1	2	500	1	4500		
3	45	2	2	100	2	700	4300	
4	33	1	1	200	3	600	1500	950
5	50	1	1	200	0			

If the order of these eight draws is not randomized, then all the severity draws for the first observation are adjusted before all the severity draws of the second observation, and so on. The execution of the severity adjustment program leads to the following sequence of values for AmountPaid:

Obs	deductible	_sev_	_adjsevsum_	penaltyFactor	amountPaid
1	250	350	0	1	100
1	250	2100	100	1.0067	1848.32
2	500	4500	1948.32	1.1493	3925.36
3	100	700	5873.68	1.6436	535.64
3	100	4300	6409.32	1.7461	4125.39
4	200	600	10534.72	3	0
4	200	1500	10534.72	3	900
4	200	950	11434.72	3	350

The preceding sequence of simulating loss events results in a cumulative payment of 11,784.72.

If the sequence of draws is randomized over observations, then the computation of the cumulative payment might proceed as follows for one instance of randomization:

Obs	deductible	_sev_	_adjsevsum_	penaltyFactor	amountPaid
2	500	4500	0	1	4000
1	250	350	4000	1.3636	9.09
3	100	700	4009.09	1.3648	563.52
4	200	950	4572.61	1.4385	662.30
4	200	1500	5234.91	1.5361	1192.78
1	250	2100	6427.69	1.7498	1662.54
4	200	600	8090.24	2.1708	165.83
3	100	4300	8256.07	2.2242	4077.58

In this example, a policyholder is identified by the value in the Obs column. As the table indicates, PROC CCDM randomizes the order of loss events not only across policyholders but also across the loss events that a given policyholder incurs. The particular sequence of loss events that is shown in the table results in a cumulative payment of 12,333.65. This differs from the cumulative payment that results from the previously considered nonrandomized sequence of loss events, which tends to penalize the fourth policyholder by always processing her payments after all other payments, with a possibility of underestimating the total amount paid. This comparison not only illustrates that the order of randomization affects the aggregate adjusted loss sample but also corroborates the arguments about the importance of order randomization that are made at the beginning of the section “[Randomizing the Order of Severity Draws across Observations of a Scenario](#)” on page 236.

Parameter Perturbation Analysis

It is important to realize that most of the parameters of the frequency and severity models are estimated and that there is uncertainty associated with the parameter estimates. Any compound distribution estimate that you compute by using these uncertain parameter estimates is inherently uncertain. The aggregate loss sample that you simulate by using the mean estimates of the parameters is just one possible sample from the compound distribution. If information about the uncertainty in parameter estimates is available, then it is recommended that you perform parameter perturbation analysis that generates multiple samples of the compound distribution, in which each sample is simulated by using a set of perturbed parameter estimates. You can use the **NPERTURBEDSAMPLES=** option in the PROC CCDDM statement to specify the number of perturbed samples to be generated. PROC CCDDM creates the set of perturbed parameter estimates by making a random draw of the parameter values from their joint probability distribution. If you specify **NPERTURBEDSAMPLES=P**, then PROC CCDDM creates P sets of perturbed parameters, and each set is used to simulate a full aggregate sample. The summary analysis of P such aggregate loss samples results in a set of P estimates for each summary statistic and percentile of the compound distribution. The mean and standard deviation of this set of P estimates quantify the uncertainty that is associated with the compound distribution.

The parameter estimate uncertainty information is available in the form of either the variance-covariance matrix of the parameter estimates or standard errors of the parameter estimates. If the variance-covariance matrix is available and is positive definite, then PROC CCDDM assumes that the joint probability distribution of the parameter estimates is a multivariate normal distribution, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where the mean vector $\boldsymbol{\mu}$ is the set of point parameter estimates and $\boldsymbol{\Sigma}$ is the variance-covariance matrix. If the variance-covariance matrix is not available or is not positive definite, or if you specify the **IGNORECOV** option, then PROC CCDDM assumes that each parameter has a univariate normal distribution, $\mathcal{N}(\mu, \sigma^2)$, where μ is the point estimate of the parameter and σ is the standard error of the parameter estimate.

Specifying Parameter Uncertainty

If you specify the severity models by using the SEVERITYEST= data table, then the point parameter estimates are expected to be available in the SEVERITYEST= data table in observations for which **_TYPE_='EST'**, the standard errors are expected to be available in the SEVERITYEST= data table in observations for which **_TYPE_='STDERR'**, and the variance-covariance matrix is expected to be available in the SEVERITYEST= data table in observations for which **_TYPE_='COV'**. When you use the SEVSELECT procedure to create the SEVERITYEST= data table, you need to specify the COVOUT option in the PROC SEVSELECT statement to make the variance-covariance estimates available in the SEVERITYEST= data table.

If you specify the severity models by using the SEVERITYSTORE= item store, then you need to specify the OUTSTORE= option in the PROC SEVSELECT statement to create that item store, which includes the point parameter estimates and standard errors by default. In addition, you need to specify the COVOUT option in the PROC SEVSELECT statement to make the variance-covariance estimates available in the SEVERITYSTORE= item store.

If you specify the frequency model by using the COUNTSTORE= item store, then the uncertainty estimates are available in that item store, because the CNTSELECT procedure always writes the point estimates, standard errors, and variance-covariance matrix of the parameters to the item store.

If you specify the frequency model by using the **COUNTMODEL** statement, then it is recommended that you use the **STDERROR=** option to specify the standard error for each parameter.

If you specify an external frequency model by using the EXTERNALCOUNTS statement, then there is no way to specify the uncertainty in the frequency model, so PROC CCDM conducts the perturbation analysis by perturbing only the parameters of the severity model.

Output of Perturbation Analysis

PROC CCDM displays the effect of parameter perturbation on the summary statistics and percentiles of the compound distribution in the ODS tables PerturbedSummary and PerturbedPctlSummary, respectively. For each statistic, the tables report the mean and standard error that PROC CCDM computes by aggregating the statistics from all P perturbed samples. For more information, see the section “[Displayed Output](#)” on page 250.

You can specify the OUTSUM statement to write the summary statistics and percentiles of individual perturbed samples to an output data table. You can also specify the PERTURBOUT option in the OUTPUT statement to write all perturbed samples in their entirety to an output data table.

If you specify the ADJUSTEDSEVERITY= option in the PROC CCDM statement, then PROC CCDM performs a separate perturbation analysis for the distribution of each aggregate adjusted loss.

Fixed Parameter-Value Distributions

Table 7.2 and Table 7.3 define the continuous and discrete probability distributions, respectively, that you can use in the SIMULATEDSYMBOL and COUNTMODEL statements. Note the following points when reading both tables:

- You can use any one of the names in the Names column to specify the distribution that the Distribution column identifies. The names are not case-sensitive.
- When you specify the parameter values for a distribution in the COUNTMODEL or SIMULATEDSYMBOL statement, the values must appear in the order in which the parameters are listed in the Parameters column. For example, specifying burr(3, 0.5, 1) means you are specifying $\theta = 3$, $\alpha = 0.5$, and $\gamma = 1$; similarly, specifying negbin(0.3, 5) means you are specifying $p = 0.3$ and $n = 5$.

The following points pertain to the continuous distributions in Table 7.2:

- You can use a continuous distribution in the COUNTMODEL statement only if you use the SIMULATIONMODE= option to specify a simulation mode other than the collective risk mode.
- The definitions of probability density function (PDF) and cumulative distribution functions (CDF) use the following mathematical notation:
 - x denotes the value of the continuous random variable in Table 7.2.
 - $z = x/\theta$, wherever z is used.
 - θ denotes the scale parameter for the continuous distributions.
 - μ denotes the mean parameter, except for the lognormal distribution, where μ is the log-scale parameter ($\mu = \log(\theta)$).
 - $\gamma(a, b) = \int_0^b t^{a-1} e^{-t} dt$ is the lower incomplete gamma function.

– $\Phi(y) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{y}{\sqrt{2}} \right) \right)$ is the standard normal CDF.

- Parameters of the inverse Gaussian (Wald) distribution are related to the parameters of the scaled inverse Gaussian distribution as $\mu = \theta$ and $\lambda = \theta\alpha$.
- The parameters of the Tweedie and scaled Tweedie distribution are related to each other as described in the section “[Tweedie Distributions](#)” on page 1781 in Chapter 26, “[SEVSELECT Procedure](#).” PROC SEVSELECT uses the same parameterizations.
- The parameters of the Wald and scaled Wald distributions are related to each other as $\theta = \mu$ and $\alpha = \lambda/\mu$. PROC SEVSELECT uses the scaled Wald parameterization because it has a scale parameter.

Table 7.2 Fixed Parameter-Value Continuous Distributions

Distribution	Names	Parameters	PDF (f) and CDF (F)
Burr’s Type XII	BurrTypeXII	$\theta > 0$	$f(x) = \frac{\alpha\gamma z^\gamma}{x(1+z^\gamma)^{(\alpha+1)}}$
	Burr	$\alpha > 0$	$F(x) = 1 - \left(\frac{1}{1+z^\gamma} \right)^\alpha$
	Bur	$\gamma > 0$	
Exponential	Exponential	$\theta > 0$	$f(x) = \frac{1}{\theta} e^{-x/\theta}$
	Expo		$F(x) = 1 - e^{-x/\theta}$
	Exp		
Gamma	Gamma	$\theta > 0$	$f(x) = \frac{z^\alpha e^{-z}}{x\Gamma(\alpha)}$
	Gamm	$\alpha > 0$	$F(x) = \frac{\gamma(\alpha, z)}{\Gamma(\alpha)}$
	Gam		
Generalized Pareto	GeneralizedPareto	$\theta > 0$	$f(x) = \frac{1}{\theta} (1 + \xi z)^{-1-1/\xi}$
	GPD	$\xi > 0$	$F(x) = 1 - (1 + \xi z)^{-1/\xi}$
	GPare		
Inverse Gaussian (Wald)	Wald	$\mu > 0$	$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} e^{-\frac{\lambda(x-\mu)^2}{2\mu^2 x}}$
	IGauss	$\lambda > 0$	$F(x) = \Phi\left(\left(\frac{x}{\mu} - 1\right)\sqrt{\frac{\lambda}{x}}\right) +$
	IGau		$\Phi\left(-\left(\frac{x}{\mu} + 1\right)\sqrt{\frac{\lambda}{x}}\right) e^{2\lambda/\mu}$
Lognormal	Lognormal	μ	$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$
	Logn	$\sigma > 0$	$F(x) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right)$
	Lgn		
Normal	Normal	μ	$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$
	Gauss	$\sigma > 0$	$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$
	Norm		
Pareto (Type I)	ParetoTypeI	$\theta > 0$	$f(x) = \frac{\alpha}{\theta} \left(\frac{1}{x}\right)^{\alpha+1}$ if $x \geq \theta$; 0 otherwise
	Pareto1	$\alpha > 0$	$F(x) = 1 - \left(\frac{\theta}{x}\right)^\alpha$
	Pare		

Table 7.2 *continued*

Distribution	Names	Parameters	PDF (f) and CDF (F)
Pareto (Type II)	ParetoTypeII	$\theta > 0$	$f(x) = \frac{\alpha\theta^\alpha}{(x+\theta)^{\alpha+1}}$
	Pareto2	$\alpha > 0$	$F(x) = 1 - \left(\frac{\theta}{x+\theta}\right)^\alpha$
	Par2		
Scaled inverse	ScaledWald	$\theta > 0$	$f(x) = \frac{1}{\theta} \sqrt{\frac{\alpha}{2\pi z^3}} e^{-\frac{\alpha(z-1)^2}{2z}}$
Gaussian (Wald)	SWald	$\alpha > 0$	$F(x) = \Phi\left((z-1)\sqrt{\frac{\alpha}{z}}\right) +$
	SIGau		$\Phi\left(-(z+1)\sqrt{\frac{\alpha}{z}}\right) e^{2\alpha}$
Scaled Tweedie	ScaledTweedie	$\theta > 0$	$f(x) = a(x, \theta, \lambda, p) \exp\left(-\frac{x}{\theta} - \lambda\right)$
	STweedie	$\lambda > 0$	$F(x) = \int_0^x f(t)dt$
	STwdy	$1 < p < 2$	
Tweedie	Tweedie	$p > 1$	$f(x) = a(x, \phi) \exp\left[\frac{1}{\phi} \left(\frac{x\mu^{1-p}}{1-p} - \kappa(\mu, p)\right)\right]$
	Twee	$\mu > 0$	$F(x) = \int_0^x f(t)dt$
	Twdy	$\phi > 0$	
Uniform	Uniform	l	$f(x) = \frac{1}{r-l}$ if $l \leq x \leq r$; 0 otherwise
	Unif	r	$F(x) = \frac{x-l}{r-l}$ if $l \leq x < r$;
	Uni		1 if $x \geq r$; 0 otherwise
Weibull	Weibull	$\theta > 0$	$f(x) = \frac{\tau}{\theta} z^{\tau-1} e^{-z^\tau}$
	Weib	$\tau > 0$	$F(x) = 1 - e^{-z^\tau}$
	Wbl		

The following points pertain to the discrete distributions in [Table 7.3](#):

- The definition of the probability mass functions (PMF) uses m to denote the value of the discrete random variable. The table does not show an explicit expression for the CDF because the CDF of each distribution is defined in the standard manner as $F(m) = \sum_{i=0}^m f(i)$.
- PROC CCDM supports three parameterizations of the negative binomial distribution to offer flexibility of specification. The standard parameterization (Negbin) is the same as the one you specify in the SAS PDF and CDF functions. The other two are inspired by the parameterizations that the CNTSELECT procedure supports:
 - The NegbinP1 parameterization allows the variance to be linear in mean. NegbinP1(μ, α) is equivalent to Negbin($p = 1/(1 + \alpha)$, $n = \mu/\alpha$).
 - The NegbinP2 parameterization allows the variance to be quadratic in mean. NegbinP2(μ, α) is equivalent to Negbin($p = 1/(1 + \mu\alpha)$, $n = 1/\alpha$).
- The logarithmic versions of the discrete distributions correspond to the distributions that the CNTSELECT procedure uses for the count regression models. The relationship of the parameters of the logarithmic versions and their nonlogarithmic counterparts is as follows:

- $\text{LogCMPoissonLambda}(\beta, \delta) = \text{CMPoisson}(\lambda = e^\beta, \nu = e^{-\delta})$. This distribution corresponds to the parameterization of the Conway-Maxwell-Poisson distribution that PROC CNTSELECT uses when you specify the `DIST=CMPOISSON` and `PARAMETER=LAMBDA` options in the `MODEL` statement of PROC CNTSELECT. The δ parameter corresponds to the `_lnNu` parameter that PROC CNTSELECT reports.
- $\text{LogCMPoissonMu}(\beta, \delta) = \text{CMPoisson}(\lambda = e^{\beta e^{-\delta}}, \nu = e^{-\delta})$. This distribution corresponds to the default parameterization of the Conway-Maxwell-Poisson distribution that PROC CNTSELECT uses, which is equivalent to specifying the `DIST=CMPOISSON` and `PARAMETER=MU` options in the `MODEL` statement of PROC CNTSELECT. The δ parameter corresponds to the `_lnNu` parameter that PROC CNTSELECT reports.
- $\text{LogNegbinP1}(\beta, \alpha) = \text{NegbinP1}(\mu = e^\beta, \alpha)$. This distribution corresponds to the parameterization of the negative binomial distribution that PROC CNTSELECT uses when you specify the `DIST=NEGBIN(P=1)` option in the `MODEL` statement of PROC CNTSELECT.
- $\text{LogNegbinP2}(\beta, \alpha) = \text{NegbinP2}(\mu = e^\beta, \alpha)$. This distribution corresponds to the parameterization of the default negative binomial distribution that PROC CNTSELECT uses when you specify the `DIST=NEGBIN` option in the `MODEL` statement of PROC CNTSELECT.
- $\text{LogPoisson}(\beta) = \text{Poisson}(\lambda = e^\beta)$. This distribution corresponds to the parameterization of the Poisson distribution that PROC CNTSELECT uses when you specify the `DIST=POISSON` option in the `MODEL` statement of PROC CNTSELECT.

Table 7.3 Fixed Parameter-Value Discrete Distributions

Distribution	Names	Parameters	Probability Mass Function (f)
Conway-Maxwell-Poisson (Standard)	CMPOisson	$\lambda > 0$	$f(m) = \frac{\lambda^m}{(m!)^\nu} \frac{1}{Z(\lambda, \nu)}$
	ConMaxPoi CMP	$\nu \geq 0$	where $Z(\lambda, \nu) = \sum_{j=0}^{\infty} \frac{\lambda^j}{(j!)^\nu}$
Log Conway-Maxwell-Poisson (λ -parameterization)	LogCMPoissonLambda	β	$f(m) = \frac{e^{\beta m}}{(m!)^{e^{-\delta}}} \frac{1}{Z(e^\beta, e^{-\delta})}$
	LogCMPL LCMPL	δ	where $Z(e^\beta, e^{-\delta}) = \sum_{j=0}^{\infty} \frac{e^{\beta j}}{(j!)^{e^{-\delta}}}$
Log Conway-Maxwell-Poisson (μ -parameterization)	LogCMPoissonMu	β	$f(m) = \frac{e^{\beta m e^{-\delta}}}{(m!)^{e^{-\delta}}} \frac{1}{Z(e^{\beta e^{-\delta}}, e^{-\delta})}$
	LogCMP LCMP	δ	where $Z(e^{\beta e^{-\delta}}, e^{-\delta}) = \sum_{j=0}^{\infty} \frac{e^{\beta j e^{-\delta}}}{(j!)^{e^{-\delta}}}$
Log negative binomial (variance = $e^\beta + \alpha e^\beta$)	LogNegbinP1	β	$f(m) = \frac{\Gamma(m+\alpha^{-1}e^\beta)}{m!\Gamma(\alpha^{-1}e^\beta)} (1-q)^{\alpha^{-1}e^\beta} q^m$
	LogNB1 LNB1	$\alpha > 0$	where $q = \frac{1}{\alpha^{-1}+1}$
Log negative binomial (variance = $e^\beta + \alpha e^{2\beta}$)	LogNegbinP2	β	$f(m) = \frac{\Gamma(m+\alpha^{-1})}{m!\Gamma(\alpha^{-1})} (1-q)^{\alpha^{-1}} q^m$
	LogNB2 LNB2	$\alpha > 0$	where $q = \frac{e^\beta}{\alpha^{-1}+e^\beta}$

Table 7.3 *continued*

Distribution	Names	Parameters	Probability Mass Function (f)
Log Poisson	LogPoisson LogPoi LPoi	β	$f(m) = e^{-e^\beta} \left(\frac{e^{\beta m}}{m!} \right)$
Negative binomial (standard)	NegativeBinomial Negbin Negb	$0 \leq p \leq 1$ $n > 0$	$f(m) = \binom{n+m-1}{n-1} p^n (1-p)^m$
Negative binomial (variance = $\mu + \alpha\mu$)	NegbinP1 Negb1 NB1	$\mu > 0$ $\alpha > 0$	$f(m) = \frac{\Gamma(m+\alpha^{-1}\mu)}{m!\Gamma(\alpha^{-1}\mu)} (1-q)^{\alpha^{-1}\mu} q^m$ where $q = \frac{1}{\alpha^{-1}+1}$
Negative binomial (variance = $\mu + \alpha\mu^2$)	NegbinP2 Negb2 NB2	$\mu > 0$ $\alpha > 0$	$f(m) = \frac{\Gamma(m+\alpha^{-1})}{m!\Gamma(\alpha^{-1})} (1-q)^{\alpha^{-1}} q^m$ where $q = \frac{\mu}{\alpha^{-1}+\mu}$
Poisson	Poisson Pois Poi	$\lambda > 0$	$f(m) = e^{-\lambda} \frac{\lambda^m}{m!}$

Descriptive Statistics

This section provides computational details for the descriptive statistics that are computed for each aggregate loss sample. You can also save these statistics in an OUTSUM= data table by specifying appropriate keywords in the OUTSUM statement.

This section gives specific details about the moment statistics. For more information about the methods of computing percentile statistics, see the description of the PCTLDEF= option in the UNIVARIATE procedure in the *Base SAS Procedures Guide: Statistical Procedures*.

Standard algorithms (Fisher 1973) are used to compute the moment statistics. The computational methods that the CCDM procedure uses are consistent with those that other SAS procedures use for calculating descriptive statistics.

Mean

The sample mean is calculated as

$$\bar{y} = \frac{\sum_{i=1}^n y_i}{n}$$

where n is the size of the generated aggregate loss sample and y_i is the i th value of the aggregate loss.

Standard Deviation

The standard deviation is calculated as

$$s = \sqrt{\frac{1}{d} \sum_{i=1}^n (y_i - \bar{y})^2}$$

where n is the size of the generated aggregate loss sample, y_i is the i th value of the aggregate loss, $\bar{y}$ is the sample mean, and d is the divisor controlled by the **VARDEF=** option in the PROC CCDDM statement:

$$d = \begin{cases} n - 1 & \text{if VARDEF=DF (default)} \\ n & \text{if VARDEF=N} \end{cases}$$

Skewness

The sample skewness, which measures the tendency of the deviations to be larger in one direction than in the other, is calculated as

$$\frac{1}{d_s} \sum_{i=1}^n \left(\frac{y_i - \bar{y}}{s} \right)^3$$

where n is the size of the generated aggregate loss sample, y_i is the i th value of the aggregate loss, $\bar{y}$ is the sample mean, s is the sample standard deviation, and d_s is the divisor controlled by the **VARDEF=** option in the PROC CCDDM statement:

$$d_s = \begin{cases} \frac{(n-1)(n-2)}{n} & \text{if VARDEF=DF (default)} \\ n & \text{if VARDEF=N} \end{cases}$$

If VARDEF=DF, then n must be greater than 2.

The sample skewness can be positive or negative; it measures the asymmetry of the data distribution and estimates the theoretical skewness $\sqrt{\beta_1} = \mu_3 \mu_2^{-\frac{3}{2}}$, where μ_2 and μ_3 are the second and third central moments. Observations that are normally distributed should have a skewness near zero.

Kurtosis

The sample kurtosis, which measures the heaviness of tails, is calculated as in Table 7.4, depending on the value that you specify in the **VARDEF=** option.

Table 7.4 Formulas for Kurtosis

VARDEF= Value	Formula
DF (default)	$\frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^n \left(\frac{y_i - \bar{y}}{s} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$
N	$\frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \bar{y}}{s} \right)^4 - 3$

In these formulas, n is the size of the generated aggregate loss sample, y_i is the i th value of the aggregate loss, $\bar{y}$ is the sample mean, and s is the sample standard deviation. If VARDEF=DF, then n must be greater than 3.

The sample kurtosis measures the heaviness of the tails of the data distribution. It estimates the adjusted theoretical kurtosis denoted as $\beta_2 - 3$, where $\beta_2 = \frac{\mu_4}{\mu_2^2}$ and μ_4 is the fourth central moment. Observations that are normally distributed should have a kurtosis near zero.

Input Specification

PROC CCDM accepts the COUNTSTORE= item store, DATA= data table, SEVERITYEST= data table, and SEVERITYSTORE= item store as input. This section describes the information that they are expected to contain.

COUNTSTORE= Item Store

The COUNTSTORE= item store is expected to be created by using the STORE= option in the CNTSELECT procedure. You must specify the COUNTSTORE= item store when you do not specify either the COUNTMODEL statement or the EXTERNALCOUNTS statement. For more information, see the description of the STORE= option in the CNTSELECT procedure in *SAS Econometrics: Econometrics Procedures*.

If you specify the BY statement, then the COUNTSTORE= item store must have been created by using a PROC CNTSELECT step that uses an identical BY statement.

DATA= Data Table

If you specify the BY statement, then the DATA= data table must contain all the BY variables that you specify in the BY statement.

If the severity models in the SEVERITYEST= data table or SEVERITYSTORE= item store contain any scale regressors, then all those regressors must be present in the DATA= data table.

If you specify the programming statements to compute an aggregate adjusted loss, and if any of your specified ADJUSTEDSEVERITY= option symbols depend on the severity adjustment variables, then the DATA= data table must contain all such adjustment variables.

The rest of the contents of the DATA= data table depends on whether you specify the EXTERNALCOUNTS statement. If you specify the EXTERNALCOUNTS statement, then the DATA= data table is expected to contain the COUNT= and ID= variables that you specify in the EXTERNALCOUNTS statement. If you specify the COUNTSTORE= item store but you do not specify the EXTERNALCOUNTS statement, then the DATA= data table must contain all the regressors that are present in the count model that the COUNTSTORE= item store contains.

You do not need to specify the DATA= data table if *all* the following conditions are true:

- You do not specify the BY statement.
- You specify the severity models such that none of them are scale regression models.
- You do not specify the EXTERNALCOUNTS statement.

- You specify the COUNTMODEL statement, or you specify a COUNTSTORE= item store such that the count model contains no count regressors.
- Your severity adjustment programming statements, if you specify any, do not use any external input.

If you specify the BY statement, then PROC CCDM analyzes only the BY groups that are present in the input source of the severity models and count models. If neither the severity models nor the count models contain regression effects, then the DATA= data table must contain BY variables and one row for each BY group that you want PROC CCDM to analyze.

SEVERITYEST= Data Table

The SEVERITYEST= data table is expected to contain the parameter estimates of the severity models. This is a required data table; you must specify it whenever you use PROC CCDM.

The SEVERITYEST= data table must be an OUTEST= data table that the SEVSELECT procedure creates. For more information, see the description of the OUTEST= data table in the SEVSELECT procedure in *SAS Econometrics: Econometrics Procedures*.

If you specify the BY statement, then the SEVERITYEST= data table must contain all the BY variables that you specify in the BY statement.

SEVERITYSTORE= Item Store

The SEVERITYSTORE= item store is expected to be created by using the OUTSTORE= option in a PROC SEVSELECT statement. For more information, see the description of the OUTSTORE= option in the SEVSELECT procedure in *SAS Econometrics: Econometrics Procedures*.

You must specify this item store when you do not specify the SEVERITYEST= data table. Also, if your severity model is a scale regression model that contains classification or interaction effects, then you cannot use the SEVERITYEST= data table. You must specify such severity models by specifying the SEVERITYSTORE= item store.

If you specify the BY statement, then the SEVERITYSTORE= item store must have been created by using a PROC SEVSELECT step that uses an identical BY statement.

Output Data Tables

PROC CCDM writes the output data tables that you specify in the OUT= option of the OUTPUT and OUTSUM statements. It also produces an output data table of individual severity draws when you specify the OUTDRAW= option. The contents of these output data tables are described in the following sections.

OUTDRAW= Data Table

The OUTDRAW= data table records the individual severity draws.

If you specify the BY statement, then the data table contains variables that you specify in the BY statement. In addition, the OUTDRAW= data table contains the following variables:

`_SEVERITYMODEL_`

indicates the name of the severity distribution model.

`_COUNTMODEL_`

indicates the name of the count model. If you specify the `EXTERNALCOUNTS` statement, then the value of this variable is “`_EXTERNAL_`”.

`_DRAWID_`

indicates the identifier for the perturbed sample. This variable is created only when you specify the `NPERTURBEDSAMPLES=` option in the `PROC CCDM` statement. The value of this variable identifies the perturbed sample. A value of 0 indicates an unperturbed sample.

`_BEGINAGG_`

indicates whether the observation starts a new aggregate sample point or continues the previous aggregate sample point. A value of 1 indicates that a new aggregate sample point begins. A value of 0 indicates that the observation continues the aggregate sample point.

`_SEVERITYDRAW_`

indicates the value of the severity draw.

OUTSAMPLE= Data Table

The `OUTSAMPLE=` data table records the full sample of the aggregate loss and aggregate adjusted loss.

If you specify the `BY` statement, then the data table contains variables that you specify in the `BY` statement. In addition, the `OUTSAMPLE=` data table contains the following variables:

`_SEVERITYMODEL_`

indicates the name of the severity distribution model.

`_COUNTMODEL_`

indicates the name of the count model. If you specify the `EXTERNALCOUNTS` statement, then the value of this variable is “`_EXTERNAL_`”.

`<sample variable>`

indicates the value of the aggregate unadjusted loss. The name of this variable is the value of the `SAMPLEVAR=` option in the `OUTPUT` statement. If you do not specify the `SAMPLEVAR=` option, then this variable is named `_AGGSEV_`.

`<adjusted sample variable 1> ... <adjusted sample variable D>`

indicates the values of the D aggregate adjusted losses, where D is the number of adjusted severity symbols that you specify in the `ADJUSTEDSEVERITY=` option in the `PROC CCDM` statement. The names of these variables are the values of the `ADJSAMPLEVAR=` option in the `OUTPUT` statement, which are applied to the symbols in the order in which you specify them in the `ADJUSTEDSEVERITY=` option. If you specify fewer names in the `ADJSAMPLEVAR=` option than symbols in the `ADJUSTEDSEVERITY=` option, then the variable for each remaining adjusted severity symbol has the same name as the corresponding symbol. If you do not specify the `ADJSAMPLEVAR=` option in the `OUTPUT` statement, if you specify only one adjusted severity symbol in the `ADJUSTEDSEVERITY=` option, and if you specify the `ONEADJSEVCOMPAT` option, then the adjusted sample variable is named `_AGGADJSEV_`.

DRAWID indicates the identifier for the perturbed sample. This variable is created only when you specify the **NPERTURBEDSAMPLES=** option in the PROC CCDM statement and the **PERTURBOUT** option in the OUTPUT statement. The value of this variable identifies the perturbed sample. A value of 0 indicates an unperturbed sample.

Observations that have the same **_DRAWID_** variable value are always distributed across all worker nodes for the unperturbed sample (**_DRAWID_** = 0). For each of the perturbed samples (**_DRAWID_** > 0), observations are either distributed or partitioned across worker nodes, depending on the value of the **PERTURBMODE=** option. For more information, see the description of the **PERTURBMODE=AUTO** option.

OUTSUM= Data Table

The OUTSUM= data table records the summary statistics and percentiles of the compound distributions of aggregate loss and aggregate adjusted loss. Only the estimates that you request in the OUTSUM statement are written to the OUTSUM= data table. For more information about the method of naming the variables that correspond to the summary statistics or percentiles, see the description of the **OUTSUM** statement.

If you specify the BY statement, then the data table contains variables that you specify in the BY statement. In addition, the OUTSUM= data table contains the following variables:

SEVERITYMODEL indicates the name of the severity distribution model.

COUNTMODEL indicates the name of the count model. If you specify the **EXTERNALCOUNTS** statement, then the value of this variable is “**_EXTERNAL_**”.

SAMPLEVAR indicates the name of the aggregate loss sample. For an unadjusted sample, the value of the variable is the value of the **SAMPLEVAR=** option that you specify in the OUTPUT statement. If you do not specify the **SAMPLEVAR=** option, then the default value is **_AGGSEV_**.

For an adjusted sample, the value of the variable is decided by the value of the **ADJSAMPLEVAR=** option that you specify in the OUTPUT statement or the name of the adjusted severity symbol. If you do not specify the **ADJSAMPLEVAR=** option, if you specify only one adjusted severity symbol in the **ADJUSTEDSEVERITY=** option, and if you specify the **ONEADJSEVCOMPAT** option, then the value is **_AGGADJSEV_**.

DRAWID indicates the identifier for the perturbed sample. This variable is created only when you specify the **NPERTURBEDSAMPLES=** option in the PROC CCDM statement. The value of this variable identifies the perturbed sample. A value of 0 indicates an unperturbed sample.

Displayed Output

The CCDM procedure optionally produces displayed output by using the Output Delivery System (ODS). You control all output by using the PRINT= option in the PROC CCDM statement. Table 7.5 relates the PRINT= options to ODS tables.

Table 7.5 ODS Tables Produced in PROC CCDM

ODS Table Name	Description	Option
CompoundInfo	Compound distribution information	Default
DataSummary	Input data summary	Default
Percentiles	Percentiles of the aggregate loss sample	PRINT=PERCENTILES
PerturbedPctlSummary	Perturbation analysis of percentiles	PRINT=PERTURBSUMMARY and NPerturbedSamples > 0
PerturbedSummary	Perturbation analysis of summary statistics	PRINT=PERTURBSUMMARY and NPerturbedSamples > 0
SummaryStatistics	Summary statistics of the aggregate loss sample	PRINT=SUMMARYSTATISTICS
Timing	Timing information for various computational stages of the procedure	PRINT=ALL

PRINT= Option

This section provides detailed descriptions of the tables that you display by using different PRINT= options.

- If you do not specify the PRINT= option and if you do not specify the NOPRINT or PRINT=NONE options, then by default PROC CCDM produces the CompoundInfo, DataSummary, and SummaryStatistics ODS tables.

The “Compound Distribution Information” table (ODS name CompoundInfo) displays the information about the severity and count models.

The “Input Data Summary” table (ODS name DataSummary) is displayed when you specify the DATA= data table. The table displays the total number of observations and the valid number of observations in the data table. If you specify the EXTERNALCOUNTS statement, then the table also displays the number of replications and total number of loss events across all replications.

- If you specify PRINT=PERCENTILES, the “Percentiles” table (ODS name Percentiles) is displayed for the distribution of the aggregate loss. The table contains estimates of all predefined percentiles in addition to the percentiles that you request in the OUTSUM statement.

If you specify the programming statements and the ADJUSTEDSEVERITY= option symbols, then an additional table is displayed for the distribution of each aggregate adjusted loss. These tables also contain estimates of all predefined percentiles in addition to the percentiles that you request in the OUTSUM statement.

- If you specify `PRINT=PERTURBSUMMARY`, two tables are displayed for the distribution of the aggregate loss. The “Perturbed Summary Statistics” table (ODS name `PerturbedSummary`) displays the summary of the effect of perturbing model parameters on the following five summary statistics of the distribution: mean, standard deviation, variance, skewness, and kurtosis. The “Perturbed Percentiles” table (ODS name `PerturbedPctlSummary`) displays the perturbation summary for all predefined percentiles in addition to the percentiles that you request in the `OUTSUM` statement.

The tables are displayed only if you specify a value greater than 0 for the `NPERTURBEDSAMPLES=` option.

If you specify a value of P for the `NPERTURBEDSAMPLES=` option, then for each summary statistic and percentile, an average and standard error of the set of P values of that summary statistic or percentile are displayed in the respective perturbation summary tables.

If you specify the programming statements and the `ADJUSTEDSEVERITY=` option symbols, then additional perturbation summary tables are displayed for the distribution of each aggregate adjusted loss.

- If you specify `PRINT=SUMMARYSTATISTICS`, the “Summary Statistics” table (ODS name `SummaryStatistics`) is displayed for the distribution of the aggregate loss. The table contains estimates of the following summary statistics: number of observations in the sample, maximum value in the sample, minimum value in the sample, mean, median, standard deviation, interquartile range, variance, skewness, and kurtosis.

If you specify the programming statements and the `ADJUSTEDSEVERITY=` option symbols, then an additional table of summary statistics is displayed for the distribution of each aggregate adjusted loss.

ODS Graphics

This section describes the use of ODS for creating graphics with the CCDM procedure.

NOTE: To create the plots, PROC CCDM prepares a plot data table that contains an estimate of the empirical distribution function (EDF) for a carefully selected subsample of the simulated aggregate loss sample and brings that table back to the client machine where your SAS session is running. You can control the size of the plot data table by specifying the `NOBS=` and `NADJSEVTOPILOT=` options. The larger the size of this table, the more accurate are the plots, but the more expensive it is to prepare and display the plots. So it is recommended that you request plots only when the full aggregate loss sample is not too large and that you do not specify large values for the `NOBS=` and `NADJSEVTOPILOT=` options.

ODS Graph Names

PROC CCDM assigns a name to each graph that it creates by using ODS. You can use these names to selectively refer to the graphs. The names are listed in [Table 7.6](#).

Table 7.6 ODS Graphics Produced by PROC CCDDM

ODS Graph Name	Plot Description	PLOTS= Option
ConditionalDensityPlot	Conditional density plot	CONDITIONALDENSITY
DensityPlot	Probability density function plot	DENSITY
EDFPlot	Empirical distribution function plot	EDF

Conditional Density Plot

The conditional density plot helps you visually analyze two or three regions of the compound distribution by displaying a density function estimate that is conditional on the values of the aggregate loss that fall in those regions. You can specify the region boundaries in terms of quantiles by using the **LEFTQ=** and **RIGHTQ=** suboptions of the **PLOTS=CONDITIONALDENSITY** option. This is especially useful if you want to see the distribution of aggregate loss values in the right- and left-tail regions.

If you specify m symbols in the **ADJUSTEDSEVERITY=** option and if you specify **NADJSEVTOPLOT= k** in the **PLOTS=** option, then separate sets of conditional density plots are displayed for the first l aggregate adjusted loss samples, where $l = \min(k, m)$.

Probability Density Function Plot

The probability density function (PDF) plot shows the nonparametric estimates of the PDF of the aggregate loss distribution. This plot includes histogram and kernel density estimates.

If you specify m symbols in the **ADJUSTEDSEVERITY=** option and if you specify **NADJSEVTOPLOT= k** in the **PLOTS=** option, then separate density plots are displayed for the first l aggregate adjusted loss samples, where $l = \min(k, m)$.

Empirical Distribution Function Plot

The empirical density function (EDF) plot shows the nonparametric estimate of the cumulative distribution function of the aggregate loss distribution. You can specify the **ALPHA=** suboption of the **PLOTS=EDF** option to request that the upper and lower confidence limits be plotted for each EDF estimate. By default, the confidence interval is not plotted.

If you specify m symbols in the **ADJUSTEDSEVERITY=** option and if you specify **NADJSEVTOPLOT= k** in the **PLOTS=** option, then separate EDF plots are displayed for the first l aggregate adjusted loss samples, where $l = \min(k, m)$.

Examples: CCDM Procedure

Example 7.1: Estimating the Probability Distribution of Insurance Payments

The primary outcome of running the CCDM procedure is the estimate of the compound distribution of aggregate loss, given the distributions of frequency and severity of the individual losses. This aggregate loss is often referred to as the ground-up loss. If you are an insurance company or a bank, you are also interested in acting on the ground-up loss by computing an entity that is derived from the ground-up loss. For example, you might want to estimate the distribution of the amount that you are expected to pay for the losses or the distribution of the amount that you can offload to another organization, such as a reinsurance company. PROC CCDM enables you to specify a severity adjustment program, which is a sequence of SAS programming statements that adjust the severity of the individual loss event to compute the entity of interest. Your severity adjustment program can use external information that is recorded as variables in the observations of the DATA= data table in addition to placeholder symbols for information that PROC CCDM generates internally, such as the severity of the current loss event (`_SEV_`) and the sum of the adjusted severity values of the events that have been simulated thus far for the current sample point (`_ADJSEVSUM_`). If you are analyzing a scenario that contains more than one observation, then you can also access the cumulative severity and cumulative adjusted severity for the current observation by using the `_SEVSUMFOROBS_` and `_ADJSEVSUMFOROBS_` symbols.

This example continues the example of the section “[Scenario Analysis](#)” on page 183 to illustrate how you can estimate the distribution of the aggregate amount that is paid to a group of policyholders. Let the amount that is paid to an individual policyholder be computed by using what is usually referred to as a *disappearing deductible* (Klugman, Panjer, and Willmot 1998, Ch. 2). If X denotes the ground-up loss that a policyholder incurs, d denotes the lower limit on the deductible, d' denotes the upper limit on the deductible, and u denotes the limit on the total payments that are made to a policyholder in a year, then Y , the amount that is paid to the policyholder for each loss event, is defined as follows:

$$Y = \begin{cases} 0 & X \leq d \\ d' \frac{X-d}{d'-d} & d < X \leq d' \\ X & d' < X \leq u \\ u & X > u \end{cases}$$

You can use a set of SAS programming statements to encode this logic.

The following DATA step extends the data table `mylib.GroupOfPolicies` from the example in the section “[Scenario Analysis](#)” on page 183 to include three additional variables for each policyholder: `LowDeductible` to record d , `HighDeductible` to record d' , and `Limit` to record u .

```
/* Generate the scenario data table for multiple policyholders */
data groupOfPolicies(keep=policyholderId age gender carType annualMiles
                    education carSafety income
                    lowDeductible highDeductible limit annualLimit);
  call streaminit(67897);

  do policyholderId=1 to 5;
    age = MAX(int(rand('NORMAL', 35, 15)),16)/50;
```

```

if (rand('UNIFORM') < 0.5) then gender = 1; * female;
else gender = 2; * male;

if (rand('UNIFORM') < 0.7) then carType = 1; * sedan;
else carType = 2; * SUV;

annualMiles = MAX(1000, int(rand('NORMAL', 12000, 5000)))/5000;

educationLevel = rand('UNIFORM');
if (educationLevel < 0.5) then education = 1; *high school graduate;
else if (educationLevel < 0.85) then education = 2; *college graduate;
else education = 3; *advanced degree;

carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */

income = MAX(15000,int(rand('NORMAL', education*30000, 50000)))/100000;

lowDeductible = 100*(1+floor(rand('UNIFORM')*5));
highDeductible = lowDeductible + 500*(1+floor(rand('UNIFORM')*2));
limit = 2500*(1+floor(rand('UNIFORM')*3));
annualLimit = 10000*(1+floor(rand('UNIFORM')*2));

output;
end;
run;

/* Load the data */
data mylib.groupOfPolicies;
  set groupOfPolicies;
run;

```

The data table contains the observations as shown in [Output 7.1.1](#).

Output 7.1.1 Scenario Analysis Data for Multiple Policyholders with Policy Provisions

policyholderId	age	gender	carType	annualMiles	education	carSafety	income
1	1.18	2	1	2.2948	3	0.99532	1.59870
2	0.66	2	2	2.8148	1	0.05625	0.67539
3	0.82	1	2	1.6130	2	0.84146	1.05940
4	0.44	1	1	1.2280	3	0.14324	0.24110
5	0.44	1	1	0.9670	2	0.08656	0.65979

lowDeductible	highDeductible	limit	annualLimit
400	1400	7500	10000
300	1300	2500	20000
100	1100	5000	10000
300	800	5000	20000
100	1100	5000	20000

The following PROC CCDM step estimates the compound distributions of the aggregate loss and the aggregate amount that is paid to the group of policyholders in the data table mylib.GroupOfPolicies by using the count model that is stored in the item store mylib.CountregModel and the lognormal severity model that is

stored in the data table mylib.SevRegEst:

```
/* Simulate the aggregate loss distribution and aggregate adjusted
   loss distribution for the scenario of multiple policyholders */
proc ccdm data=mylib.groupOfPolicies nreplicates=10000 seed=13579 print=all
      countstore=mylib.countregmodel severityest=mylib.sevregest
      nperturbedSamples=50 adjustedseverity=amountPaid;
      severitymodel logn;

      if (_sev_ <= lowDeductible) then
        amountPaid = 0;
      else do;
        if (_sev_ <= highDeductible) then
          amountPaid = highDeductible *
            (_sev_-lowDeductible)/(highDeductible-lowDeductible);
        else
          amountPaid = MIN(_sev_, limit); /* imposes per-loss payment limit */
        end;
      run;
```

The preceding step uses a severity adjustment program to compute the value of the symbol AmountPaid and specifies that symbol in the ADJUSTEDSEVERITY= option in the PROC CCDM step. The program is executed for each simulated loss event. The PROC CCDM supplies your program with the value of the severity in the _SEV_ placeholder symbol.

The “Sample Summary Statistics” table in [Output 7.1.2](#) shows the summary statistics of the compound distribution of the aggregate ground-up loss. The “Adjusted Sample Summary Statistics” table shows the summary statistics of the compound distribution of the aggregate AmountPaid.

Output 7.1.2 Summary Statistics of Compound Distributions of the Total Loss and Total Amount Paid

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Compound Distribution Information	
Severity Model	Lognormal Distribution
Scale Model Effects	carSafety carType income
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Sample Summary Statistics			
Mean	5904.2	Median	4702.3
Standard Deviation	4820.3	Interquartile Range	5196.5
Variance	23235325.2	Minimum	0
Skewness	2.47224	Maximum	72844.8
Kurtosis	13.78076	Sample Size	10000

Output 7.1.2 *continued***The CCDM Procedure**

Severity Model: Logn
Count Model: NegBin(p=2)

Summary Statistics for amountPaid			
Mean	4352.6	Median	3744.0
Standard Deviation	3168.6	Interquartile Range	4095.4
Variance	10039734.9	Minimum	0
Skewness	1.15304	Maximum	31285.3
Kurtosis	2.05121	Sample Size	10000

The perturbation summary of the distribution of AmountPaid is shown in [Output 7.1.3](#). It shows that you can expect to pay a median amount to this group of five policyholders in a year.

Output 7.1.3 Perturbation Summary of the Total Amount Paid

Percentile Perturbation Analysis for amountPaid			
Percentile	Estimate	Standard Error	
1	0.35077	2.45538	
5	391.31471	57.78032	
25	1988.2	164.12121	
50	3781.8	247.83745	
75	6131.5	344.12552	
95	10433.8	508.50465	
99	14094.6	663.38390	
99.5	15570.8	727.38674	
Number of Perturbed Samples = 50			
Size of Each Sample = 10000			

Now consider that, in the future, you want to modify your company's policy provisions to add a limit on the total amount of payment that an individual policyholder receives in one year and to impose a group limit of 15,000 on the total amount of payments the group as a whole receives in one year. You can analyze the effects of these modified policy provisions on the distribution of the aggregate paid amount by recording the individual policyholder's annual limit in the variable AnnualLimit of the input data table and then modifying your severity adjustment program by using the placeholder symbols `_ADJSEVSUMFOROBS_` and `_ADJSEVSUM_`, as in the following PROC CCDM step:

```
/* Simulate the aggregate loss distribution and aggregate adjusted
   loss distribution for the modified set of policy provisions */
proc ccdm data=mylib.groupOfPolicies nreplicates=10000 seed=13579 print=all
      countstore=mylib.countregmodel severityest=mylib.sevregest
      nperturbedSamples=50 adjustedseverity=amountPaid;
severitymodel logn;

if (_sev_ <= lowDeductible) then
  amountPaid = 0;
```

```

else do;
  if (_sev_ <= highDeductible) then
    amountPaid = highDeductible *
      (_sev_-lowDeductible)/(highDeductible-lowDeductible);
  else
    amountPaid = MIN(_sev_, limit); /* imposes per-loss payment limit */

  /* impose policyholder's annual limit */
  amountPaid = MIN(amountPaid, MAX(0,annualLimit - _adjsevsumforobs_));

  /* impose group's annual limit */
  amountPaid = MIN(amountPaid, MAX(0,15000 - _adjsevsum_));
end;
run;

```

The results of the perturbation analysis for these modified policy provisions are shown in [Output 7.1.4](#). When you compare them to the results in [Output 7.1.3](#), you see that the additional policy provisions of restricting the total payment to the policyholder and the group have reduced the median payment and the worst-case payment (99.5th percentile).

Output 7.1.4 Perturbation Summary of the Total Amount Paid after Modified Policy Provisions

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Percentile Perturbation Analysis for amountPaid		
Percentile	Estimate	Standard Error
1	0.51545	2.12215
5	391.98140	61.77616
25	1994.5	167.82065
50	3788.2	230.84670
75	6133.3	318.88793
95	10392.7	453.75218
99	13894.2	614.89744
99.5	14840.0	317.91901
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

Example 7.2: Using Externally Simulated Count Data

The COUNTREG procedure in SAS/ETS and the CNTSELECT procedure in SAS Econometrics enable you to estimate count regression models that are based on the most commonly used discrete distributions, such as the Poisson, negative binomial (both $p = 1$ and $p = 2$), and Conway-Maxwell-Poisson distributions. Those procedures also enable you to fit zero-inflated models that are based on Poisson, negative binomial ($p = 2$), and Conway-Maxwell-Poisson distributions. However, you might encounter situations in which you want to use a different method of fitting count regression models. For example, if you are modeling the number of loss events that are incurred by two financial instruments such that there is some dependency between the two, then you might use multivariate frequency modeling methods and simulate the counts for each instrument by using the dependency structure between the count model parameters of the two instruments. As another example, you might want to use different types of count models for different BY groups in your data; this is not possible in PROC COUNTREG or PROC CNTSELECT. So you need to simulate the counts for such BY groups externally. The CCDM procedure enables you to supply externally simulated counts by using the EXTERNALCOUNTS statement. PROC CCDM then does not need to simulate the counts internally; it simulates only the severity of each loss event by using the severity model estimates that you specify in the SEVERITYEST= data table or the SEVERITYSTORE= item store. The simulation process is described and illustrated in the section “[Simulation with External Counts](#)” on page 221.

Consider that you are a bank, and as part of quantifying your operational risk, you want to estimate the aggregate loss distributions for two lines of business, retail banking and commercial banking, by using some key risk indicators (KRIs). Assume that your model fitting and model selection process has determined that the Poisson regression model and negative binomial regression model are the best-fitting count models for the number of loss events that are incurred in the retail banking and commercial banking lines, respectively. Let CorpKRI1, CorpKRI2, CbKRI1, CbKRI2, and CbKRI3 be the KRIs that you use in the count regression model of the commercial banking line, and let CorpKRI1, RbKRI1, and RbKRI2 be the KRIs that you use in the count regression model of the retail banking line. Examples of corporate-level KRIs (CorpKRI1 and CorpKRI2 in this example) are the ratio of temporary to permanent employees and the number of security breaches that are reported during a year. Examples of KRIs that are specific to the commercial banking business (CbKRI1, CbKRI2, and CbKRI3 in this example) are number of credit defaults, proportion of financed assets that are movable, and penalty claims against your bank because of processing delays. Examples of KRIs that are specific to the retail banking business (RbKRI1 and RbKRI2 in this example) are number of credit cards that are reported stolen, fraction of employees who have not undergone fraud detection training, and number of forged drafts and checks that are presented in a year.

Let the severity of each loss event in the commercial banking business be dependent on two KRIs, CorpKRI1 and CbKRI2. Let the severity of each loss event in the retail banking business be dependent on three KRIs, CorpKRI2, RbKRI1, and RbKRI3. Note that for each line of business, the set of KRIs that are used for the severity model is different from the set of KRIs that are used for the count model, although the two sets overlap. Further, the severity model for retail banking includes a new regressor (RbKRI3) that is not used for any of the count models. Such use of different sets of KRIs for count and severity models is typical of real-world applications.

Let the parameter estimates of the negative binomial and Poisson regression models be available in the data sets Work.CountEstEx2NB2 and Work.CountEstEx2Poisson, respectively. The following statements produce them by using the OUTEST= option in the respective PROC COUNTREG statements:

```

/* Simulate count data */
data cntdataex2(keep=line corpKRI1 corpKRI2
                cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 numloss);
  call streaminit(12345);
  array cx{7} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2;
  array cbetaR{8} _TEMPORARY_ (0.35 1 0 0 0 0 0.5 0.25);
  array cbetaC{8} _TEMPORARY_ (0.9 0.75 0.3 0.1 0.25 0.5 0 0);

  alpha = 0.3;
  theta = 1/alpha;
  do obs=1 to 5000;
    do i=1 to dim(cx);
      cx(i) = rand('NORMAL');
    end;

    line = 'CommercialBanking';
    xbeta = cbetaC(1);
    do i=1 to dim(cx);
      xbeta = xbeta + cx(i) * cbetaC(i+1);
    end;
    Mu = exp(xbeta);
    p = theta/(Mu+theta);
    numloss = rand('NEGB',p,theta);
    output;

    line = 'RetailBanking';
    xbeta = cbetaR(1);
    do i=1 to dim(cx);
      xbeta = xbeta + cx(i) * cbetaR(i+1);
    end;
    lambda = exp(xbeta);
    numloss = rand('POISSON', lambda);
    output;
  end;
run;

proc sort data=cntdataex2;
  by line;
run;

/* Fit negative binomial (p=2) regression model for each business line */
proc countreg data=cntdataex2 outest=countEstEx2NB2;
  by line;
  model numloss = corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3
                rbKRI1 rbKRI2 / dist=negbin;
run;

/* Fit Poisson regression model for each business line */
proc countreg data=cntdataex2 outest=countEstEx2Poisson;
  by line;
  model numloss = corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3
                rbKRI1 rbKRI2 / dist=poisson;
run;

```

NOTE: PROC CNTSELECT does not support the OUTEST= option, so unlike other examples that use PROC CNTSELECT to estimate count models, the preceding statements use PROC COUNTREG.

Let the parameter estimates of the best-fitting severity models, as determined by PROC SEVSELECT, be available in the data table mylib.SevEstEx2, as prepared by the following statements:

```
/* Simulate severity data */
data sevdataEx2(keep=line corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 lossValue);
  array sx{5} corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3;
  array sbetaC{6} _TEMPORARY_ (5 1 0 0.3 0 0);
  array sbetaR{6} _TEMPORARY_ (3.5 0 0.5 0 -0.8 0.6);
  if (_n_=1) then call streaminit(67890);

  set cntdataex2(keep=line numloss corpKRI1 corpKRI2 cbKRI2 rbKRI1);
  sigma = 1;
  alpha = 2.5;
  if (numloss > 0) then do;
    sx(5) = rand('NORMAL'); /* simulate rbKRI3 value */

    if (line='CommercialBanking') then do;
      /* lognormal */
      Mu = sbetaC(1);
      do i=1 to dim(sx);
        Mu = Mu + sx(i) * sbetaC(i+1);
      end;
      lossValue = exp(Mu) * rand('LOGNORMAL')**Sigma;
    end;
  else do;
    /* gamma */
    Mu = sbetaR(1);
    do i=1 to dim(sx);
      Mu = Mu + sx(i) * sbetaR(i+1);
    end;
    lossValue = quantile('Gamma', rand('UNIFORM'), Alpha, exp(Mu));
  end;
  output;
end;
run;

/* Load the data */
data mylib.sevdataEx2;
  set sevdataEx2;
run;

/* Fit severity models for each business line */
proc sevselect data=mylib.sevdataEx2 outest=mylib.sevestEx2;
  by line;
  loss lossValue;
  scalemodel corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3;
  dist logn gamma;
run;
```

Now, consider that you want to estimate the distribution of the aggregate loss for a scenario, which is represented by a specific set of KRI values. The following DATA step illustrates one such scenario:

```

/* Generate a scenario data table for a single operating condition */
data singleScenario (keep=corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3
                    rbKRI1 rbKRI2 rbKRI3);
    array x{8} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3;
    call streaminit(5151);
    do i=1 to dim(x);
        x(i) = rand('NORMAL');
    end;
    output;
run;

/* Load the data */
data mylib.singleScenario;
    set singleScenario;
run;

```

The data table mylib.SingleScenario contains all the KRIs that are included in the count and severity models of both business lines. Note that if you standardize or scale the KRIs while fitting the count and severity models, then you must apply the same standardization or scaling method to the values of the KRIs that you specify in the scenario. In this particular example, all KRIs are assumed to be standardized.

The following DATA step uses the scenario in the data table mylib.SingleScenario to simulate 10,000 replications of the number of loss events that you might observe for each business line and writes the simulated counts to the variable NumLoss of the data table mylib.LossCounts1:

```

/* Simulate multiple replications of the number of loss events that
   you can expect in the scenario being analyzed */
data lossCounts1 (keep=line corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 numloss);
    array cxR{3} corpKRI1 rbKRI1 rbKRI2;
    array cbetaR{4} _TEMPORARY_;
    array cxC{5} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3;
    array cbetaC{6} _TEMPORARY_;

    retain theta;
    if _n_ = 1 then do;
        call streaminit(5151);
        /* read count model estimates */
        set countEstEx2NB2(where=(line='CommercialBanking' and _type_='PARM'));
        cbetaC(1) = Intercept;
        do i=1 to dim(cxC);
            cbetaC(i+1) = cxC(i);
        end;
        alpha = _Alpha;
        theta = 1/alpha;

        set countEstEx2Poisson(where=(line='RetailBanking' and _type_='PARM'));
        cbetaR(1) = Intercept;
        do i=1 to dim(cxR);
            cbetaR(i+1) = cxR(i);
        end;
    end;

    set singleScenario;

```

```

do iline=1 to 2;
  if (iline=1) then line = 'CommercialBanking';
  else line = 'RetailBanking';
  do repid=1 to 10000;
    /* draw from count distribution */
    if (iline=1) then do;
      xbeta = cbetaC(1);
      do i=1 to dim(cxC);
        xbeta = xbeta + cxC(i) * cbetaC(i+1);
      end;
      Mu = exp(xbeta);
      p = theta/(Mu+theta);
      numloss = rand('NEGB',p,theta);
    end;
    else do;
      xbeta = cbetaR(1);
      do i=1 to dim(cxR);
        xbeta = xbeta + cxR(i) * cbetaR(i+1);
      end;
      numloss = rand('POISSON', exp(xbeta));
    end;
    output;
  end;
end;
run;

/* Load the data */
data mylib.lossCounts1;
  set lossCounts1;
run;

```

The mylib.LossCounts1 data table contains the variable NumLoss in addition to the KRIs that the severity regression model uses, which PROC CCDM needs in order to simulate the aggregate loss.

Now, you are ready to estimate the aggregate loss distribution for each line of business by submitting the following PROC CCDM step, in which you specify the EXTERNALCOUNTS statement to request that external counts in the variable NumLoss of the DATA= data table be used for simulation of the aggregate loss:

```

/* Estimate the distribution of the aggregate loss for both
   lines of business by using the externally simulated counts */
proc ccdm data=mylib.lossCounts1 seed=13579 print=all
  severityest=mylib.sevestEx2;
  by line;
  externalcounts count=numloss;
  severitymodel logn gamma;
run;

```

Each observation in the mylib.LossCounts1 data table represents one replication of the external counts simulation process. For each such replication, the preceding PROC CCDM step makes as many severity draws from the severity distribution as the value of the NumLoss variable, and it adds together the severity values from those draws to compute one sample point of the aggregate loss. The severity distribution that is used for making the severity draws has a scale parameter value that is determined by the KRI values in the given observation and by the regression parameter values that are read from the mylib.SevEstEx2 data table.

The summary statistics and percentiles of the aggregate loss distribution for the commercial banking business, which uses the lognormal severity model, are shown in [Output 7.2.1](#). The “Input Data Summary” table indicates that each of the 10,000 observations in the BY group is treated as one replication and that a total of 19,028 loss events were produced by all the replications together. For the scenario in the data table mylib.SingleScenario, the commercial banking line incurs an average aggregate loss as shown in the “Sample Summary Statistics” table, and the chance that the loss will exceed the 99.5th percentile is about 0.5%, as shown in the “Sample Percentiles” table.

Output 7.2.1 Aggregate Loss Summary for Commercial Banking Line

The CCDM Procedure

line=CommercialBanking

Input Data Summary	
Name	LOSSCOUNTS1
Observations	10000
Valid Observations	10000
Replications	10000
Total Count	19028

The CCDM Procedure

Severity Model: Logn
Count Model: External

line=CommercialBanking

Sample Summary Statistics			
Mean	599.56756	Median	344.00869
Standard Deviation	786.88364	Interquartile Range	785.04322
Variance	619185.9	Minimum	0
Skewness	2.80654	Maximum	11176.8
Kurtosis	13.58752	Sample Size	10000

line=CommercialBanking

Sample Percentiles	
Percentile	Value
1	0
5	0
25	52.25800
50	344.00869
75	837.30121
95	2107.5
99	3637.3
99.5	4428.4
Percentile Method = 5	

For the retail banking line, which uses the gamma severity model, the “Sample Percentiles” table in [Output 7.2.2](#) indicates that the median operational loss of that line of business is about 70 units.

Output 7.2.2 Aggregate Loss Percentiles for Retail Banking Line

line=RetailBanking	
Sample Percentiles	
Percentile	Value
1	0
5	0
25	0
50	70.46309
75	142.15990
95	271.76555
99	378.32158
99.5	437.12308
Percentile Method = 5	

When you conduct simulation and estimation for a scenario that contains only one observation, you assume that the operating environment does not change over the period of time that is being analyzed. That assumption might be valid for shorter durations and stable business environments, but often the operating environments change, especially if you are estimating the aggregate loss over a longer period of time. So you might want to include in your scenario all the possible operating environments that you expect to see during the analysis time period. Each environment is characterized by its own set of KRI values. For example, the operating conditions might change from quarter to quarter, and you might want to estimate the aggregate loss distribution for the entire year. You start the estimation process for such scenarios by creating a scenario data table. The following DATA step creates the data table `mylib.MultiConditionScenario`, which consists of four operating environments, one for each quarter:

```
/* Generate a scenario data table for multiple operating conditions */
data multiConditionScenario (keep=opEnvId corpKRI1 corpKRI2
    cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3);
    array x{8} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3;
    call streaminit(5151);
    do opEnvId=1 to 4;
        do i=1 to dim(x);
            x(i) = rand('NORMAL');
        end;
        output;
    end;
run;

/* Load the data */
data mylib.multiConditionScenario;
    set multiConditionScenario;
run;
```

All four observations of the data table `mylib.MultiConditionScenario` together form one scenario. When you simulate the external counts for such multi-entity scenarios, one replication consists of the possible number of loss events that can occur as a result of all four operating environments. In any given replication, some operating environments might not produce any loss event, or all four operating environments might produce some loss events. The following DATA step creates the data table `mylib.LossCounts2`, which contains, for each business line, 10,000 replications of the loss counts, and which identifies each replication by using the

variable RepId:

```

/* Simulate multiple replications of the number of loss events that you can
   expect for all operating environments in the scenario being analyzed */
data lossCounts2 (keep=line opEnvId corpKRI1 corpKRI2 cbKRI2
                  rbKRI1 rbKRI3 repid numloss);
array cxR{3} corpKRI1 rbKRI1 rbKRI2;
array cbetaR{4} _TEMPORARY_;
array cxC{5} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3;
array cbetaC{6} _TEMPORARY_;

/* Read the count model estimates */
retain theta;
if _n_ = 1 then do;
  call streaminit(5151);
  set countEstEx2NB2(where=(line='CommercialBanking' and _type_='PARM'));
  cbetaC(1) = Intercept;
  do i=1 to dim(cxC);
    cbetaC(i+1) = cxC(i);
  end;
  alpha = _Alpha;
  theta = 1/alpha;

  set countEstEx2Poisson(where=(line='RetailBanking' and _type_='PARM'));
  cbetaR(1) = Intercept;
  do i=1 to dim(cxR);
    cbetaR(i+1) = cxR(i);
  end;
end;

/* Find the number of observations in the scenario data set */
nobs = 0;
do while(last=0);
  set multiConditionScenario end=last;
  nobs = nobs+1;
end;
nobstotal=nobs;

do iline=1 to 2;
  if (iline=1) then line = 'CommercialBanking';
  else line = 'RetailBanking';
  do repid=1 to 10000;
    do nobs=1 to nobstotal;
      set multiConditionScenario point=nobs;
      /* Draw from the appropriate count distribution */
      if (line = 'CommercialBanking') then do;
        xbeta = cbetaC(1);
        do i=1 to dim(cxC);
          xbeta = xbeta + cxC(i) * cbetaC(i+1);
        end;
        Mu = exp(xbeta);
        p = theta/(Mu+theta);
        numloss = rand('NEGB',p,theta);
      end;
    end;
  end;
end;

```

```

        else if (line = 'RetailBanking') then do;
            xbeta = cbetaR(1);
            do i=1 to dim(cxR);
                xbeta = xbeta + cxR(i) * cbetaR(i+1);
            end;
            numloss = rand('POISSON', exp(xbeta));
        end;
        output;
    end;
end;
run;

```

To use the replication identifier variable with PROC CCDM, you need to partition the input data table of counts. The PARTITION= data table option partitions the data table such that a group of observations that have the same value for the variables line and Repld are located on the same worker node.

The following DATA step loads the data set Work.LossCounts2 into a data table in your CAS session that is associated with the mylib CAS engine libref. The DATA step assumes that your CAS engine libref is named mylib, but you can substitute any appropriately defined CAS engine libref.

```

/* Load the data and partition by the specified variables. */
data mylib.lossCounts2(partition=(line repld));
    set lossCounts2;
run;

```

Output 7.2.3 shows some observations of the data table mylib.LossCounts2 for each business line. For the first replication (Repld=1) of the commercial banking line, only operating environments 3 and 4 incur loss events; the other environments incur no loss events. For the second replication (Repld=2), all operating environments incur at least one loss event. For the first replication (Repld=1) of the retail banking line, operating environments 2, 3, and 4 incur two, one, and three loss events, respectively.

Output 7.2.3 Snapshot of the External Counts Data with Replication Identifier

line	opEnvld	corpKRI1	corpKRI2	cbKRI2	rbKRI1	rbKRI3	repld	numloss
CommercialBanking	1	0.45224	0.40661	-0.33680	-1.08692	-2.20557	1	0
CommercialBanking	2	-0.03799	0.98670	-0.03752	1.94589	1.22456	1	0
CommercialBanking	3	-0.29120	-0.45239	0.98855	-0.37208	-1.51534	1	3
CommercialBanking	4	0.87499	-0.67812	-0.04839	-1.44881	0.78221	1	1
CommercialBanking	1	0.45224	0.40661	-0.33680	-1.08692	-2.20557	2	2
CommercialBanking	2	-0.03799	0.98670	-0.03752	1.94589	1.22456	2	5
CommercialBanking	3	-0.29120	-0.45239	0.98855	-0.37208	-1.51534	2	12
CommercialBanking	4	0.87499	-0.67812	-0.04839	-1.44881	0.78221	2	12
RetailBanking	1	0.45224	0.40661	-0.33680	-1.08692	-2.20557	1	0
RetailBanking	2	-0.03799	0.98670	-0.03752	1.94589	1.22456	1	2
RetailBanking	3	-0.29120	-0.45239	0.98855	-0.37208	-1.51534	1	1
RetailBanking	4	0.87499	-0.67812	-0.04839	-1.44881	0.78221	1	3
RetailBanking	1	0.45224	0.40661	-0.33680	-1.08692	-2.20557	2	2
RetailBanking	2	-0.03799	0.98670	-0.03752	1.94589	1.22456	2	2
RetailBanking	3	-0.29120	-0.45239	0.98855	-0.37208	-1.51534	2	0
RetailBanking	4	0.87499	-0.67812	-0.04839	-1.44881	0.78221	2	1

You can now use this simulated count data to estimate the distribution of the aggregate loss that is incurred in all four operating environments by submitting the following PROC CCDM step, in which you specify the replication identifier variable RepId in the ID= option of the EXTERNALCOUNTS statement:

```
/* Estimate the distribution of the aggregate loss for both
   lines of business by using the externally simulated counts
   for the multiple operating environments */
proc ccdm data=mylib.lossCounts2 seed=13579 print=all
    severityest=mylib.sevestEx2;
    by line;
    externalcounts count=numloss id=repid;
    severitymodel logn gamma;
run;
```

Within each BY group, for each value of the variable RepId, one point of the aggregate loss sample is simulated by using the process that is described in the section “[Simulation with External Counts](#)” on page 221.

The summary statistics and percentiles of the distribution of the aggregate loss, which is the aggregate of the losses across all four operating environments, are shown in [Output 7.2.4](#) for the commercial banking line. The “Input Data Summary” table indicates that the BY group contains 10,000 replications and that a total of 145,721 loss events are generated across all replications. The “Sample Percentiles” table indicates the median aggregate loss, as defined by the 50th percentile, and a worst-case loss, as defined by the 99.5th percentile, from the commercial banking line when you combine losses from all four operating environments.

Output 7.2.4 Aggregate Loss Summary for the Commercial Banking Line in Multiple Operating Environments

The CCDM Procedure

line=CommercialBanking

Input Data Summary

Name	LOSSCOUNTS2
Observations	40000
Valid Observations	40000
Replications	10000
Total Count	145721

line=CommercialBanking

Sample Percentiles

Percentile	Value
1	843.45367
5	1525.9
25	3112.8
50	4789.0
75	6983.8
95	11502.3
99	15822.8
99.5	17749.2
Percentile Method = 5	

Example 7.3: Scenario Analysis with Rich Regression Effects and BY Groups

This example illustrates scenario analysis when frequency and severity models use regression models that contain classification and interaction effects. It also illustrates how you can analyze scenarios for multiple groups of observations in one PROC CCDM step without having to simulate counts externally.

The example in the section “[Scenario Analysis](#)” on page 183 encodes the discrete-valued, nominal (nonordinal) variables Gender, CarType, and Education as numerical variables that have an implied order. For example, a high school diploma is assigned a smaller number than an advanced degree. This method of forcing an order on otherwise nonordinal (categorical) variables is not natural and might lead to biased estimates. A more accurate approach is to treat such variables as classification variables that enter the statistical analysis or model not through their values but through their levels. For example, when you specify Education as a classification variable, the modeling process creates different parameters for the Education = ‘High School’ and Education = ‘Advanced Degree’ levels and estimates a regression coefficient for each. When you specify such variables in the CLASS statement of PROC CNTSELECT and PROC SEVSELECT, those procedures perform the appropriate levelization for you, which is the process of finding and transforming levels into regression parameters. For more information, see the description of the CLASS statement in Chapter 26, “[SEVSELECT Procedure](#).”

In addition to specifying nominal variables as classification (CLASS) variables, you can include interaction effects in severity and frequency models. For example, you might want to evaluate how the distribution of losses that are incurred by a policyholder who has a college degree and drives an SUV differs from that of a policyholder who has an advanced degree and drives a sedan. You can do this by including an interaction between CarType and Education in your severity model. Similarly, if you want to evaluate how the number of losses that a policyholder incurs per year varies by the number of annual miles for different types of cars, you can include an interaction between CarType and AnnualMiles in your frequency model. Analyzing such a rich set of regression effects can help you make more accurate predictions about the frequency and severity distributions of losses. PROC CCDM is designed to use such rich models to simulate a more accurate distribution of the aggregate loss.

To start the scenario analysis process, let the following programming statements simulate the frequency and severity events that are affected by the regression effects:

```
/* Define a macro to simulate regressor data */
%macro generateRegvars;
    age = MAX(int(rand('NORMAL', 35, 15)),16)/50;

    if (rand('UNIFORM') < 0.5) then do;
        genderNum = 1; * female;
        gender = 'F';
    end;
    else do;
        genderNum = 2; * male;
        gender = 'M';
    end;

    if (rand('UNIFORM') < 0.7) then do;
        carTypeNum = 1; * sedan;
        carType = 'Sedan';
    end;
    else do;
```

```

    carTypeNum = 2; * SUV;
    carType = 'SUV';
end;

educationLevel = rand('UNIFORM');
if (educationLevel < 0.5) then do;
    educationNum = 1; *high school graduate;
    education = 'High School';
end;
else if (educationLevel < 0.85) then do;
    educationNum = 2; *college graduate;
    education = 'College';
end;
else do;
    educationNum = 3; *advanced degree;
    education = 'Advanced Degree';
end;

annualmiles = MAX(1000, int(rand('NORMAL', 12000, 5000)))/5000;

carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */

income = MAX(15000, int(rand('NORMAL', educationNum*30000, 50000)))/100000;
%mend;

/* Simulate loss severity data */
%let npolicy=1000;

data losses(keep=region policyholderId age gender carType
              annualmiles education carSafety
              income lossAmount noloss);
    call streaminit(12345);
    array cx{6} age genderF milesSUV milesSedan educationAdv educationColl;
    array cbeta{7} _TEMPORARY_ (1 0.75 -1 -1.2 -0.6 0.5 0.75);

    array czx{4} age carTypeSUV educationAdv educationColl;
    array czbeta{5} _TEMPORARY_ (-0.75 -1 -1.2 0.6 0.7);

    array sx{8} _temporary_;
    array sbeta{9} _TEMPORARY_ (5 0.5 1.15 -0.75 -0.3 0.4 0.7 -0.5 -0.3);

    length gender $1 carType $8 education $16;

    alpha = 0.9; theta = 1/alpha;

    do iregion=1 to 2;
        if (iregion = 1) then do;
            region = 'East';
            sigma = 0.5;
        end;
        else do;
            region = 'West';
            /* slightly change parameter estimates for this group */
            do i=1 to dim(cbeta);

```

```

        cbeta(i) = cbeta(i) + 0.1;
    end;
    do i=1 to dim(czbeta);
        czbeta(i) = czbeta(i) + 0.1;
    end;
    do i=1 to dim(sbeta);
        sbeta(i) = sbeta(i) + 0.05;
    end;
    sigma = 0.3;
    alpha = 1.6; theta = 1/alpha;
end;

do policyholderId=1 to &npolicy;
    /* simulate policyholder and vehicle attributes */
    %generateRegvars;

    do i=1 to dim(sx);
        sx(i) = 0;
    end;
    if (carTypeNum = 2) then do;
        sx(1) = 1;
        carTypeSUV = 1;
        milesSUV = annualmiles;
        milesSedan = 0;
    end;
    else do;
        carTypeSUV = 0;
        milesSUV = 0;
        milesSedan = annualmiles;
    end;

    if (genderNum = 1) then do;
        sx(2) = 1;
        genderF = 1;
    end;
    else genderF = 0;

    sx(3) = carSafety;
    sx(4) = income;
    if (educationNum = 3 and carTypeNum = 2) then sx(5) = 1;
    if (educationNum = 2 and carTypeNum = 2) then sx(6) = 1;
    if (educationNum = 3 and carTypeNum = 1) then sx(7) = 1;
    if (educationNum = 2 and carTypeNum = 1) then sx(8) = 1;

    if (educationNum = 3) then do;
        educationAdv = 1;
        educationColl = 0;
    end;
    else do;
        educationAdv = 0;
        if (educationNum = 2) then educationColl = 1;
        else educationColl = 0;
    end;
end;

```



```

/* simulate number of losses incurred by this policyholder */
czxbeta = czbeta(1);
do i=1 to dim(czx);
    czxbeta = czxbeta + czx(i) * czbeta(i+1);
end;
phi = exp(czxbeta)/(1+exp(czxbeta));

if (rand('UNIFORM') < phi) then do;
    * zero process is selected with probability phi;
    numloss = 0;
end;
else do;
    * negbin process is selected with probability (1-phi);
    cxbeta = cbeta(1);
    do i=1 to dim(cx);
        cxbeta = cxbeta + cx(i) * cbeta(i+1);
    end;
    Mu = exp(cxbeta);
    numloss = rand('POISSON',Mu);
end;

/* simulate severity of each loss */
if (numloss > 0) then do;
    noloss = 0;
    Mu = sbeta(1);
    do i=1 to dim(sx);
        Mu = Mu + sx(i) * sbeta(i+1);
    end;
    do iloss=1 to numloss;
        lossAmount = exp(Mu) * rand('LOGNORMAL')**Sigma;
        output;
    end;
end;
else do;
    noloss = 1;
    lossAmount = .;
    output;
end;
end;
end;

run;

/* Load the loss severity data */
data mylib.losses;
    set losses;
run;

/* Simulate loss counts data */
proc sort data=losses out=lossesByPolicy;
    by region policyholderId;
run;

data losscounts(drop=policyholderId carSafety lossAmount noloss);
    set lossesByPolicy;

```

```

    by region policyholderId;
    retain numloss zeroloss 0;
    if (noloss eq 1) then
        zeroloss = zeroloss + 1;
    else
        numloss = numloss + 1;
    if (last.policyholderId) then do;
        output;
        numloss = 0;
    end;
run;

/* Load the loss counts data */
data mylib.losscounts;
    set losscounts;
run;

```

The following programming statements use the data tables that the preceding statements create to fit the severity and count models that contain a certain set of regression effects:

```

proc sevselect data=mylib.losses(where=(lossAmount ne .))
    covout outstore=mylib.sevstore print=all;
    by region;
    loss lossAmount;
    class carType gender education;
    scalemodel carType gender carSafety income education*carType
        income*gender carSafety*income;
    selection method=backward(select=sbc) details=summary;
    dist logn burr;
run;

proc cntselect data=mylib.losscounts store=mylib.cstore;
    by region;
    class gender carType education;
    model numloss = age income gender carType*annualmiles
        education / dist=poisson;
    zeromodel numloss ~ age income carType education;
    selection method=backward(select=sbc) details=summary;
run;

```

Note the following points about these statements:

- The data tables are organized in groups of observations that represent data from two regions, East and West. You can analyze both groups at the same time by specifying the BY statement with Region as the BY variable. The observations in the input data table do not need to be sorted by the BY variables.
- Both severity and count models use three CLASS variables. The severity model includes three interaction effects (Education*CarType, Income*Gender, and CarSafety*Income) and four main effects.
- The count model is a zero-inflated model, which is essentially a mixture of two models: a model to estimate the occurrence of zero loss events and a model to estimate nonzero counts. The model for zero counts, which is specified in the ZEROMODEL statement, is a regression model that has four main effects and uses the default logistic link function. The model for nonzero counts, which is specified in

the MODEL statement, is a Poisson model that has one interaction effect (CarType*AnnualMiles) and four main effects.

- Both the PROC SEVSELECT and PROC CNTSELECT steps use the SELECTION statement to perform automatic effect selection. In particular, they use the backward selection method to choose a subset of regression effects that maximize the Schwarz Bayesian information criterion (SBC). PROC SEVSELECT conducts the effect selection process independently for each distribution.
- The PROC SEVSELECT step uses the OUTSTORE= option to store the parameter estimates in an item store. When your scale regression model contains classification or interaction effects, you must store the parameter estimates in an item store instead of storing them in an OUTEST= data table because PROC CCDDM cannot obtain the necessary information about classification or interaction effects from an OUTEST= data table.

For severity models, you can inspect the “All Fit Statistics” table for each BY group to decide which severity distribution has the best fit for the data in that BY group. When you specify the SELECTION statement, each row in this table shows the fit statistics of the scale regression model that contains only the final selected set of regression effects for the candidate distribution of that row. The tables in [Output 7.3.1](#) indicate that the model that uses the lognormal distribution is the best according to the majority of fit statistics for both regions, so you can just specify the LOGN distribution in the SEVERITYMODEL statement of the subsequent PROC CCDDM step. In cases where different distributions are deemed best for different BY groups, you have two alternatives. The first alternative is to run one PROC CCDDM step with the SEVERITYMODEL statement that specifies the best distributions from all BY groups and to later process the results of only the best severity distribution in each BY group. The second alternative is to run separate PROC CCDDM steps for each BY group by specifying the best severity distribution for that BY group in the SEVERITYMODEL statement. The latter alternative requires more coding effort but it is more efficient than the first alternative, which runs the analysis for all best severity distributions for all BY groups.

Also note that in some cases, you might see that the likelihood-based fit statistics (–2 log likelihood, AIC, AICC, and BIC) mark one distribution as the best distribution and the EDF-based statistics (KS, AD, and CvM) mark another distribution as the best distribution. In such cases, it is recommended that you conduct aggregate loss simulation by using both severity distributions and compare the summary statistics and percentiles of resulting aggregate loss distributions.

Output 7.3.1 Comparison of Final Models of Severity Distributions

The SEVSELECT Procedure

Burr Distribution Selected Model

region=East

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	8715*	8735*	8735*	8782*	2.68389*	37.94117	2.64223*
Burr	8721	8743	8743	8794	2.76081	33.56827*	2.92479

Asterisk (*) denotes the best model in the column.

Output 7.3.1 *continued*

region=West							
All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	12989*	13009*	13009*	13060*	6.34841	305.19555	15.48743*
Burr	13004	13026	13026	13082	6.31408*	233.46457*	15.52778
Asterisk (*) denotes the best model in the column.							

If you want to examine the summary of the effect selection process and the final selected set of regression effects for a severity distribution, you can examine the “Selection Summary” and “Selected Effects” tables as shown in [Output 7.3.2](#). These tables indicate that the best scale regression model for the lognormal distribution in the region=East group is obtained after removing the carSafety*income and income*gender effects.

Output 7.3.2 Summary of the Effect Selection Process for LOGN Severity Model for Region=East**The SEVSELECT Procedure****Logn Distribution
Selection Details**

region=East			
Selection Summary			
Step	Effect Removed	Number Effects In	SBC
0		8	8793.6358
1	carSafety*income	7	8787.0918
2	income*gender	6	8781.8787*
3	income	5	8807.5961
4	carType*education	4	8921.5365
* Optimal Value Of Criterion			
region=East			
Selected Effects: Intercept carType gender carSafety income carType*education			

You can examine a similar summary of the selection process that the CNTSELECT procedure follows to select the best effect subset for the count distribution, as shown in [Output 7.3.3](#). The best model for the zero-inflated Poisson (ZIP) distribution in the region=East group is obtained after removing the education, age, and income effects from the model for the zero counts and removing the income effect from the model for nonzero counts. So the final model contains only the carType effect in the zero-model and age, income, gender, carType*annualmiles, and education effects in the nonzero-model.

Output 7.3.3 Summary of the Effect Selection Process for ZIP Count Model for Region=East**The CNTSELECT Procedure**

region=East			
Selection Summary			
Step	Effect Removed	Number Effects In	SBC
0		11	2099.9659
1	Inf_education	10	2092.2022
2	Inf_age	9	2085.4951
3	Inf_income	8	2079.0596
4	income	7	2074.1768*
5	Inf_carType	6	2074.7483
6	age	5	2101.6437
7	education	4	2162.7788
* Optimal Value Of Criterion			

PROC SEVSELECT conducts a separate effect selection analysis for each severity distribution within each BY group, so scale regression models for different severity distributions in the same BY group can contain different final sets of selected effects. Similarly, scale regression models for the same severity distribution in different BY groups can contain different final sets of selected effects. PROC CNTSELECT also conducts a separate effect selection analysis for each BY group, so the final sets of selected effects in the count regression model can be different for different BY groups.

After the CNTSELECT and SEVSELECT procedures have selected the best frequency and severity models, it is time to estimate the distribution of the aggregate loss by using the CCDM procedure. Given that the models depend on regressors, first you need to create the scenario data table, which must contain the final set of regressors that are used in both the severity model and the frequency model. Even if your models contain classification or interaction effects, your scenario data table needs to contain only the columns for individual variables of the effects. PROC CCDM internally performs *levelization* of each observation, which is the process of expanding the variable values to match them with the parameters of each effect. For example, you do not need to create multiple columns for the parameters of the carType*education effect. You just need to create two columns for carType and education. PROC CCDM internally creates the six columns that correspond to the six parameters that are associated with the carType*education effect. However, you must make sure that the values of each classification variable, such as carType, in the scenario data table belong to the same set of values that appear in the input data tables of PROC SEVSELECT and PROC CNTSELECT.

A typical scenario for an insurance application usually consists of a large number of policyholders. However, for illustration purposes, this example uses a small scenario of only a few policyholders per region that is generated by the following DATA step and shown in [Output 7.3.4](#):

```
/* Create scenario for internal count simulation */
data scenario(keep=region policyholderId age gender
               carType annualmiles education carSafety income);
  call streaminit(456);

  length region $4 gender $1 carType $8 education $16;

  do iregion=1 to 2;
    if (iregion = 1) then do;
```

```

        region = 'East';
        npolicy = 3;
    end;
    else do;
        region = 'West';
        npolicy = 5;
    end;
    do policyholderId = 1 to npolicy;
        %generateRegvars;
        output;
    end;
end;
run;

/* Load the scenario data */
data mylib.scenario;
    set scenario;
run;

```

Output 7.3.4 Data Table mylib.Scenario for BY-Group Processing

Obs	region	gender	carType	education	age	annualmiles	carSafety	income
1	East	F	SUV	High School	1.16	2.1540	0.29288	0.26090
2	East	F	Sedan	High School	0.86	2.3978	0.69844	0.15000
3	East	F	Sedan	Advanced Degree	0.78	1.9926	0.59421	0.58808
4	West	M	Sedan	College	0.82	1.8550	0.66849	0.15000
5	West	M	SUV	College	0.40	3.6240	0.23194	1.25274
6	West	M	Sedan	High School	0.62	3.6162	0.86477	0.42597
7	West	F	Sedan	College	0.32	3.4598	0.66294	0.36132
8	West	M	Sedan	Advanced Degree	0.90	3.2580	0.37172	0.15000

The following PROC CCDM step simulates the aggregate losses for the scenario in the data table mylib.Scenario by each region:

```

proc ccdm data=mylib.scenario nreplicates=10000 seed=123 print=all
    severitystore=mylib.sevstore countstore=mylib.cstore
    nperturb=30;
    by region;
    severitymodel logn;
    outsum out=mylib.agglossStats mean stddev skewness kurtosis
    pctlpts=(90 97.5 99.5);
run;

```

The BY statement must contain the same set of variables and in the same order as the variables in the BY statements that you specify in PROC SEVSELECT and PROC CNTSELECT. The **SEVERITYSTORE=** and **COUNTSTORE=** options specify the item stores that contain the effect information and parameter estimates of the severity and count models, respectively, for both BY groups. The count item store that PROC CNTSELECT creates always contains the covariance estimates of the count model parameters, and the COVOUT option in the PROC SEVSELECT step ensures that the severity item store includes the covariance estimates. Availability of these covariance estimates makes the perturbation analysis that the NPERTURB= option requests more accurate.

One of the main points that this example illustrates is that irrespective of the complexity of the regression effects in your frequency or severity models, you do not have to do any extra work to convey the effect information to PROC CCDDM, except to specify the item stores. The frequency and severity item stores contain all the necessary information that PROC CCDDM needs for computing the values of the regressor-dependent parameters of the underlying frequency and severity distributions before making random draws from those distributions. When you use the BY and SELECTION statements in PROC CNTSELECT and PROC SEVSELECT, the respective item stores contain the information about the final selected regression effects for each distribution in each BY group. PROC CCDDM uses this information to process only the regression parameters that correspond to the selected regression effects for each distribution in each BY group.

Output 7.3.5 and Output 7.3.6 show the summary of the perturbation analysis for the two regions.

Output 7.3.5 Aggregate Loss Simulation Results for Region=East

The CCDDM Procedure

Severity Model: Logn
Count Model: ZIP

region=East

Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	0	0
5	0	0
25	1.32068	7.11212
50	578.05674	67.55693
75	2330.8	232.67455
90	3688.4	331.99961
95	4495.2	384.34629
97.5	5215.7	442.70674
99	6096.0	486.57812
99.5	6708.0	543.85538
Number of Perturbed Samples = 30		
Size of Each Sample = 10000		

Output 7.3.6 Aggregate Loss Simulation Results for Region=West

region=West		
Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	22.54771	33.67374
5	172.06866	63.67050
25	478.18324	68.06828
50	715.15933	69.06287
75	982.39124	72.51530
90	1254.1	77.78615
95	1434.9	82.18084
97.5	1597.6	85.80075
99	1793.8	88.37708
99.5	1945.0	92.41512
Number of Perturbed Samples = 30		
Size of Each Sample = 10000		

Example 7.4: Illustrating Pure Premium and Custom Simulation Modes

The CCDM procedure supports simulation modes other than the default collective risk simulation mode. This example illustrates how you can use those modes to simulate samples for different types of aggregate losses. Suppose you are an actuary at an insurance company and you want to estimate the distribution of the pure premium for the claims that you expect to be filed by the policyholders in a particular time period. The pure premium method of estimating claims cost is useful for certain types of peril. Assume that you have estimated a simple severity distribution model for the average severity of each claim by submitting the following statements, where the data table mylib.ClaimSev contains a simulation of historic average severity data:

```
/* Simulate average severity data and load them */
data claimsev(keep=averagesev);
  call streaminit(789);
  label averagesev='Average severity per claim';
  do n = 1 to 10000;
    averagesev = rand('Weibull', 3, 500);
    output;
  end;
run;
data mylib.claimsev;
  set claimsev;
run;

/* Fit severity models and write estimates to an item store */
proc sevselect data=mylib.claimsev crit=sbc covout outstore=mylib.sevstore;
  loss averagesev;
  dist gamma igauss logn weibull;
run;
```


Output 7.4.1 Comparison of Severity Distributions for Average Severity**The SEVSELECT Procedure**

Model Selection			
Distribution	Converged	SBC	Selected
Gamma	Yes	130839	No
Igauss	Yes	132718	No
Logn	Yes	132081	No
Weibull	Yes	130163	Yes

The “Model Selection” table in [Output 7.4.1](#) shows that the Weibull distribution is the best-fitting distribution for the average severity. Next, assume that you have used a time-series analysis method to create a model for average number of claims that you expect to process in a particular time period. Let the average frequency follow a normal distribution $\mathcal{N}(5000, 100)$, which implies that the point forecast of the average frequency for the time period of interest is 5,000 with an associated standard error of 100. To analyze the distribution of the pure premium, which is the product of average severity and average frequency, you can use the `SIMULATIONMODE=PUREPREMIUM` option in PROC CCDM as shown in the following statements:

```
proc ccdm severitystore=mylib.sevstore simulationmode=purepremium
    seed=135 nreplicates=500000 print=all plots=density;
    severitymodel weibull;
    countmodel normal(5000, 100);
run;
```

The COUNTMODEL statement specifies the distribution of the average frequency. The SEVERITYMODEL statement requests that only the Weibull distribution be read from the SEVERITYSTORE= item store. The PRINT=ALL option displays the quantitative summary of the pure premium distribution, which is shown in the “Sample Summary Statistics” and “Sample Percentiles” tables in [Output 7.4.2](#). The PLOTS= option creates a plot of the nonparametric estimate of the probability density function (PDF) of the pure premium distribution, which is shown in [Output 7.4.3](#).

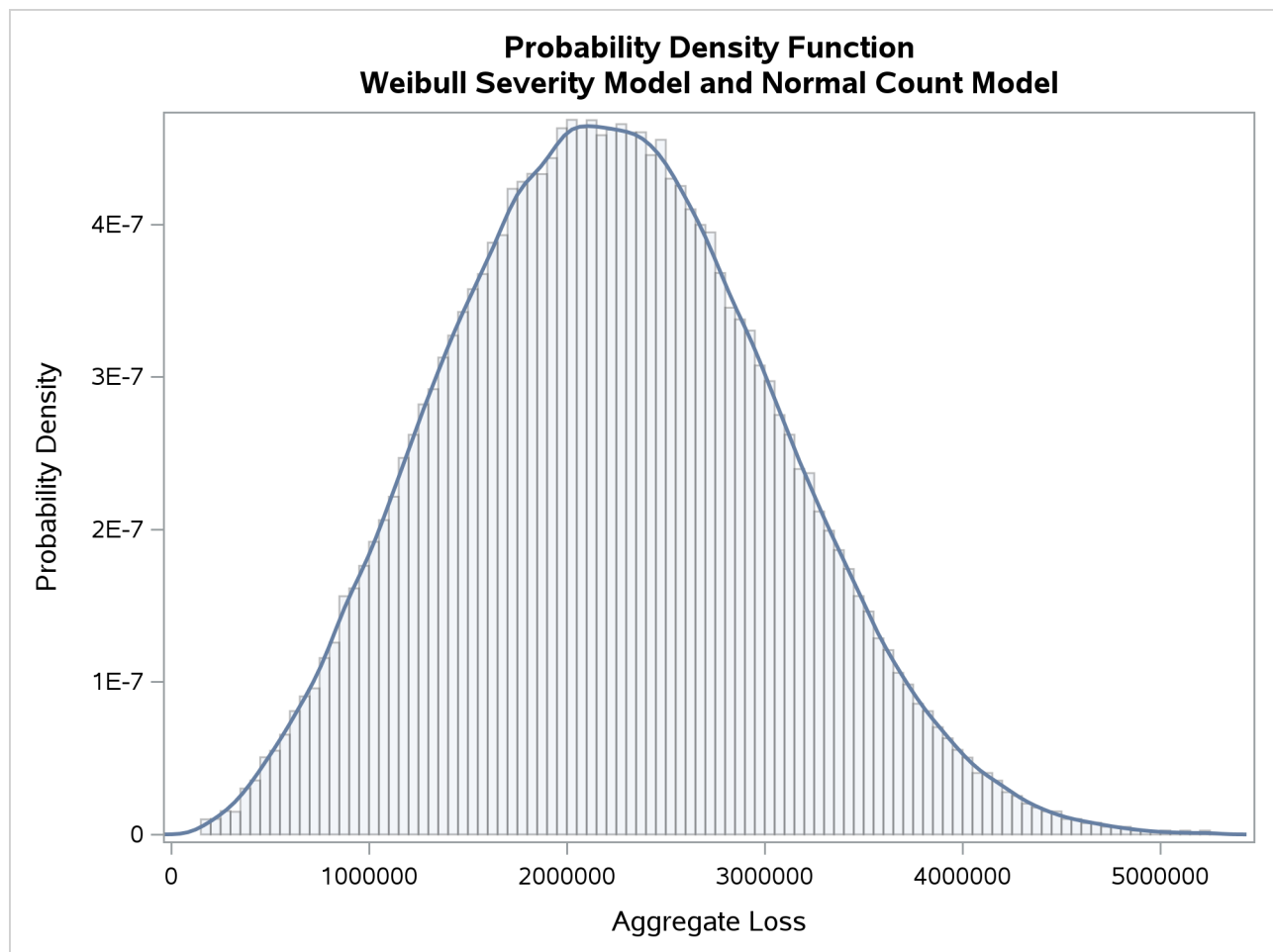
Output 7.4.2 Summary Statistics and Percentiles of the Pure Premium Distribution**The CCDM Procedure**

Severity Model: Weibull
Count Model: Normal(5000,100)

Sample Summary Statistics			
Mean	2225717.6	Median	2204042.8
Standard Deviation	817761.9	Interquartile Range	1143309.1
Variance	6.687345E11	Minimum	32858.0
Skewness	0.18463	Maximum	5889889.8
Kurtosis	-0.26164	Sample Size	500000

Output 7.4.2 *continued*

Sample Percentiles	
Percentile	Value
1	527953.7
5	915327.3
25	1638602.1
50	2204042.8
75	2781911.1
95	3613171.7
99	4180009.8
99.5	4381630.4
Percentile Method = 5	

Output 7.4.3 Nonparametric PDF Plot of the Pure Premium Distribution

Now consider a situation where the average frequency is defined as the average number of claims *per policy* and you have estimated a separate distribution model for the number of policies, denoted by n_{Policy} , that you expect to be in force for the particular time period. If you want to estimate a distribution of the total

expected loss across all policies according to the pure premium method, then you need to multiply the pure premium by an estimate of `nPolicy`. Because `nPolicy` is stochastic, the simulation process needs to make a random draw from its distribution. Assuming that the average number of claims per policy follows a standard negative binomial distribution with parameters $p = 0.75$ and $n = 5$ and that `nPolicy` follows a lognormal distribution with parameters $\mu = 10$ and $\sigma = 0.35$, the following PROC CCDM step simulates a sample of the total expected loss, `tLoss`, across all policies:

```
proc ccdm severitystore=mylib.sevstore simulationmode=purepremium
    seed=135 nreplicates=500000 print=all adjustedseverity=tloss;
    severitymodel weibull;
    countmodel negbin(0.75, 5);
    simulatedsymbol npolicy ~ lognormal(10,0.35);
    tloss = _sev_ * npolicy;
run;
```

The COUNTMODEL statement specifies that the frequency follows the Negbin(0.75, 5) distribution. The SIMULATEDSYMBOL statement specifies the stochastic variable `nPolicy`. The severity adjustment program contains just one statement in this example, and it computes `tLoss` by multiplying the average severity value in the `_SEV_` placeholder symbol by the simulated value of `nPolicy`. PROC CCDM supplies the simulated values of both the `_SEV_` and `nPolicy` symbols to your adjustment program. In the pure premium simulation mode, for each replication, PROC CCDM multiplies the `tLoss` value that your adjustment program returns by the frequency value that it draws from the Negbin(0.75, 5) distribution to compute the final value of `tLoss` for that replication. The distribution of `tLoss` is summarized in the tables shown in [Output 7.4.4](#).

Output 7.4.4 Summary Statistics and Percentiles of the Total Loss Distribution

The CCDM Procedure

Severity Model: Weibull
Count Model: NegativeBinomial(0.75,5)

Summary Statistics for tloss			
Mean	17380429	Median	12051182
Standard Deviation	19869773	Interquartile Range	21118588
Variance	3.948079E14	Minimum	0
Skewness	2.42533	Maximum	422595056
Kurtosis	10.71788	Sample Size	500000

Percentiles for tloss	
Percentile	Value
1	0
5	0
25	3132245.7
50	12051182
75	24250834
95	55453569
99	91066951
99.5	108199963
Percentile Method = 5	

If you want to further account for the loss adjustment expenses (LAE) in your simulations and if you have

estimated that the average LAE per policy follows a normal distribution $\mathcal{N}(100, 10)$, then the total loss and expense, denoted by `tLossExp`, is computed as `tLossExp = ((severity/claim) * (claims/policy) + LAE)*nPolicy`. Because this does not fit the standard pure premium formula, you need to use the custom simulation mode by specifying the `SIMULATIONMODE=CUSTOM` option in the `PROC CCDM` statement as illustrated by the following statements:

```
proc ccdm severitystore=mylib.sevstore simulationmode=custom nperturb=50
    seed=135 nreplicates=500000 print=all adjustedseverity=(tloss tlossexp)
    plot(nadjsev=2)=(conditionaldensity(leftq=0.2 rightq=0.9));
    severitymodel weibull;
    countmodel negbin(0.75(se=0.05), 5(se=0.5));
    simulatedsymbol npolicy ~ lognormal(10,0.35) lae ~ normal(100,10);
    tloss = _sev_ * _freq_ * npolicy;
    tlossexp = (_sev_ * _freq_ + lae) * npolicy;
run;
```

The `SIMULATEDSYMBOL` statement now contains the specification for the stochastic variable LAE. For the custom simulation mode, `PROC CCDM` analyzes only the symbols that you specify in the `ADJUSTED-SEVERITY=` option; unlike collective risk or pure premium simulation modes, it does not create a sample for the unadjusted aggregate loss. However, you can compute the pure premium loss in the custom mode yourself as illustrated by the programming statement that computes `tLoss` in the preceding `PROC CCDM` step.

For each replication, `PROC CCDM` draws one random value each from the probability distributions of severity, frequency, LAE, and `nPolicy`, and it supplies those values to your severity adjustment program in appropriate symbols (the `_FREQ_` and `_SEV_` symbols contain frequency and severity values, respectively). The values of `tLoss` and `tLossExp` symbols that your severity adjustment program returns are added to the `tLoss` and `tLossExp` samples, respectively. The summary of the `tLossExp` sample thus generated is shown in [Output 7.4.5](#). As expected, the mean, median, and percentile estimates of total loss and expenses are higher than the corresponding estimates for the total loss distribution in [Output 7.4.4](#). The conditional PDF plot in [Output 7.4.6](#) shows the nonparametric estimate of the density function in different ranges of values of `tLossExp`. The left-tail plot indicates that there are at least 20% zero-valued observations in the sample of `tLossExp`, which implies that many policies do not result in a loss event. The right-tail plot starts at the 90th percentile and the density is nonzero for large losses, which implies that the distribution of `tLossExp` has a heavy tail.

Output 7.4.5 Summary Statistics and Percentiles of the Total Loss and Expenses Distribution

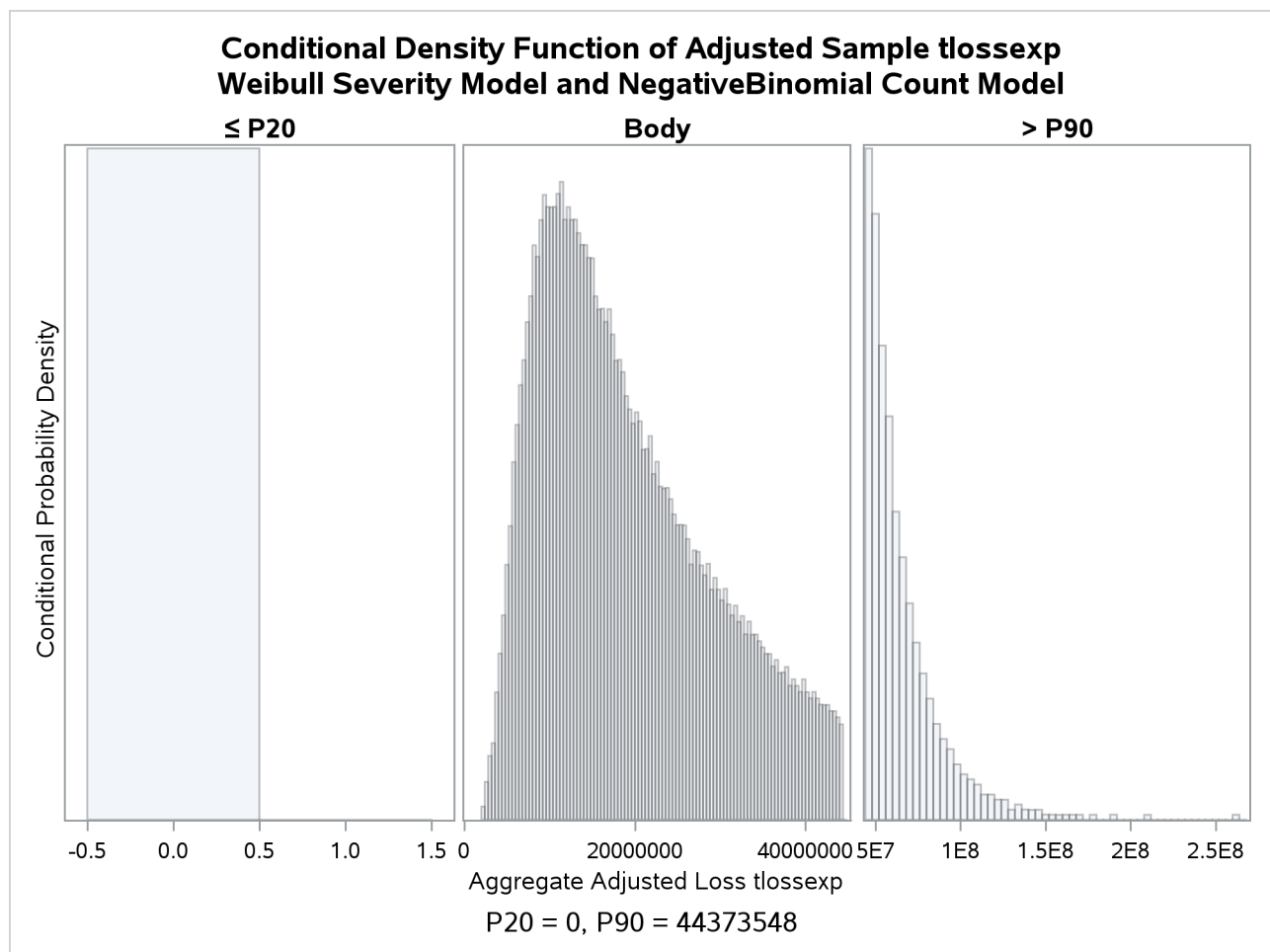
The CCDM Procedure

Severity Model: Weibull
Count Model: NegativeBinomial(0.75,5)

Summary Statistics for tlossexp			
Mean	19176593	Median	14247865
Standard Deviation	20638306	Interquartile Range	22132642
Variance	4.259397E14	Minimum	0
Skewness	2.24320	Maximum	421986852
Kurtosis	9.39176	Sample Size	500000

Output 7.4.5 *continued*

Percentiles for tlossexp	
Percentile	Value
1	0
5	0
20	0
25	4681927.7
50	14247865
75	26814570
90	44373548
95	58477496
99	94131961
99.5	111374899
Percentile Method = 5	

Output 7.4.6 Nonparametric Density Estimate of the Total Loss and Expenses Distribution

The NPERTURB= option in the preceding PROC CCDM step requests perturbation analysis, which requires

the variability estimates of distribution parameters. For the severity distribution, PROC CCDM reads the variability estimates from the SEVERITYSTORE= item store. For the externally estimated count distribution, you need to specify the standard errors of its parameters by using the SE= option after each parameter value in the COUNTMODEL statement, as shown in the preceding PROC CCDM step. For each perturbed sample, PROC CCDM uses perturbed values of the severity and frequency distribution parameters, but it does not perturb the parameters of the simulated symbols. The perturbation summary for the total loss and expense distribution is shown in [Output 7.4.7](#).

Output 7.4.7 Perturbation Analysis of the Total Loss and Expenses Distribution

Sample Perturbation Analysis for tlossexp		
Statistic	Estimate	Standard Error
Mean	18835327	4227131.4
Standard Deviation	20397977	3168516.2
Variance	4.26117E14	1.29497E14
Skewness	2.29054	0.03592
Kurtosis	10.01735	0.53879
Number of Perturbed Samples = 50		
Size of Each Sample = 500000		

Percentile Perturbation Analysis for tlossexp		
Percentile	Estimate	Standard Error
1	0	0
5	0	0
20	1753376.5	2847569.7
25	3549576.1	3834518.9
50	13963172	3189204.1
75	26367356	5316166.3
90	43627022	8129759.3
95	57577445	10306651
99	93220090	15798459
99.5	110085244	18265290
Number of Perturbed Samples = 50		
Size of Each Sample = 500000		

References

- Fisher, R. A. (1973). *Statistical Methods for Research Workers*. 14th ed. New York: Hafner Publishing.
- Klugman, S. A., Panjer, H. H., and Willmot, G. E. (1998). *Loss Models: From Data to Decisions*. New York: John Wiley & Sons.

Chapter 8

CCOPULA Procedure

Contents

Overview: CCOPULA Procedure	286
PROC CCOPULA Features	286
PROC CCOPULA Compared with Other SAS Procedures	286
Using CAS Sessions and CAS Engine Librefs	287
Getting Started: CCOPULA Procedure	288
Syntax: CCOPULA Procedure	290
Functional Summary	291
PROC CCOPULA Statement	292
BY Statement	293
DEFINE Statement	293
DISPLAY Statement	294
DISPLAYOUT Statement	295
FIT Statement	296
SIMULATE Statement	300
VAR Statement	303
VIEWSTORE Statement	303
Details: CCOPULA Procedure	304
Sklar's Theorem	304
Dependence Measures	305
Data Processing and Fitting	306
Normal Copula	308
Student's t copula	309
Archimedean Copulas	311
Canonical Maximum Likelihood Estimation (CMLE)	317
Exact Maximum Likelihood Estimation (MLE)	318
Calibration Estimation	318
Nonlinear Optimization Options	319
Displayed Output	319
OUTPSEUDO= and OUTUNIFORM= Data Tables	321
ODS Table Names	321
ODS Graph Names	322
Examples: CCOPULA Procedure	323
Example 8.1: Simulating Default Times	323
Example 8.2: Copula Analysis of Stock Returns	326
Example 8.3: Copula Analysis of S&P 500 Sectors	330
References	334

Overview: CCOPULA Procedure

A multivariate distribution for a random vector contains a description of both the marginal distributions and their dependence structure. A copula approach to formulating a multivariate distribution provides a way to isolate the description of the dependence structure from the marginal distributions. A copula is a function that combines marginal distributions of variables into a specific multivariate distribution. All of the one-dimensional marginals in the multivariate distribution are the cumulative distribution functions of the factors. Copulas help perform large-scale multivariate simulation from separate models, each of which can be fitted using different, even nonnormal, distributional specifications.

The CCOPULA procedure enables you to fit multivariate distributions or copulas from a specified sample data table. You can do the following:

- estimate the parameters for a specified copula type
- simulate a given copula

PROC CCOPULA Features

The CCOPULA procedure enables you to simulate or fit (or both) a specified copula, and it supports the following types of copulas:

- normal copula
- t copula
- Clayton copula
- Gumbel copula
- Frank copula

PROC CCOPULA Compared with Other SAS Procedures

The CCOPULA procedure provides copula simulation and model fitting functionality comparable to that of the HPCOPULA and COPULA procedures in SAS/ETS software.

PROC CCOPULA Compared with the HPCOPULA Procedure

The functionality of the CCOPULA procedure closely resembles that of the HPCOPULA procedure, which is a high-performance procedure. The CCOPULA procedure is the next generation of the HPCOPULA procedure, and it was developed specifically for SAS Viya. Both procedures are designed to run on a cluster of machines that distribute the data and the computations. Both procedures perform computations in multiple threads on multiple machines.

The CCOPULA procedure provides all the functionality of the HPCOPULA procedure.

PROC CCOPULA Compared with the COPULA Procedure

The CCOPULA procedure is specifically designed to operate on SAS Viya and performs computations in multiple threads on multiple machines. By contrast, the COPULA procedure runs only in a single thread on a single machine.

This release of the CCOPULA procedure contains some, but not all, of the functionality present in the COPULA procedure. The COPULA procedure provides the following feature, which is not present in the CCOPULA procedure:

- hierarchical Archimedean copulas

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section [“Introduction to Shared Concepts”](#) on page 54 in Chapter 4, [“Shared Concepts.”](#)

Getting Started: CCOPULA Procedure

The following example illustrates the use of PROC CCOPULA. The data are daily returns on several major stocks. The main purpose of this example is to estimate the joint distribution of stock returns and then use this distribution to simulate a new sample of specified size.

Figure 8.1 shows the first 10 observations of the daily stock return data set.

Figure 8.1 First 10 Observations of Daily Returns

Obs	date	ret_msft	ret_ko	ret_ibm	ret_duk	ret_bp
1	01/03/2008	0.004182	0.010367	0.002002	0.003503	0.019114
2	01/04/2008	-0.027960	0.001913	-0.035861	-0.000582	-0.014536
3	01/07/2008	0.006732	0.023607	-0.010671	0.025611	0.017922
4	01/08/2008	-0.033435	0.004239	-0.024610	-0.002838	-0.016049
5	01/09/2008	0.029560	0.026680	0.007301	0.010814	-0.027078
6	01/10/2008	-0.003054	0.004441	0.016414	-0.001689	-0.004395
7	01/11/2008	-0.012255	-0.027346	-0.022546	-0.012408	-0.018473
8	01/14/2008	0.013958	0.008418	0.053857	0.003427	0.001166
9	01/15/2008	-0.011318	-0.010851	-0.010689	-0.017075	-0.040925
10	01/16/2008	-0.022587	-0.015021	-0.001955	0.002316	-0.021336

The following statements fit a normal copula to the returns data (by using the FIT statement) and create a new SAS item store that contains parameter estimates of the model along with a complete description of the model (including variable names, copula type, and so on). The VAR statement specifies the list of variables, which in this case are the daily returns of stocks from five large companies.

```
/* Copula estimation */
proc ccopula data = mylib.returns;
  var ret_ibm ret_msft ret_bp ret_ko ret_duk;
  fit normal / store=mylib.estimates_norm;
run;
```

The first table in Figure 8.2 shows some general information about the copula fitting procedure: the number of observations, the name of the input data table, the type of model, and the correlations matrix.

Figure 8.2 Copula Estimation: Fit Summary and Correlations Matrices

The CCOPULA Procedure

Model Fit Summary	
Number of Observations	603
Data Set	RETURNS
Copula Type	Normal
Fit Method	MLE
Marginals	Empirical
Empirical Approximation	Adaptive Binning
Adaptive Refinement Resolution	1000
Adaptive Interpolation Method	Linear
Number of Adaptive Iterations	4

Figure 8.2 *continued*

Pearson Correlations Matrix					
Variable	ret_ibm	ret_msft	ret_bp	ret_ko	ret_duk
ret_ibm	1.0000	0.6232	0.5294	0.4725	0.4902
ret_msft	0.6232	1.0000	0.5229	0.5015	0.4567
ret_bp	0.5294	0.5229	1.0000	0.3980	0.4378
ret_ko	0.4725	0.5015	0.3980	1.0000	0.5283
ret_duk	0.4902	0.4567	0.4378	0.5283	1.0000

Kendall Correlations Matrix					
Variable	ret_ibm	ret_msft	ret_bp	ret_ko	ret_duk
ret_ibm	1.0000	0.4283	0.3551	0.3133	0.3261
ret_msft	0.4283	1.0000	0.3503	0.3344	0.3020
ret_bp	0.3551	0.3503	1.0000	0.2606	0.2885
ret_ko	0.3133	0.3344	0.2606	1.0000	0.3544
ret_duk	0.3261	0.3020	0.2885	0.3544	1.0000

Spearman Correlations Matrix					
Variable	ret_ibm	ret_msft	ret_bp	ret_ko	ret_duk
ret_ibm	1.0000	0.6052	0.5116	0.4555	0.4729
ret_msft	0.6052	1.0000	0.5052	0.4841	0.4400
ret_bp	0.5116	0.5052	1.0000	0.3826	0.4214
ret_ko	0.4555	0.4841	0.3826	1.0000	0.5106
ret_duk	0.4729	0.4400	0.4214	0.5106	1.0000

Then, in the following statements, the `SIMULATE` statement specifies a copula named `COP`, and the `RESTORE=` option specifies that the item store `Estimates_Norm` be used as the source for the variable names, copula type, and parameters. The `NDRAWS=500` option in the `SIMULATE` statement generates 500 observations from the normal copula. The `OUTUNIFORM=` option specifies the name of the data table to contain the simulated sample with uniform marginal distributions. Note that this syntax does not require the `DATA=` option and that neither the `VAR` statement nor the `DEFINE` statement is needed.

```
/* Ccopula simulation of uniforms */
proc ccopula;
    simulate cop / restore=mylib.estimates_norm
                ndraws      = 500
                seed        = 1234
                outuniform  = mylib.simulated_uniforms;
run;
```

The simulated data are contained in the new data table, `mylib.Simulated_Uniforms`.

The preceding sequence of `PROC CCOPULA` usage—first fit, then simulate using estimated parameters—is a legitimate sequence but has a limitation in that the second `PROC CCOPULA` call does not generate the sample according to the empirical distribution of the raw data. It generates only marginally uniform series.

In the following statements, the `FIT` statement fits a t copula to the returns data and at the same time simulates the sample according to empirical marginal distributions. Because there is no `VAR` or `DEFINE` statement, `PROC CCOPULA` assumes that you want to initialize the simulation by using the information stored in an item store. Because the `RESTORE=` option is not used in the `SIMULATE` statement, `PROC CCOPULA` uses the item store that is created when the `FIT` statement is executed.

```

/* Ccopula estimation and simulation */
proc ccopula data = mylib.returns;
  var ret_ibm ret_msft ret_bp ret_ko ret_duk;
  fit T / store=mylib.estimated_t;
  simulate / ndraws = 1000
            seed    = 1234
            outuniform = mylib.simuni;
run;

```

The output of the statements is similar in structure to the output displayed in Figure 8.2, with the addition of parameter estimates and inference statistics that are specific to the copula model as shown in Figure 8.3. For a t copula, the degrees of freedom are displayed (as in Figure 8.3); for Archimedean copulas, the parameter “theta” is displayed; and for a normal copula, this table is not printed.

Figure 8.3 Copula Estimation: Specific Parameter Estimates

The CCOPULA Procedure

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Approx Pr > t
DF	3.659299	0.320474	11.42	<.0001

The simulated data are contained in the new data table, mylib.Simuni.

Syntax: CCOPULA Procedure

The following statements are available in the CCOPULA procedure:

```

PROC CCOPULA options ;
  BY variables ;
  DEFINE name copula-type <(parameter-value-options ...) > ;
  DISPLAY <table-list> </ options > ;
  DISPLAYOUT table-spec-list </ options > ;
  FIT type <NAME=name > <INIT=(parameter-value-options) > / options ;
  SIMULATE < copula-name-list > / options ;
  VAR variables ;
  VIEWSTORE / options ;

```

Functional Summary

Table 8.1 summarizes the statements and options that the CCOPULA procedure uses.

Table 8.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CCOPULA	DATA=
Specifies the input data table that contains the correlations matrix for elliptical copulas	DEFINE	CORR=
Specifies the input data table that contains the correlations matrix defined in Kendall's tau for elliptical copulas	DEFINE	KENDALL=
Specifies the input data table that contains the correlations matrix defined in Spearman's rho for elliptical copulas	DEFINE	SPEARMAN=
Copula Fitting Options		
Specifies the method to use	FIT	METHOD=
Specifies the input data table that contains the correlations matrix for elliptical copulas	FIT	CORR=
Specifies the parameter value for Archimedean copulas	FIT	THETA=
Specifies the degrees of freedom for t copulas	FIT	DF=
Declaring the Role of Variables		
Specifies the names of the variables to use in copula fitting or in simulation	VAR	
Specifies BY-group processing	BY	
Options to Control the Optimization Process		
Selects the iterative minimization method to use	FIT	OPTMETHOD=
Specifies the maximum number of iterations for the optimization when you are fitting a t or Archimedean copula and METHOD=MLE	FIT	MAXITER=
Sets initial values for parameters	FIT	INIT=
Copula Estimation Options		
Specifies the marginal distribution of the individual variables	FIT	MARGINALS=
Copula Simulation Options		
Specifies the random sample size	SIMULATE	NDRAWS=
Specifies the random number generator seed	SIMULATE	SEED=

Description	Statement	Option
Output Control Options		
Specifies the output data table to contain the normalized pseudo-data used for model fitting	FIT	OUTPSEUDO=
Specifies the output item store to preserve the properties of the copula that is fit and the results of fitting the copula	FIT	STORE=
Specifies the output data table to contain the random samples from the simulation with uniform marginal distribution	SIMULATE	OUTUNIFORM=
Prints timing information	PROC CCOPULA	PRINTTIMING
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	

PROC CCOPULA Statement

PROC CCOPULA <options> ;

The following *options* can be used in the PROC CCOPULA statement.

Input Data Table Options

DATA=*libref.data-table*

names the input data table for PROC CCOPULA to use. The default is the most recently created data table. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 287.

data-table specifies the name of the input data table.

When the procedure is used for simulation only, the input data table is not required in order to run the procedure. However, when you specify the OUTEMPIRICAL= option in the SIMULATE statement, or the DATATYPE=ORIGINAL plotting option in the SIMULATE statement, the input data table determines the marginal distributions for the output variables.

Printing Options

PRINTTIMING

prints a timing report.

BY Statement

BY *variables* ;

The BY statement specifies groups in which separate copula fitting analyses or simulations are performed. When you use the BY statement with the FIT statement, the *variables* must be present in the input data table and are excluded from the model fitting. You must also provide a VAR statement. The BY statement can also be used when simulating a copula. When you use the BY statement with the SIMULATE statement, you must specify an item store in the RESTORE= option. The item store must contain model fitting results for the same BY groups that are defined by the *variables* in your BY statement.

DEFINE Statement

DEFINE *name copula-type* <(parameter-value-options ...) > ;

The DEFINE statement specifies the relevant information about the copula that is used for the simulation. You can specify the following arguments:

name specifies the name of the copula definition. You can use this *name* later in the SIMULATE statement.

copula-type specifies the type of copula. You must specify one of the following copula types, which are described in the section “[Details: CCOPULA Procedure](#)” on page 304:

NORMAL	fits the normal copula.
T	fits the <i>t</i> copula.
CLAYTON	fits the Clayton copula.
FRANK	fits the Frank copula.
GUMBEL	fits the Gumbel copula.

parameter-value-options

specify the input parameters that are used to simulate the specified copula. These options must be appropriate for the type of copula specified. You can specify the following *parameter-value-options*:

CORR=*libref.SAS-data-table*

specifies the data table that contains the correlations matrix to use for elliptical copulas. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 287. All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. You can use this option for normal and *t* copulas.

DF=value

specifies the degrees of freedom. You can use this option for t copulas.

KENDALL=libref.SAS-data-table

specifies the data table that contains the correlations matrix defined in Kendall's tau. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 287.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row n contains the correlations between the n th variable in the VAR statement and the other variables. You can use this option for normal and t copulas.

SPEARMAN=libref.SAS-data-table

specifies the data table that contains the correlations matrix defined in Spearman's rho. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 287.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row n contains the correlations between the n th variable in the VAR statement and the other variables. You can use this option for normal copulas.

THETA=value

specifies the parameter value for the Archimedean copulas.

The DEFINE statement is used with the SIMULATE statement and the FIT statement.

DISPLAY Statement

DISPLAY < table-list > < / options > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 321. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 321. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

FIT Statement

FIT *type* <**NAME**=*name*><**INIT**=(*parameter-value-options*)> / *options* ;

The FIT statement estimates the parameters for a specified copula type.

You must specify a *type*:

type

specifies the type of the copula to be estimated, which is one of the following:

CLAYTON	fits the Clayton copula.
FRANK	fits the Frank copula.
GUMBEL	fits the Gumbel copula.
NORMAL	fits the normal copula.
T	fits the <i>t</i> copula.

You can also specify the following options:

INIT=(*parameter-value-options*)

provides the initial values for the numerical optimization.

[*parameter-value-options*]

specify the input parameters that are used to initialize the specified copula. These options must be appropriate for the type of copula that you specify. You can specify the following options:

CORR=*libref.SAS-data-table*

specifies the data table that contains the correlations matrix to use for elliptical copulas. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 287.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row *n* contains the correlations between the *n*th variable in the VAR statement and the other variables. You can use this option for *t* copulas.

DF=*value*

specifies the degrees of freedom. You can use this option for t copulas.

KENDALL=*libref.SAS-data-table*

specifies the data table that contains the correlations matrix defined in Kendall's tau. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 287.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row n contains the correlations between the n th variable in the VAR statement and the other variables. You can use this option for t copulas.

SPEARMAN=*libref.SAS-data-table*

specifies the data table that contains the correlations matrix defined in Spearman's rho. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 287.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row n contains the correlations between the n th variable in the VAR statement and the other variables. You can use this option for t copulas.

THETA=*value*

specifies the parameter value for Archimedean copulas.

For Archimedean copulas, the default initial values of the parameter are computed using the calibration method. The default initial value of the degrees-of-freedom parameter in the t copula is set to 2.0. The following statement shows an initialization for Student's t copula, where the Kendall's tau correlations matrix is stored in the mylib.corrmat data table and the DF is set to 2.5:

```
fit t init=(df=2.5 kendall=mylib.corrmat);
```

NAME=*name*

specifies an identifier for the fit.

You can specify the following *options* after a slash (/):

MARGINALS=EMPIRICAL<(*approx-options*)> | **UNIFORM**

specifies the marginal distribution of the individual variables. You can specify the following values:

EMPIRICAL<(*approx-options*)> uses an approximation of the marginal empirical cumulative distribution function (CDF) to transform the data and uses the transformed data to fit the copula. You can specify the following *approx-options*:

ALG=BIN | SORT

specifies the algorithm to use during the approximation of the marginal distribution. You can specify the following values:

BIN	uses bins to approximate the marginal distributions.
SORT	stores and sorts each variable's observations in the input data table.

By default, ALG=BIN.

INTERP=LINEAR | SPLINE | STEP

specifies how to interpolate values within intervals in the distributions' approximations. You can specify the following values:

LINEAR	specifies that values within an interval be linearly interpolated between the endpoints.
SPLINE	specifies that values within an interval be interpolated using a cubic spline function that is constrained to be monotonically increasing.
STEP	specifies that values within an interval take the value of the left endpoint.

By default, INTERP=LINEAR.

MAXITERS=*integer*

specifies the maximum number of iterations to perform when you are generating the distribution approximation function. By default, MAXITERS=10. For more information, see the section "[Adaptive Approximation of Marginal Distributions](#)" on page 307.

REFINERES=*integer*

specifies the maximum number of intervals to add to the distribution function during each iteration of the empirical approximation. By default, REFINERES=1000.

UNIFORM uses the input data without transformation to fit the copula.

METHOD=MLE | CAL

specifies the method to use to estimate parameters. You can specify the following values:

CAL	specifies the calibration method that uses the correlations matrix (only Kendall's tau is implemented in this procedure).
MLE	specifies canonical maximum likelihood estimation (CMLE) or maximum likelihood estimation (MLE).

For the t copula, if METHOD=CAL, then the correlations matrix is estimated using the calibration method with Kendall's tau and the degrees of freedom are estimated by the MLE. For the normal copula, only METHOD=MLE is supported and METHOD=CAL is ignored. By default for all copula types, METHOD=MLE.

OUTPSEUDO=libref.data-table

specifies the output data table for saving the pseudo-samples with uniform marginal distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 287.

The pseudo-samples are obtained by transforming the individual variables of the original data by using the empirical CDFs. The data table is not created if you omit this option.

PLOTS<(global-plot-options) > <=(specific-plot-options)>

controls the plots of the input data table that are produced by the CCOPULA procedure. By default, PROC CCOPULA produces a scatter plot matrix for variables (that is, it displays a symmetric matrix plot that contains the variables specified in the VAR statement).

You can specify the following *global-plot-options*:

KENDALL

creates Kendall plots. These plots compare observed quantiles of pairs of variables to quantiles of independent variables. If you specify this option together with the UNPACK option, PROC CCOPULA displays a Kendall plot of each applicable pair of distinct variables that you specify in the VAR statement. If you specify this option without the UNPACK option, PROC CCOPULA displays a scatter plot matrix; the lower triangular section of the matrix shows regular scatter plots between distinct pairs of variables that are specified in the VAR statement, and the upper triangular section of the matrix shows Kendall plots of corresponding pairs of variables.

NSAMPLES=*n*

specifies the number points to plot when you create Kendall plots. For each pair of variables, *n* samples are randomly selected from the DATA= data table. The same samples are plotted for all the Kendall plots. By default, NSAMPLES=1000.

NVAR=ALL | *n*

specifies the maximum number of variables that you specify in the VAR statement to display in the matrix plot. NVAR=ALL uses all variables that you specify in the VAR statement. By default, NVAR=5.

RESOLUTION=*n*

specifies the number of bins to use for the X and Y axes when you create plots. By default, RESOLUTION=50.

SCATTER

creates a scatter plot matrix of variables that you specify in the VAR statement. The maximum number of variables that you can display in the scatter plot matrix is 10.

TAIL | CHI

creates tail dependence plots (chi plots). If you specify this option together with the UNPACK option, PROC CCOPULA displays a chi plot of each applicable pair of distinct variables that you specify in the VAR statement. If you specify this option without the UNPACK option, PROC CCOPULA displays a scatter plot matrix; the lower triangular section of the matrix shows regular scatter plots between distinct pairs of variables that are specified in the VAR statement, and the upper triangular section of the matrix shows chi plots of corresponding pairs of variables.

UNPACK | UNPACKPANEL

creates scatter plots of pairs of variables. If you specify this option, PROC CCOPULA displays a scatter plot of each applicable pair of distinct variables that you specify in the VAR statement.

You can also specify the following *specific-plot-options*:

DATATYPE=ORIGINAL | UNIFORM | BOTH

specifies the data type to be plotted. DATATYPE=ORIGINAL presents the data in their original marginal distribution. DATATYPE=UNIFORM shows the transformed data with a uniform marginal distribution. DATATYPE=BOTH plots both the original and uniform data types. When MARGINALS=UNIFORM, the transformation is omitted and the DATATYPE= option is ignored.

NONE

suppresses all plots.

STORE=SAS-item-store

specifies the item store that preserves the properties of the model and the fit results.

Printing Options

ITPRINT

prints a summary iteration listing.

NOCORR

suppresses the correlations matrix.

NOPRINT

suppresses all output.

PRINTALL

prints all output. This is the default.

SIMULATE Statement

SIMULATE < *copula-name-list* > / *options* ;

The SIMULATE statement simulates data from a specified copula model. The copula name specification is the name of a defined copula as specified by *name* in the DEFINE statement. You can specify the following *options*:

EMPAPPROX< (*approx-options*) >

uses an approximation of the inverse marginal empirical cumulative distribution function to transform the data when you specify the OUTEMPIRICAL= output option. You can specify the following *approx-options*:

ALG=BIN | SORT

specifies the algorithm to use during the approximation of the marginal distribution. You can specify the following values:

BIN	uses bins to approximate the marginal distributions.
SORT	stores and sorts each variable's observations in the input data table.

By default, ALG=BIN.

INTERP=LINEAR | SPLINE

specifies how to interpolate values within intervals in the distributions' approximations. You can specify the following values:

LINEAR	specifies that values within an interval be linearly interpolated between the endpoints.
SPLINE	specifies that values within an interval be interpolated using a cubic spline function that is constrained to be monotonically increasing.

By default, INTERP=LINEAR.

MAXITERS=*integer*

specifies the maximum number of iterations to perform when you are generating the distribution approximation function. By default, MAXITERS=10. For more information, see the section [“Adaptive Approximation of Marginal Distributions”](#) on page 307.

REFINERES=*integer*

specifies the maximum number of intervals to add to the distribution function during each iteration of the empirical approximation. By default, REFINERES=1000.

NDRAWS=*integer*

specifies the number of draws to generate for this simulation. By default, NDRAWS=100.

OUTEMPIRICAL=*libref.data-table*

specifies the output data table to contain the result of the simulation that uses empirical margins. The distribution that is used to transform each variable's uniformly simulated value to its empirical value is specified by the corresponding variable in the DATA= input data table in the PROC CCOPULA statement. The data are not created if you omit this option. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section [“Using CAS Sessions and CAS Engine Librefs”](#) on page 287.

OUTUNIFORM=*libref.data-table*

specifies the output data table to contain the result of the simulation in uniform margins. The data are not created if you do not specify this option. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section [“Using CAS Sessions and CAS Engine Librefs”](#) on page 287.

PLOTS<(global-plot-options)> <= (specific-plot-options)>

controls the plots of simulated data that are produced by the CCOPULA procedure. By default, PROC CCOPULA produces a scatter plot matrix of variables (that is, it displays a symmetric matrix plot that contains the variables specified in the VAR statement).

You can specify the following *global-plot-options*:

KENDALL

creates Kendall plots. These plots compare observed quantiles of pairs of variables to quantiles of independent variables. If you specify this option together with the UNPACK option, PROC CCOPULA displays a Kendall plot of each applicable pair of distinct variables that you specify in the VAR statement. If you specify this option without the UNPACK option, PROC CCOPULA displays a scatter plot matrix; the lower triangular section of the matrix shows regular scatter plots between distinct pairs of variables that are specified in the VAR statement, and the upper triangular section of the matrix shows Kendall plots of corresponding pairs of variables.

NSAMPLES=*n*

specifies the number points to plot when you create Kendall plots. For each pair of variables, *n* samples are randomly selected from the OUTUNIFORM= data table. The same samples are plotted for all the Kendall plots. By default, NSAMPLES=1000.

NVAR=ALL | *n*

specifies the maximum number of variables that you specify in the VAR statement to display in the matrix plot. NVAR=ALL uses all variables that you specify in the VAR statement. By default, NVAR=5.

RESOLUTION=*n*

specifies the number of bins to use for the X and Y axes when you create plots. By default, RESOLUTION=50.

SCATTER

creates a scatter plot matrix of variables that you specify in the VAR statement. The maximum number of variables that you can display in the scatter plot matrix is 10.

TAIL | **CHI**

creates tail dependence plots (chi plots). If you specify this option together with the UNPACK option, PROC CCOPULA displays a chi plot of each applicable pair of distinct variables that you specify in the VAR statement. If you specify this option without the UNPACK option, PROC CCOPULA displays a scatter plot matrix; the lower triangular section of the matrix shows regular scatter plots between distinct pairs of variables that are specified in the VAR statement, and the upper triangular section of the matrix shows chi plots of corresponding pairs of variables.

UNPACK | **UNPACKPANEL**

creates scatter plots of pairs of variables. If you specify this option, PROC CCOPULA displays a scatter plot of each applicable pair of distinct variables that you specify in the VAR statement.

You can also specify the following *specific-plot-options*:

DATATYPE=UNIFORM

shows the data by using uniformly distributed margins. Plotting data by using the empirical marginal distribution specified in the DATA= data table is not supported in this release.

DATATYPE=ORIGINAL | UNIFORM | BOTH

specifies the data type to be plotted. DATATYPE=ORIGINAL plots the data transformed into the marginal distributions specified in the DATA= option in the PROC CCOPULA statement. DATATYPE=UNIFORM shows the data with a uniform marginal distribution. DATATYPE=BOTH plots both the original and uniform data types.

NONE

suppresses all plots.

RESTORE=SAS-item-store

specifies the input item store that contains the results of fitting a copula. The final estimates, correlations, and other properties of the model that are preserved in the RESTORE= item store are used to define the simulation. The DEFINE and VAR statements are ignored if this option is specified.

SEED=integer

specifies the seed for generating random numbers for the simulation. If you do not specify the seed, then the seed is generated by reading the time of day from the computer's clock.

TOLERANCE=value

specifies the tolerance allowed for the simulation.

VAR Statement

VAR *variables* ;

The VAR statement specifies the variable names in the input data table that is specified by the DATA= option in the PROC CCOPULA statement. The subset of variables in the data table is used for the copula models in the FIT statement. If there is no input data table, the VAR statement creates the list of variable names for the SIMULATE statement.

VIEWSTORE Statement

VIEWSTORE / *options* ;

The VIEWSTORE statement enables you to print various reports that describe the model fitting information that is preserved in an item store. The item store must have been created using the STORE= option in the FIT statement. You can specify the following *options*:

ALL

prints the entire contents of the item store.

BYVARVALUES

prints the values of the BY-group variables that are used to fit the model.

CORRELATIONS

prints the correlations matrices.

FINALESTIMATES

prints the final parameter estimates.

FITMODELSUMMARY

prints the fit model summary.

INITIALESTIMATES

prints the initial parameter estimates.

INSTORE=SAS-item-store

specifies the item store that you want to view. If you omit this option, and if you use a FIT statement to specify a STORE= item store, then the STORE= item store that is specified in the FIT statement is used as the INSTORE= item store.

MINIMAL

prints only the fit model summary and the solution.

MODELDEFINITION

prints the statements that define the model.

NONE

suppresses all reports.

OPTIMIZERSETTINGS

prints the optimizer settings that are used to fit the model.

Details: CCOPULA Procedure

Sklar's Theorem

The copula models are tools for studying the dependence structure of multivariate distributions. The usual joint distribution function contains the information both about the marginal behavior of the individual random variables and about the dependence structure between the variables. The copula is introduced to decouple the marginal properties of the random variables and the dependence structures. An m -dimensional *copula* is a joint distribution function on $[0, 1]^m$, where all marginal distributions are standard uniform. The common notation for a copula is $C(u_1, \dots, u_m)$.

The Sklar (1959) theorem shows the importance of copulas in modeling multivariate distributions. The first part of the theorem states that a copula can be derived from any joint distribution functions, and the second part asserts the opposite: that any copula can be combined with any set of marginal distributions to result in a multivariate distribution function. The theorem follows:

- Let F be a joint distribution function, and let $F_j, j = 1, \dots, m$, be the marginal distributions. Then there exists a copula $C : [0, 1]^m \rightarrow [0, 1]$ such that

$$F(x_1, \dots, x_m) = C(F_1(x_1), \dots, F_m(x_m))$$

for all $x_1, \dots, x_m$ in $[-\infty, \infty]$. Moreover, if the margins are continuous, then C is unique; otherwise C is uniquely determined on $\text{Ran} F_1 \times \dots \times \text{Ran} F_m$, where $\text{Ran} F_j = F_j([-\infty, \infty])$ is the range of F_j .

- The converse is also true. That is, if C is a copula and $F_1, \dots, F_m$ are univariate distribution functions, then the multivariate function that is defined in the preceding equation is a joint distribution function with marginal distributions $F_j, j = 1, \dots, m$.

Dependence Measures

There are three basic types of dependence measures: linear correlation, rank correlation, and tail dependence. Linear correlation is given by

$$\rho \equiv \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sqrt{\text{var}(X)}\sqrt{\text{var}(Y)}}$$

The linear correlation coefficient contains very limited information about the joint properties of the variables. A well-known property is that zero correlation does not imply independence, whereas independence implies zero correlation. In addition, there are distinct bivariate distributions that have the same marginal distribution and the same correlation coefficient. These results suggest that caution must be used in interpreting the linear correlation.

Another statistical measure of dependence is rank correlation, which is nonparametric. For example, Kendall's tau is the covariance between the sign statistics $X_1 - \tilde{X}_1$ and $X_2 - \tilde{X}_2$, where $(\tilde{X}_1, \tilde{X}_2)$ is an independent copy of (X_1, X_2) :

$$\rho_\tau \equiv E[\text{sign}(X_1 - \tilde{X}_1)(X_2 - \tilde{X}_2)]$$

The sign function (sometimes written as sgn) is defined as

$$\text{sign}(x) = \begin{cases} -1 & \text{if } x \leq 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

Spearman's rho is the correlation between the transformed random variables:

$$\rho_S(X_1, X_2) \equiv \rho(F_1(X_1), F_2(X_2))$$

The variables are transformed by their distribution functions so that the transformed variables are uniformly distributed on $[0, 1]$. The rank correlations depend only on the copula of the random variables and are indifferent to the marginal distributions. Like linear correlation, rank correlation has its limitations. In particular, different copulas result in the same rank correlation.

A third measure, tail dependence, focuses on only part of the joint properties between the variables. Tail dependence measures the dependence when both variables have extreme values. Formally, they can be defined as the conditional probabilities of quantile exceedances. There are two types of tail dependence:

- Upper tail dependence is defined as

$$\lambda_u(X_1, X_2) \equiv \lim_{q \rightarrow 1^-} P(X_2 > F_2^{-1}(q) | X_1 > F_1^{-1}(q))$$

when the limit exists and $\lambda_u \in [0, 1]$. Here F_j^{-1} is the quantile function (that is, the inverse of the CDF).

- Lower tail dependence is defined symmetrically.

Tail dependence is hard to detect by looking at a scatter plot of realizations of two random variables. One graphical way to detect tail dependence between two variables is by creating the chi plot of those two variables. The chi plot, as defined in Fisher and Switzer (2001), has characteristic patterns that depend on the dependence structure between the variables. The chi plot of the random variables X and Y is a scatter plot of the pairs (λ_i, χ_i) for each data point (x_i, y_i) . λ_i is a measure of the distance of the data point (x_i, y_i) from the center of the data as measured by the median values of (x_i, y_i) , and χ_i is a correlation coefficient between dichotomized values of X and Y . A positive λ_i means that x_i and y_i are either both large with respect to their median values or both small. A negative λ_i means that x_i or y_i is large with respect to its median and that the other value is small. Signs of tail dependence manifest as clusters of points that are significantly far from the χ axis around λ values of ± 1 . If X and Y are uncorrelated, the χ values cluster around the λ axis.

A Kendall plot enables you to compare the multivariate cumulative distribution function (CDF) for observed data to the CDF for a reference distribution. The Kendall plot is a multidimensional analog to the Q-Q plot that is used to compare distributions in one dimension. In both Q-Q plots and Kendall plots, you visually assess the similarity between the distributions of two random samples, A and B , by creating a scatter plot of the ordered values of A versus B . In the unpacked version of Kendall plots output by the CCOPULA procedure, three reference curves are displayed in addition to the observed, or simulated data. The horizontal reference curve is a Kendall plot for variables with a correlation of -1. The diagonal curve is Kendall plot for an uncorrelated bivariate uniform distribution. The arced curve is a Kendall plot for variables with a correlation of +1.

Data Processing and Fitting

Before you can fit a copula, you must transform the data into pseudo-uniform data. If you specify the MARGINALS=UNIFORM option, then it is assumed that the data are already in the proper form, so the required transformation is the trivial identity transformation. However, if you specify MARGINALS=EMPIRICAL (or if you omit the MARGINALS= option), then the transformation is nontrivial.

For relatively small data tables, each observation in the original input data table can be transformed into a pseudo-uniform observation by using a simple value-comparison algorithm. In this case, the initial estimates of the parameters in t copulas and Archimedean copulas can be derived by computing a Kendall correlations matrix on the basis of the transformed data.

However, for larger data tables, the simple value-comparison algorithm becomes computationally impractical as a means of transforming the original data into pseudo-uniform data. Thus, a data binning algorithm is used to transform the original data into pseudo-uniform data. After the data have been transformed in this way, it can still be computationally impractical to compute a Kendall correlations matrix directly from the data, because the algorithm for deriving the Kendall correlations matrix requires a pairwise comparison of every

pair of observations. Therefore, after the data have been transformed via binning, a Pearson correlations matrix is computed on the basis of the transformed data, and then a Kendall correlations matrix is derived from the Pearson correlations matrix by applying an analytical formula. Then the initial estimates for the parameters in the t copulas and Archimedean copulas are derived from the Kendall correlations matrix.

Adaptive Approximation of Marginal Distributions

Simple binning into fixed-width intervals of each variable's empirical data can lead to a poor approximation of distributions for variables that have highly nonsmooth distributions. For smaller data tables, fixed-interval bins can be an inefficient representation of the empirical distributions because many bins are empty.

To accommodate variables that have a wide range of empirical distributions, PROC CCOPULA uses an adaptive binning algorithm to generate an approximation of each variable's cumulative distribution function (CDF). The following steps summarize how an approximate CDF is computed for each variable:

1. A partitioning of the equal-width intervals specified in the REFINERES= option of the FIT statement between the minimum and maximum values is created so that each value falls into one interval.
2. The number of values that fall within each interval is counted.
3. Each interval is assigned a weight on the basis of its count.
4. Each interval is subdivided into a number of subintervals proportional to its weight so that the total number of new subdivisions across all intervals equals the value specified in the REFINERES= option of the FIT statement.
5. The process of subdividing intervals is repeated starting at step 2 until either the weight of each interval falls below a threshold value or the algorithm has run the number of times specified in the MAXITERS= option of the FIT statement.

After a final partitioning with counts has been created, the count that is associated with each interval is modified to include the cumulative counts for all smaller intervals.

The CDF for a particular value is computed by interpolating the counts between the endpoints of its interval, by using the method that you specify in the INTERP= option.

Two alternative approaches are available for counting the number of observations in step 2 of the approximation algorithm. When you specify ALG=BIN, the interval counts are computed by locating the interval associated with each observation and incrementing that interval's count. When you specify ALG=SORT, all the observations are stored in memory and sorted to make the assignment of observations to intervals faster in step 2.

The adaptive algorithm executes its steps in parallel across all available threads and machines except for step 5, in which the partitioning available on each thread must be merged and shared with all other threads.

Normal Copula

Let $u_j \sim U(0, 1)$ for $j = 1, \dots, m$, where $U(0, 1)$ represents the uniform distribution on the $[0, 1]$ interval. Let Σ be the correlations matrix, where $m(m-1)/2$ parameters satisfy the positive semidefiniteness constraint. The normal copula can be written as

$$C_{\Sigma}(u_1, u_2, \dots, u_m) = \Phi_{\Sigma}(\Phi^{-1}(u_1), \dots, \Phi^{-1}(u_m))$$

where Φ is the distribution function of a standard normal random variable and Φ_{Σ} is the m -variate standard normal distribution with mean vector 0 and covariance matrix Σ . That is, the distribution Φ_{Σ} is $N_m(0, \Sigma)$.

Simulation

For the normal copula, the input of the simulation is the correlations matrix Σ . The normal copula can be simulated by the following steps, in which $\mathbf{U} = (U_1, \dots, U_m)$ denotes one random draw from the copula:

1. Generate a multivariate normal vector $\mathbf{Z} \sim N(0, \Sigma)$, where Σ is an m -dimensional correlations matrix.
2. Transform the vector $\mathbf{Z}$ into $\mathbf{U} = (\Phi(Z_1), \dots, \Phi(Z_m))^T$, where Φ is the distribution function of univariate standard normal.

The first step can be achieved by Cholesky decomposition of the correlations matrix $\Sigma = LL^T$, where L is a lower triangular matrix with positive elements on the diagonal. If $\tilde{\mathbf{Z}} \sim N(0, I)$, then $L\tilde{\mathbf{Z}} \sim N(0, \Sigma)$.

Fitting

To fit a normal copula is to estimate the covariance matrix Σ from an input sample data table. Given a random sample $\mathbf{u}_i = (u_{i,1}, \dots, u_{i,m})^T$ where $i = 1, \dots, n$, the log-likelihood function is

$$\begin{aligned} \log L(\Sigma; \mathbf{u}_1, \dots, \mathbf{u}_n) \\ = \sum_{t=1}^n \log f_{\Sigma}(\Phi^{-1}(u_{t,1}), \dots, \Phi^{-1}(u_{t,m})) - \sum_{t=1}^n \sum_{j=1}^m \log \phi(\Phi^{-1}(u_{t,j})) \end{aligned}$$

Here f_{Σ} is the joint density of the multivariate normal with mean zero and variance Σ , and ϕ is the univariate density of the standard normal distribution. Note that the second term is not related to the parameters Σ and, therefore, can be ignored during the optimization. The restriction that Σ is a correlations matrix is very inconvenient, and it is common practice to circumvent this problem by first assuming that Σ has the covariance form. Therefore, Σ can be estimated by

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^n \xi_i \xi_i^T$$

where

$$\xi_i = (\Phi^{-1}(u_{i,1}), \Phi^{-1}(u_{i,2}), \dots, \Phi^{-1}(u_{i,m}))^T$$

This estimate is consistent with the form of a covariance matrix but not necessarily with the form of a correlations matrix. The approximation to the original MLE problem can be obtained using the normalizing operator defined as follows:

$$\Delta(\Sigma) = \text{diag}(\sigma_{11}^{1/2}, \dots, \sigma_{mm}^{1/2})$$

$$\mathcal{P}(\Sigma) = (\Delta(\Sigma))^{-1} \Sigma (\Delta(\Sigma))^{-1}$$

Student's t copula

Let $\Theta = \{(\nu, \Sigma) : \nu \in (1, \infty), \Sigma \in \mathbb{R}^{m \times m}\}$, and let t_ν be a univariate t distribution with ν degrees of freedom.

The Student's t copula can be written as

$$C_\Theta(u_1, u_2, \dots, u_m) = t_{\nu, \Sigma} \left(t_\nu^{-1}(u_1), t_\nu^{-1}(u_2), \dots, t_\nu^{-1}(u_m) \right)$$

where $t_{\nu, \Sigma}$ is the multivariate Student's t distribution that has a correlations matrix Σ with ν degrees of freedom.

Simulation

The input parameters for the simulation are (ν, Σ) . The t copula can be simulated by the following steps:

1. Generate a multivariate vector $\mathbf{X} \sim t_m(\nu, 0, \Sigma)$ that follows the centered t distribution with ν degrees of freedom and correlations matrix Σ .
2. Transform the vector $\mathbf{X}$ into $\mathbf{U} = (t_\nu(X_1), \dots, t_\nu(X_m))^T$, where t_ν is the distribution function of univariate t distribution with ν degrees of freedom.

To simulate centered multivariate t random variables, you can use the property that $\mathbf{X} \sim t_m(\nu, 0, \Sigma)$ if $\mathbf{X} = \sqrt{\nu/s} \mathbf{Z}$, where $\mathbf{Z} \sim N(0, \Sigma)$ and the univariate random variable $s \sim \chi_\nu^2$.

Fitting

To fit a t copula is to estimate the covariance matrix Σ and degrees of freedom ν from given multivariate data. Given a random sample $\mathbf{u}_i = (u_{i,1}, \dots, u_{i,m})^T$, $i = 1, \dots, n$ that has uniform marginal distributions, the log likelihood is

$$\log L(\nu, \Sigma; u_{i,1}, \dots, u_{i,m})$$

$$= \sum_{i=1}^n \log g_{\nu, \Sigma}(t_\nu^{-1}(u_{i,1}), \dots, t_\nu^{-1}(u_{i,m})) - \sum_{i=1}^n \sum_{j=1}^m \log g_\nu(t_\nu^{-1}(u_{i,j}))$$

where ν denotes the degrees of freedom of the t copula, $g_{\nu, \Sigma}$ denotes the joint density function of the centered multivariate t distribution with parameters (ν, Σ) , t_ν is the distribution function of a univariate t distribution with ν degrees of freedom, Σ is a correlations matrix, and g_ν is the density function of univariate t distribution with ν degrees of freedom.

The log likelihood can be maximized with respect to the parameters $\theta = (\nu, \Sigma) \in \Theta$ using numerical optimization. If you allow the parameters in Σ to be such that Σ is symmetric and with ones on the diagonal, then the MLE estimate for Σ might not be positive semidefinite. In that case, you need to apply the adjustment to convert the estimated matrix to positive semidefinite, as shown by McNeil, Frey, and Embrechts (2005), Algorithm 5.55.

When the dimension of the data m increases, the numerical optimization quickly becomes infeasible. It is common practice to estimate the correlations matrix Σ by calibration using Kendall's tau. Then, using this fixed Σ , the single parameter ν can be estimated by MLE. By proposition 5.37 in McNeil, Frey, and Embrechts (2005),

$$\rho_{\tau}(U_i, U_j) = \frac{2}{\pi} \arcsin \rho_{ij}$$

where ρ_{τ} is the Kendall's tau and ρ_{ij} is the off-diagonal elements of the correlations matrix Σ of the t copula. Therefore, an estimate for the correlation is

$$\hat{\rho}_{ij} = \sin \left(\frac{1}{2} \pi \hat{\rho}_{i,j}^{\tau} \right)$$

where $\hat{\rho}$ and $\hat{\rho}^{\tau}$ are the estimates of the sample correlations matrix and Kendall's tau, respectively. However, it is possible that the estimate of the correlations matrix $\hat{\Sigma}$ is not positive definite. In this case, there is a standard procedure that uses the eigenvalue decomposition to transform the correlations matrix into one that is positive definite. Let Σ be a symmetric matrix with ones on the diagonal, with off-diagonal entries in $[-1, 1]$. If Σ is not positive semidefinite, use Algorithm 5.55 from McNeil, Frey, and Embrechts (2005):

1. Compute the eigenvalue decomposition $\Sigma = EDE^T$, where D is a diagonal matrix that contains all the eigenvalues and E is an orthogonal matrix that contains the eigenvectors.
2. Construct a diagonal matrix $\tilde{D}$ by replacing all negative eigenvalues in D by a small value $\delta > 0$.
3. Compute $\tilde{\Sigma} = E\tilde{D}E^T$, which is positive definite but not necessarily a correlations matrix.
4. Apply the normalizing operator $\mathcal{P}$ on the matrix $\tilde{\Sigma}$ to obtain the correlations matrix desired.

The log-likelihood function and its gradient function for a single observation are listed as follows, where $\xi = (\xi_1, \dots, \xi_m)$, with $\xi_j = t_{\nu}^{-1}(u_j)$, and g is the derivative of the log Γ function:

$$\begin{aligned}
l = \log(c) &= -\frac{1}{2} \log(|\Sigma|) + \log \Gamma\left(\frac{\nu+m}{2}\right) + (m-1) \log \Gamma\left(\frac{\nu}{2}\right) - m \log \Gamma\left(\frac{\nu+1}{2}\right) \\
&\quad - \frac{\nu+m}{2} \log(1 + \zeta^T \Sigma^{-1} \zeta / \nu) + \frac{\nu+1}{2} \sum_{j=1}^m \log\left(1 + \frac{\zeta_j^2}{\nu}\right) \\
\frac{\partial l}{\partial \nu} &= \frac{1}{2} g\left(\frac{\nu+m}{2}\right) + \frac{m-1}{2} g\left(\frac{\nu}{2}\right) - \frac{m}{2} g\left(\frac{\nu+1}{2}\right) \\
&\quad - \frac{1}{2} \log(1 + \zeta^T \Sigma^{-1} \zeta / \nu) + \frac{\nu+m}{2\nu^2} \frac{\zeta^T \Sigma^{-1} \zeta}{1 + \zeta^T \Sigma^{-1} \zeta / \nu} \\
&\quad + \frac{1}{2} \sum_{j=1}^m \log(1 + \zeta_j^2 / \nu) - \frac{\nu+1}{2\nu^2} \sum_{j=1}^m \frac{\zeta_j^2}{1 + \zeta_j^2 / \nu} \\
&\quad - \frac{(\nu+m)}{\nu} \frac{\zeta^T \Sigma^{-1} (d\zeta/d\nu)}{1 + \zeta^T \Sigma^{-1} \zeta / \nu} + \frac{\nu+1}{\nu} \sum_{j=1}^m \frac{\zeta_j (d\zeta_j/d\nu)}{1 + \zeta_j^2 / \nu}
\end{aligned}$$

The derivative of the likelihood with respect to the correlations matrix Σ follows:

$$\begin{aligned}
\frac{\partial l}{\partial \Sigma} &= -\frac{1}{2} (\Sigma^{-1})^T + \frac{\nu+m}{2} \frac{\Sigma^{-T} \zeta \zeta^T \Sigma^{-T} / \nu}{1 + \zeta^T \Sigma^{-1} \zeta / \nu} \\
&= -\frac{1}{2} (\Sigma^{-1})^T + \frac{\nu+m}{2} \frac{\Sigma^{-T} \zeta \zeta^T \Sigma^{-T}}{\nu + \zeta^T \Sigma^{-1} \zeta}
\end{aligned}$$

Archimedean Copulas

Overview of Archimedean Copulas

Let function $\phi : [0, 1] \rightarrow [0, \infty)$ be a strict Archimedean copula generator function, and suppose that its inverse ϕ^{-1} is completely monotonic on $[0, \infty)$. A strict generator is a decreasing function $\phi : [0, 1] \rightarrow [0, \infty)$ that satisfies $\phi(0) = \infty$ and $\phi(1) = 0$. A decreasing function $f(t) : [a, b] \rightarrow (-\infty, \infty)$ is completely monotonic if it satisfies

$$(-1)^k \frac{d^k}{dt^k} f(t) \geq 0, k \in \mathbb{N}, t \in (a, b)$$

An Archimedean copula is defined as follows:

$$C(u_1, u_2, \dots, u_m) = \phi^{-1}\left(\phi(u_1) + \dots + \phi(u_m)\right)$$

The Archimedean copulas available in the CCOPULA procedure are the Clayton copula, the Frank copula, and the Gumbel copula.

Clayton Copula

Let the generator function $\phi(u) = \theta^{-1} (u^{-\theta} - 1)$. A Clayton copula is defined as

$$C_\theta(u_1, u_2, \dots, u_m) = \left[\sum_{i=1}^m u_i^{-\theta} - m + 1 \right]^{-1/\theta}$$

where $\theta > 0$.

Frank Copula

Let the generator function be

$$\phi(u) = -\log \left[\frac{\exp(-\theta u) - 1}{\exp(-\theta) - 1} \right]$$

A Frank copula is defined as

$$C_\theta(u_1, u_2, \dots, u_m) = \frac{1}{\theta} \log \left\{ 1 + \frac{\prod_{i=1}^m [\exp(-\theta u_i) - 1]}{[\exp(-\theta) - 1]^{m-1}} \right\}$$

where $\theta \in (-\infty, \infty) \setminus \{0\}$ for $m = 2$ and $\theta > 0$ for $m \geq 3$.

Gumbel Copula

Let the generator function $\phi(u) = (-\log u)^\theta$. A Gumbel copula is defined as

$$C_\theta(u_1, u_2, \dots, u_m) = \exp \left\{ - \left[\sum_{i=1}^m (-\log u_i)^\theta \right]^{1/\theta} \right\}$$

where $\theta > 1$.

Simulation

Suppose that the generator of the Archimedean copula is ϕ . Then the simulation method that uses a Laplace-Stieltjes transformation of the distribution function is given by Marshall and Olkin (1988), where $\tilde{F}(t) = \int_0^\infty e^{-tx} dF(x)$:

1. Generate a random variable V that has the distribution function F such that $\tilde{F}(t) = \phi^{-1}(t)$.
2. Draw samples from the independent uniform random variables $X_1, \dots, X_m$.
3. Return $U = (\tilde{F}(-\log(X_1)/V), \dots, \tilde{F}(-\log(X_m)/V))^T$.

The Laplace-Stieltjes transformations are as follows:

- For the Clayton copula, $\tilde{F} = (1 + t)^{-1/\theta}$, and the distribution function F is associated with a gamma random variable that has a shape parameter of θ^{-1} and a scale parameter of 1.

- For the Gumbel copula, $\tilde{F} = \exp(-t^{1/\theta})$, and F is the distribution function of the stable variable $\text{St}(\theta^{-1}, 1, \gamma, 0)$, where $\gamma = [\cos(\pi/(2\theta))]^\theta$.
- For the Frank copula where $\theta > 0$, $\tilde{F} = -\log\{1 - \exp(-t)[1 - \exp(-\theta)]\}/\theta$, and F is a discrete probability function $P(V = k) = (1 - \exp(-\theta))^k/(k\theta)$. This probability function is related to a logarithmic random variable that has a parameter value of $1 - e^{-\theta}$.

For more information about simulating a random variable from a stable distribution, see Theorem 1.19 in Nolan (2010). For more information about simulating a random variable from a logarithmic series, see Chapter 10.5 in Devroye (1986).

For a Frank copula where $m = 2$ and $\theta < 0$, the simulation can be done through conditional distributions as follows:

1 Draw independent v_1, v_2 from a uniform distribution.

2 Let $u_1 = v_1$.

3 Let $u_2 = -\frac{1}{\theta} \log \left(1 + \frac{v_2(1-e^{-\theta})}{v_2(e^{-\theta v_1}-1)-e^{-\theta v_1}} \right)$.

Fitting

One method to estimate the parameters is to calibrate with Kendall's tau. The relation between the parameter θ and Kendall's tau is summarized in Table 8.2 for the three Archimedean copulas.

Table 8.2 Calibration Using Kendall's Tau

Copula Type	τ	Formula for θ
Clayton	$\theta/(\theta + 2)$	$2\tau/(1 - \tau)$
Gumbel	$1 - 1/\theta$	$1/(1 - \tau)$
Frank	$1 - 4\theta^{-1}(1 - D_1(\theta))$	No closed form

In Table 8.2, $D_1(\theta) = \theta^{-1} \int_0^\theta t/(\exp(t) - 1)dt$ for $\theta > 0$, and $D_1(\theta) = D_1(\theta) + 0.5\theta$ for $\theta < 0$. In addition, for the Frank copula, the formula for θ has no closed form. The numerical algorithm for root finding can be used to invert the function $\tau(\theta)$ to obtain θ as a function of τ .

Alternatively, you can use the MLE or CMLE method to estimate the parameter θ given the data $\mathbf{u} = \{u_{i,j}\}$ and $i = 1, \dots, n, j = 1, \dots, m$. The log-likelihood function for each type of Archimedean copula is provided in the following sections.

Fitting the Clayton Copula

For the Clayton copula, the log-likelihood function is as follows (Cherubini, Luciano, and Vecchiato 2004, Chapter 7):

$$l = n \left[m \log(\theta) + \log \left(\Gamma \left(\frac{1}{\theta} + m \right) \right) - \log \left(\Gamma \left(\frac{1}{\theta} \right) \right) \right] - (\theta + 1) \sum_{i,j} \log u_{ij} \\ - \left(\frac{1}{\theta} + m \right) \sum_i \log \left(\sum_j u_{ij}^{-\theta} - m + 1 \right)$$

Let $g(\cdot)$ be the derivative of $\log(\Gamma(\cdot))$. Then the first-order derivative is

$$\frac{dl}{d\theta} = n \left[\frac{m}{\theta} + g \left(\frac{1}{\theta} + m \right) \frac{-1}{\theta^2} - g \left(\frac{1}{\theta} \right) \frac{-1}{\theta^2} \right] \\ - \sum_{i,j} \log(u_{ij}) + \frac{1}{\theta^2} \sum_i \log \left(\sum_j u_{ij}^{-\theta} - m + 1 \right) \\ - \left(\frac{1}{\theta} + m \right) \sum_i \frac{-\sum_j u_{ij}^{-\theta} \log(u_{ij})}{\sum_j u_{ij}^{-\theta} - m + 1}$$

The second-order derivative is

$$\frac{d^2l}{d\theta^2} = n \left\{ \frac{-m}{\theta^2} + g' \left(\frac{1}{\theta} + m \right) \frac{1}{\theta^4} + g \left(\frac{1}{\theta} + m \right) \frac{2}{\theta^3} - g' \left(\frac{1}{\theta} \right) \frac{1}{\theta^4} - g \left(\frac{1}{\theta} \right) \frac{2}{\theta^3} \right\} \\ - \frac{2}{\theta^3} \sum_i \log \left(\sum_j u_{ij}^{-\theta} - m + 1 \right) \\ + \frac{2}{\theta^2} \sum_i \frac{-\sum_j u_{ij}^{-\theta} \log u_{ij}}{\sum_j u_{ij}^{-\theta} - m + 1} \\ - \left(\frac{1}{\theta} + m \right) \sum_i \left\{ \frac{\sum_j u_{ij}^{-\theta} (\log u_{ij})^2}{\sum_j u_{ij}^{-\theta} - m + 1} - \left(\frac{\sum_j u_{ij}^{-\theta} \log u_{ij}}{\sum_j u_{ij}^{-\theta} - m + 1} \right)^2 \right\}$$

Fitting the Gumbel Copula

A different parameterization $\alpha = \theta^{-1}$ is used for the following part, which is related to the fitting of the Gumbel copula. For the Gumbel copula, you need to compute $\phi^{-1(m)}$. It turns out that for $k = 1, 2, \dots, m$,

$$\phi^{-1(k)}(u) = (-1)^k \alpha \exp(-u^\alpha) u^{-k+\alpha} \Psi_{k-1}(u^\alpha)$$

where Ψ_{k-1} is a function that is described later. The copula density is given by

$$c = \phi^{-1(m)}(x) \prod_k \phi'(u_k) \\ = (-1)^m \alpha \exp(-x^\alpha) x^{-k+\alpha} \Psi_{m-1}(x^\alpha) \prod_k \phi'(u_k) \\ = (-1)^m f_1 f_2 f_3 f_4 f_5$$

where $x = \sum_k \phi(u_k)$, $f_1 = \alpha$, $f_2 = \exp(-x^\alpha)$, $f_3 = x^{-k+\alpha}$, $f_4 = \Psi_{m-1}(x^\alpha)$, and $f_5 = (-1)^m \prod_k \phi'(u_k)$.

The log density is

$$\begin{aligned} l &= \log(c) \\ &= \log(f_1) + \log(f_2) + \log(f_3) + \log(f_4) + \log((-1)^m f_5) \end{aligned}$$

Now the first-order derivative of the log density has the decomposition

$$\frac{dl}{d\alpha} = \frac{1}{c} \frac{dc}{d\alpha} = \sum_{j=1}^4 \frac{1}{f_j} \frac{df_j}{d\alpha} + \frac{d \sum_k \log(-\phi'(u_k))}{d\alpha}$$

Some of the terms are given by

$$\begin{aligned} \frac{1}{f_1} \frac{df_1}{d\alpha} &= \frac{1}{\alpha} \\ \frac{1}{f_2} \frac{df_2}{d\alpha} &= -x^\alpha \log(x) - \alpha x^{\alpha-1} \frac{dx}{d\alpha} \\ \frac{1}{f_3} \frac{df_3}{d\alpha} &= \log(x) + (-k + \alpha) x^{-1} \frac{dx}{d\alpha} \end{aligned}$$

where

$$\frac{dx}{d\alpha} = \sum (-\log u_k)^{1/\alpha} \log(-\log u_k) \left(\frac{-1}{\alpha^2} \right)$$

The last term in the derivative of the $dl/d\alpha$ is

$$\begin{aligned} \log(-\phi'(u_k)) &= \log\left(\frac{1}{\alpha} (-\log u_k)^{\frac{1}{\alpha}-1} \frac{1}{u_k}\right) \\ &= -\log \alpha - \log(u_k) + \left(\frac{1}{\alpha} - 1\right) \log(-\log(u_k)) \\ \frac{d \sum_k \log(-\phi'(u_k))}{d\alpha} &= \sum_{k=1}^m -\frac{1}{\alpha} - \frac{1}{\alpha^2} \log(-\log(u_k)) \\ &= -\frac{m}{\alpha} - \frac{1}{\alpha^2} \sum_{k=1}^m \log(-\log(u_k)) \end{aligned}$$

Now the only remaining term is f_4 , which is related to Ψ_{m-1} . Wu, Valdez, and Sherris (2007) show that $\Psi_k(x)$ satisfies a recursive equation

$$\Psi_k(x) = [\alpha(x-1) + k] \Psi_{k-1}(x) - \alpha x \Psi'_{k-1}(x)$$

with $\Psi_0(x) = 1$.

The preceding equation implies that $\Psi_{k-1}(x)$ is a polynomial of x and therefore can be represented as

$$\Psi_{k-1}(x) = \sum_{j=0}^{k-1} a_j(k-1, \alpha) x^j$$

In addition, its coefficient, denoted by $a_j(k-1, \alpha)$, is a polynomial of α . For simplicity, use the notation $a_j(\alpha) \equiv a_j(m-1, \alpha)$. Therefore,

$$f_4 = \Psi_{m-1}(x^\alpha) = \sum_{j=0}^{m-1} a_j(\alpha) x^{j\alpha}$$

$$\begin{aligned} \frac{df_4}{d\alpha} &= \frac{d\Psi_{m-1}(x^\alpha)}{d\alpha} \\ &= \sum_{j=0}^{m-1} \left[\frac{da_j(\alpha)}{d\alpha} x^{j\alpha} + a_j(\alpha) x^{j\alpha} \log(x) j + a_j(\alpha) (j\alpha) x^{j\alpha-1} \frac{dx}{d\alpha} \right] \end{aligned}$$

Fitting the Frank Copula

For the Frank copula,

$$\phi^{-1(k)}(u) = -\frac{1}{\theta} \Psi_{k-1} \left((1 + e^{-u}(e^{-\theta} - 1))^{-1} \right)$$

When $\theta > 0$, a Frank copula has a probability density function

$$\begin{aligned} c &= \varphi^{-1(m)}(x) \prod_k \varphi'(u_k) \\ &= \frac{-1}{\theta} \Psi_{m-1} \left(\frac{1}{1 + e^{-x}(e^{-\theta} - 1)} \right) \prod_k \varphi'(u_k) \end{aligned}$$

where $x = \sum_k \varphi(u_k)$.

The log likelihood is

$$\log c = -\log(\theta) + \log \left(\Psi_{m-1} \left(\frac{1}{1 + e^{-x}(e^{-\theta} - 1)} \right) \right) + \sum \log(\varphi'(u_k))$$

Denote

$$y = \frac{1}{1 + e^{-x}(e^{-\theta} - 1)}$$

Then the derivative of the log likelihood is

$$\frac{d \log c}{d\theta} = -\frac{1}{\theta} + \frac{1}{\Psi_{m-1}(y)} \frac{d\Psi_{m-1}}{d\theta} + \sum_k \frac{1}{\varphi'(u_k)} \frac{d\varphi'(u_k)}{d\theta}$$

The term in the last summation is

$$\frac{1}{\varphi'(u_k)} \frac{d\varphi'(u_k)}{d\theta} = \frac{1}{\theta(1 - e^{\theta u_k})} [1 - e^{\theta u_k} + \theta u_k e^{\theta u_k}]$$

The function Ψ_{m-1} satisfies a recursive relation

$$\Psi_k(x) = x(x-1)\Psi'_{k-1}(x)$$

with $\Psi_0(x) = x - 1$. Note that Ψ_{m-1} is a polynomial whose coefficients do not depend on θ ; therefore,

$$\begin{aligned} \frac{d\Psi_{m-1}}{d\theta} &= \frac{d\Psi_{m-1}}{dy} \frac{dy}{d\theta} \\ &= \frac{d\Psi_{m-1}}{dy} \left[\frac{dy}{d\theta} + \frac{dy}{dx} \frac{dx}{d\theta} \right] \\ &= \frac{d\Psi_{m-1}}{dy} \left[\frac{e^{-x}e^{-\theta}}{[1 + e^{-x}(e^{-\theta} - 1)]^2} + \frac{e^{-x}(e^{-\theta} - 1)}{[1 + e^{-x}(e^{-\theta} - 1)]^2} \frac{dx}{d\theta} \right] \end{aligned}$$

where

$$\begin{aligned} \frac{dx}{d\theta} &= \sum_k \frac{d\varphi(u_k)}{d\theta} = \sum_k \left[-\frac{u_k e^{-\theta u_k}}{1 - e^{-\theta u_k}} + \frac{e^{-\theta}}{1 - e^{-\theta}} \right] \\ &= \sum_k \left[-\frac{u_k}{e^{\theta u_k} - 1} + \frac{1}{e^{\theta} - 1} \right] \end{aligned}$$

For the case of $m = 2$ and $\theta < 0$, the bivariate density is

$$\log c = \log(\theta(1 - e^{-\theta})) - \theta(u_1 + u_2) - \log((1 - e^{-\theta} - (1 - e^{-\theta u_1})(1 - e^{-\theta u_2}))^2)$$

Canonical Maximum Likelihood Estimation (CMLE)

In the canonical maximum likelihood estimation (CMLE) method, it is assumed that the sample data $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{im})^\top$, where $i = 1, \dots, n$, have been transformed into uniform variates $\hat{\mathbf{u}}_i = (\hat{u}_{i1}, \dots, \hat{u}_{im})$, $i = 1, \dots, n$. One commonly used transformation is the nonparametric estimation of the CDF of the marginal distributions, which is closely related to empirical CDF,

$$\hat{u}_{i,j} = \hat{F}_{j,n}(x_{i,j})$$

where

$$\hat{F}_{j,n}(x) = \frac{1}{n+1} \sum_{i=1}^n I_{[x_{i,j} \leq x]}$$

The transformed data $\hat{u}_{i,j}$ are used as if they had uniform marginal distributions; hence, they are called pseudo-samples. The function $\hat{F}_{j,n}$ is different from the standard empirical CDF in the scalar $1/(n+1)$,

which is to ensure that the transformed data cannot be on the boundary of the unit interval $[0, 1]$. It is clear that

$$\hat{u}_{i,j} = \frac{1}{n+1} \text{rank}(x_{i,j})$$

where $\text{rank}(x_{i,j})$ is the rank among $i = 1, \dots, n$ in increasing order.

Let $c(u_1, u_2, \dots, u_m; \theta)$ be the density function of a copula $C(u_1, u_2, \dots, u_m; \theta)$, and let θ be the parameter vector to be estimated. The parameter θ is estimated by maximum likelihood:

$$\hat{\theta} = \arg \max_{\theta \in \Theta} \sum_{i=1}^n \log c(\hat{u}_{i1}, \dots, \hat{u}_{im}; \theta)$$

Exact Maximum Likelihood Estimation (MLE)

Suppose that the marginal distributions of vector elements $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{im})^\top$, where $i = 1, \dots, n$, are already known to be uniform. Then the parameter θ is estimated by exact maximum likelihood:

$$\hat{\theta} = \arg \max_{\theta \in \Theta} \sum_{i=1}^n \log c(x_{i1}, x_{i2}, \dots, x_{im}; \theta)$$

Calibration Estimation

Instead of fitting the whole distribution as in MLE methods, you can directly use empirical estimates of distribution parameters. The unknown parameter that you want to estimate can be obtained by calibration that uses Kendall's tau. There is a one-to-one map between the parameter of interest and Kendall's tau. Therefore, after you estimate Kendall's tau, you can use the map to compute the parameter value. For example, the parameter matrix Σ in a t copula and the parameter θ in Archimedean copulas can be estimated in this manner. The most frequently used estimator of Kendall's tau is the rank correlation coefficient:

$$\hat{\rho}_\tau(X_i, X_j) = \left(\frac{n}{2} \right)^{-1} \sum_{1 \leq t < s \leq n} \text{sign}((x_{t,i} - x_{s,i})(x_{t,j} - x_{s,j}))$$

The preceding formula is analogous to its population counterpart,

$$\rho_\tau(X_i, X_j) = E[\text{sign}((X_i - \tilde{X}_i)(X_j - \tilde{X}_j))]$$

where $(\tilde{X}_i, \tilde{X}_j)$ has the same distribution but is independent of (X_i, X_j) .

For Archimedean multivariate copulas, there is only one parameter to estimate, τ (or its function θ), although for m variables there are $m(m-1)/2$ unique pairwise correlation coefficients. Denote the map from ρ_τ to θ by $\theta = \hat{\theta}(\rho_\tau)$. To aggregate the map, take a simple arithmetic average:

$$\hat{\theta} = \frac{2}{m(m-1)} \sum_{1 \leq i < j \leq m} \hat{\theta}[\hat{\rho}_\tau(X_i, X_j)]$$

Nonlinear Optimization Options

PROC CCOPULA uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. In the PROC CCOPULA statement, you can specify nonlinear optimization options that are then passed to the NLO subsystem. For a list of all the nonlinear optimization options, see *SAS/ETS User's Guide*.

Displayed Output

PROC CCOPULA produces the displayed output that is described in the following sections.

Optimization Start and Resulting Parameter Estimates

If you specify the ITPRINT option in the PROC CCOPULA statement, PROC CCOPULA displays two tables, “Optimization Start Parameter Estimates” and “Optimization Results Parameter Estimates.” Each table contains the following information for each model parameter:

- parameter number
- parameter name
- parameter estimate
- gradient of the objective function at the initial parameter values

In addition to this information, the “Optimization Start Parameter Estimates” table contains the following columns:

- lower-bound constraint
- upper-bound constraint

The value of the objective function at the parameter values is displayed below each table.

Iteration History for Parameter Estimates

If you specify the ITPRINT option in the PROC CCOPULA statement, PROC CCOPULA displays a table that contains the following information for each iteration. Note that some information is specific to the model-fitting method that you choose (for example, Newton-Raphson, trust region, or quasi-Newton method).

- iteration number
- number of restarts since the fitting began
- number of function calls
- number of active constraints at the current solution

- value of the objective function (-1 times the log-likelihood value) at the current solution
- change in the objective function from the previous iteration
- value of the maximum absolute gradient element
- step size (for Newton-Raphson and quasi-Newton methods)
- slope of the current search direction (for Newton-Raphson and quasi-Newton methods)
- lambda (for trust region method)
- radius value at the current iteration (for trust region method)

Model Fit Summary

The “Model Fit Summary” table contains the following information:

- number of observations used
- number of missing values in the data table, if any
- data table name
- type of model that was fit
- log-likelihood value at the solution
- maximum absolute gradient at the solution
- number of iterations
- optimization method
- value of Akaike’s information criterion (AIC) at the solution (a smaller value indicates better fit)
- value of Schwarz Bayesian criterion (SBC) at the solution (a smaller value indicates better fit)

Below the “Model Fit Summary” table is a statement about whether the algorithm successfully converged.

Parameter Estimates

The “Parameter Estimates” table contains the estimates of the model parameters. For the normal copula, this table is not displayed because the only parameters are in the correlations matrix, which is displayed in the “Pearson Correlations Matrix” table. For the t copula, the parameter is the number of degrees of freedom; in the table it is called “DF.” For Archimedean copulas such as Clayton, Frank, and Gumbel, the parameter is called “theta.”

Pearson Correlations Matrix

The “Pearson Correlations Matrix” table contains the estimates of the model Pearson correlations matrix. This table is displayed only for the normal and t copulas. Row and column names come from the list of variables that are defined in the VAR statement.

Kendall Correlations Matrix

The “Kendall Correlations Matrix” table contains the estimates of the model Kendall correlations matrix. This table is displayed only for the normal and t copulas. Row and column names come from the list of variables that are defined in the VAR statement.

Spearman Correlations Matrix

The “Spearman Correlations Matrix” table contains the estimates of the model Spearman correlations matrix. This table is displayed only for the normal copula. Row and column names come from the list of variables that are defined in the VAR statement.

OUTPSEUDO= and OUTUNIFORM= Data Tables

The number of columns and the names of columns in the OUTPSEUDO= and OUTUNIFORM= data tables match the number and names of the variables in the VAR statement.

ODS Table Names

PROC CCOPULA assigns a name to each table that it creates. You can use these names to denote the table when you use the Output Delivery System (ODS) to select tables and create output data tables. These table names are listed in [Table 8.3](#).

Table 8.3 ODS Tables Produced by PROC CCOPULA

Table Name	Description	Option
ODS Tables Created by the FIT Statement		
ConvergenceStatus	Convergence status	Default
PearsonCorrelations	Pearson correlations matrix estimates	Default with elliptical copulas
KendallCorrelations	Kendall correlations matrix estimates	Default with elliptical copulas
SpearmanCorrelations	Spearman correlations matrix estimates	Default with normal copula
ParameterEstimates	Parameter estimates	Default
ODS Tables Created by the PRINTTIMING Option		
TimingDetails	Detailed summary of time used for all phases of execution	PRINTTIMING=(DETAILS)

Table 8.3 *continued*

Table Name	Description	Option
TimingSummary	Summary of time used for main phases of execution	PRINTTIMING
ODS Table Created by the SIMULATE Statement		
SimulateModelSummary	Summary of simulation	Default

ODS Graph Names

PROC CCOPULA assigns a name to each graph that it creates by using ODS. You can use these names to selectively reference the graphs. The names are listed in [Table 8.4](#).

Table 8.4 ODS Graphics Produced by PROC CCOPULA

ODS Graph Name	Plot Description	PLOTS= Option
Fit Plots		
MatrixPlotOrig	Matrix of correlation plots	PLOTS=(DATATYPE=ORIGINAL)
MatrixPlotUnif	Matrix of transformed correlation plots	PLOTS=(DATATYPE=UNIFORM)
ScatterPlotOrig	Individual correlation plots	PLOTS(UNPACK)=(DATATYPE=ORIGINAL)
ScatterPlotUnif	Individual transformed correlation plots	PLOTS(UNPACK)=(DATATYPE=UNIFORM)
ChiPlotOrig	Matrix of correlation and tail plots	PLOTS(TAIL)=(DATATYPE=ORIGINAL)
ChiPlotUnif	Matrix of transformed correlation and tail plots	PLOTS(TAIL)=(DATATYPE=UNIFORM)
ChiPlot	Individual tail plots	PLOTS(TAIL)
KendallPlotOrig	Matrix of correlation and Kendall plots	PLOTS(KENDALL)=(DATATYPE=ORIGINAL)
KendallPlotUnif	Matrix of transformed correlation and Kendall plots	PLOTS(KENDALL)=(DATATYPE=UNIFORM)
KendallPlot	Individual Kendall plots	PLOTS(KENDALL UNPACK)
Simulation Plots		
MatrixPlotSUnif	Matrix of uniform correlation plots	PLOTS=(DATATYPE=UNIFORM)
ScatterPlotSUnif	Individual uniform correlation plots	PLOTS(UNPACK)=(DATATYPE=UNIFORM)
ChiPlotSUnif	Matrix of uniform correlation and tail plots	PLOTS(TAIL)
ChiPlotS	Individual tail plots	PLOTS(TAIL UNPACK)

Table 8.4 *continued*

ODS Graph Name	Plot Description	PLOTS= Option
KendallPlotSUnif	Matrix of uniform correlation and Kendall plots	PLOTS(KENDALL)
KendallPlotS	Individual Kendall plots	PLOTS(KENDALL UNPACK)

Examples: CCOPULA Procedure

Example 8.1: Simulating Default Times

Suppose that the correlations structure that is required for a normal copula function is already given. For example, it can be estimated from the historic data on default times in some set of industries, but this stage is not within the scope of this example. The correlations structure is saved in a SAS data set called `inparm`. The following statements and their output in [Output 8.1.1](#) show that the correlation parameter for Y1 and Y2 is set at 0.8:

```
proc print data = inparm;
run;
```

Output 8.1.1 Pearson Correlations Matrix

Obs	Y1	Y2
1	1.0	0.8
2	0.8	1.0

Then you use PROC CCOPULA as follows to simulate the data. The VAR statement specifies the list of variables to contain simulated data. The DEFINE statement assigns the name COP and specifies a normal copula that reads the correlations matrix from the `inparm` data table.

The SIMULATE statement refers to the COP label that is defined in the VAR statement and specifies some options: the NDRAWS= option specifies a sample size, the SEED= option specifies 1234 as the random number generator seed, and the OUTUNIFORM=NORMAL_UNIFDATA option names the output data table for the result of simulation in uniforms.

```
/* simulate the data from bivariate normal copula */
proc ccopula ;
  var Y1-Y2;
  define cop normal (corr=mylib.inparm);
  simulate cop /
    ndraws      = 1000000
    seed        = 1234
    outuniform  = mylib.normal_unifdata;
run;
```

The following DATA step transforms the variables from zero-one uniformly distributed to nonnegative exponentially distributed with parameter 0.5. Three indicator variables are added to the data table. SURVIVE1 and SURVIVE2 have a value of 1 if a respective company has remained in business for more than three years. DEFAULT has a value of 1 if neither company survived for more than three years.

```
/* default time has exponential marginal distribution with parameter 0.5 */
data default;
  set mylib.normal_unifdata;
  array arr{2} Y1-Y2;
  array time{2} time1-time2;
  array surv{2} survive1-survive2;
  lambda = 0.5;
  do i=1 to 2;
    time[i] = -log(1-arr[i])/lambda;
    surv[i] = 0;
    if (time[i] >3) then surv[i]=1;
  end;
  default = 0;
  if (time1 <=3) && (time2 <=3) then default = 1;
run;
```

The first analysis step is to look at correlations between survival times of two companies. This step is performed by using the following CORR procedure statements:

```
proc corr data = default pearson kendall;
  var time1 time2;
run;
```

The output is shown in [Output 8.1.2](#).

Output 8.1.2 Default Time Descriptive Statistics and Correlations

The CORR Procedure

2 Variables: time1 time2

Simple Statistics						
Variable	N	Mean	Std Dev	Median	Minimum	Maximum
time1	1000000	2.00023	1.99883	1.38633	3.85293E-6	24.18938
time2	1000000	1.99897	1.99965	1.38476	6.43006E-6	25.85567

Pearson Correlation Coefficients, N = 1000000
Prob > |r| under H0: Rho=0

	time1	time2
time1	1.00000	0.77046 <.0001
time2	0.77046 <.0001	1.00000

Output 8.1.2 *continued*

Kendall Tau b Correlation Coefficients, N = 1000000 Prob > tau under H0: Tau=0		
	time1	time2
time1	1.00000	0.59046 <.0001
time2	0.59046 <.0001	1.00000

Output 8.1.2 shows some descriptive statistics and two measures of correlation: Pearson and Kendall. Both of these measures indicate high and statistically significant dependence between life spans of two companies.

The second and final step is to empirically estimate the default probabilities of two companies. This is done using the FREQ procedure as follows:

```
proc freq data=default;
    table default survive1-survive2;
run;
```

The results are shown in Output 8.1.3.

Output 8.1.3 Probabilities of Default**The FREQ Procedure**

default	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	298431	29.84	298431	29.84
1	701569	70.16	1000000	100.00

survive1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	776594	77.66	776594	77.66
1	223406	22.34	1000000	100.00

survive2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	777292	77.73	777292	77.73
1	222708	22.27	1000000	100.00

Output 8.1.3 shows that the empirical default probabilities are 77.66% and 77.73%. If you assume that these companies are independent, the probability estimate that both companies will default during the period of three years is $0.7766 \times 0.7773 = 0.6037$ (60.37%). If you compare this naive estimate to the higher actual joint default probability of 70.16%, you see that if you neglected the correlation between the two companies, you underestimate the probability of default.

Example 8.2: Copula Analysis of Stock Returns

In this example, the correlations of daily returns on the large capital stocks Microsoft, Coca-Cola, IBM, Duke Energy, and British Petroleum are modeled using a t copula. To model the correlations, these companies' stock prices are first converted into returns by using the following DATA step:

```
data allreturns;
    format date MMDDYY10.;
    input date:MMDDYY10. price_ibm price_msft price_bp price_ko price_duk;
    ret_msft = price_msft/lag(price_msft)-1;
    ret_ko   = price_ko/lag(price_ko)-1;
    ret_ibm  = price_ibm/lag(price_ibm)-1;
    ret_duk  = price_duk/lag(price_duk)-1;
    ret_bp   = price_bp/lag(price_bp)-1;
    if ret_msft ^= .;
    keep date ret_ibm ret_msft ret_bp ret_ko ret_duk;
datalines;
01/02/2008    99.91    33.48    62.78    56.91    17.13
01/03/2008   100.11    33.62    63.98    57.50    17.19
01/04/2008    96.52    32.68    63.05    57.61    17.18
01/07/2008    95.49    32.90    64.18    58.97    17.62

... more lines ...
```

The stock return data can then be visualized and used to estimate the parameters of a t copula by using the following code:

```
/* Copula estimation and simulation of returns */
proc ccopula data = mylib.allreturns;
    var ret_ibm ret_msft ret_bp ret_ko ret_duk;
    * fit t-copula to stock returns;
    fit T / marginals = empirical
           method     = MLE
           plots = (datatype = both);
run;

proc ccopula;
    var x1-x5;
    define t t(df=5);
    simulate t / ndraws=603 seed=100 outuniform=mylib.copuniform plots;
run;

proc ccopula data=mylib.copuniform;
    var x1-x5;
    fit T / marginals = uniform
           method     = MLE
           plots = (datatype = uniform);
run;
```

The output from these statements is shown in [Output 8.2.1](#).

Output 8.2.1 Copula Estimation**The CCOPULA Procedure**

Model Fit Summary				
Number of Observations	603			
Data Set	ALLRETURNS			
Copula Type	T			
Fit Method	MLE			
Marginals	Empirical			
Empirical Approximation	Adaptive Binning			
Adaptive Refinement Resolution	1000			
Adaptive Interpolation Method	Linear			
Number of Adaptive Iterations	4			
Log Likelihood	634.6931			
Maximum Absolute Gradient	1.596E-7			
Number of Iterations	7			
Optimization Method	Newton-Raphson			
AIC	-1247.39			
SBC	-1198.97			

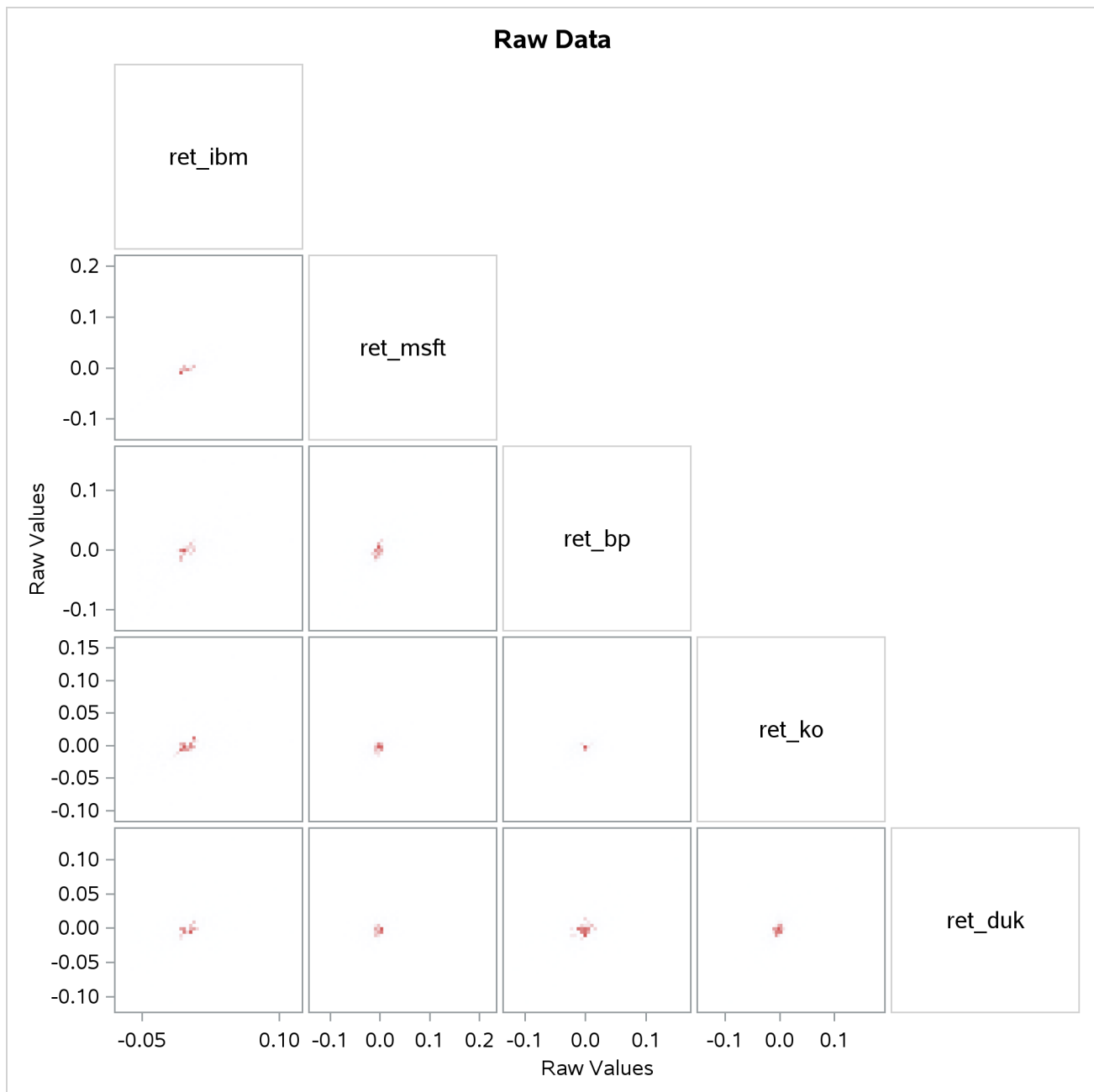
Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Approx Pr > t
DF	3.659299	0.320474	11.42	<.0001

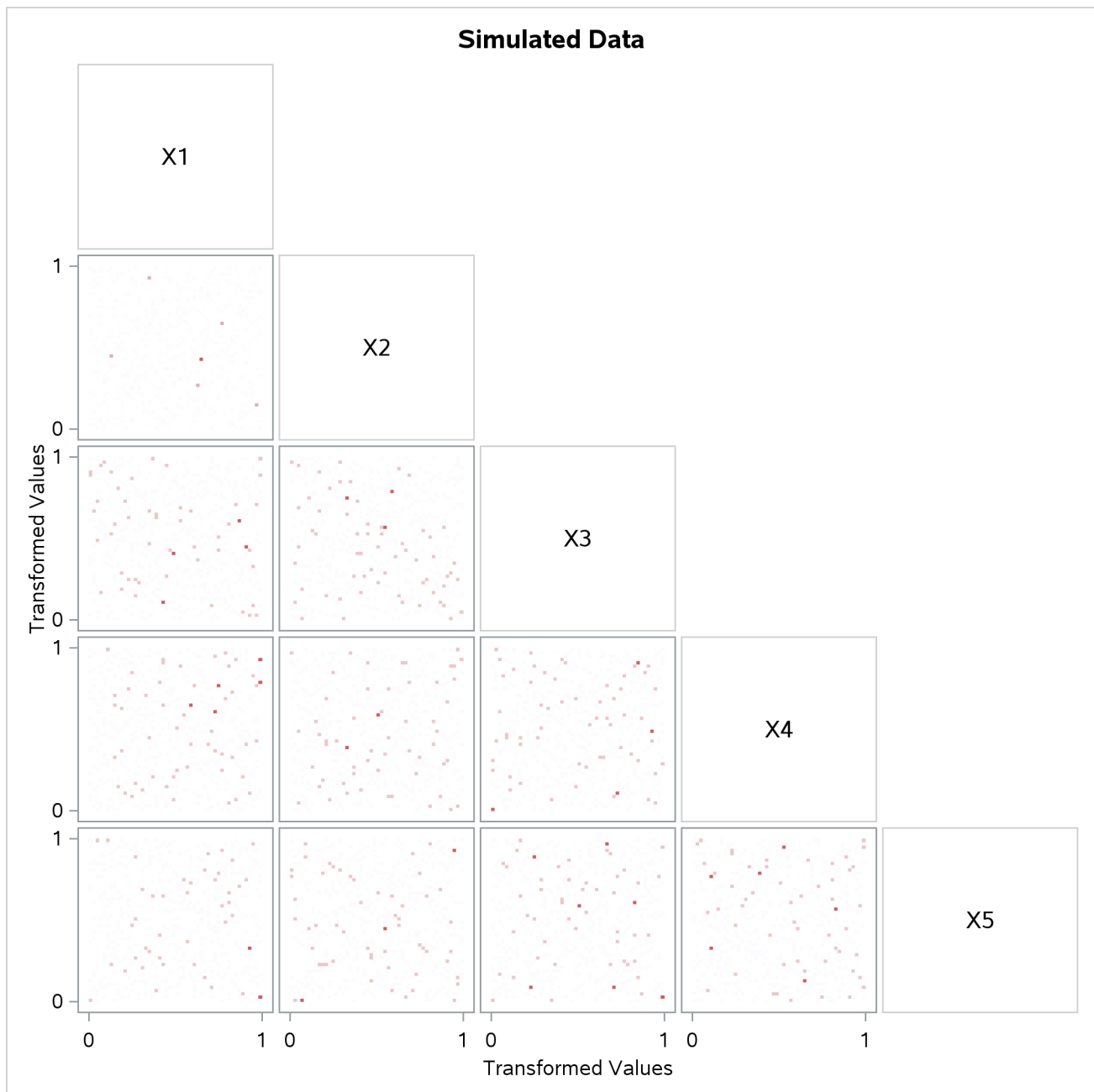
Pearson Correlations Matrix					
Variable	ret_ibm	ret_msft	ret_bp	ret_ko	ret_duk
ret_ibm	1.0000	0.6415	0.5195	0.5004	0.5023
ret_msft	0.6415	1.0000	0.5137	0.4935	0.4388
ret_bp	0.5195	0.5137	1.0000	0.4063	0.4278
ret_ko	0.5004	0.4935	0.4063	1.0000	0.5266
ret_duk	0.5023	0.4388	0.4278	0.5266	1.0000

The first table in [Output 8.2.1](#), “Model Fit Summary,” provides some general description of copula model estimation. The second table, “Parameter Estimates,” provides point estimates and inference about copula parameters. In this example, the only parameter in this table is the number of degrees of freedom in the multivariate t distribution. The last table, “Correlation Matrix,” contains estimates of copula model parameters.

The graphical output from the preceding statements is shown in [Output 8.2.2](#) and [Output 8.2.3](#).

Output 8.2.2 Original Data



Output 8.2.3 Original Data Transformed into Uniform Marginals

Note that in [Output 8.2.2](#), the most elliptical scatter plot, which is between IBM and Microsoft, indicates the strongest dependence. Similarly, in [Output 8.2.3](#), graphs that are denser along the diagonal indicate the same thing.

Example 8.3: Copula Analysis of S&P 500 Sectors

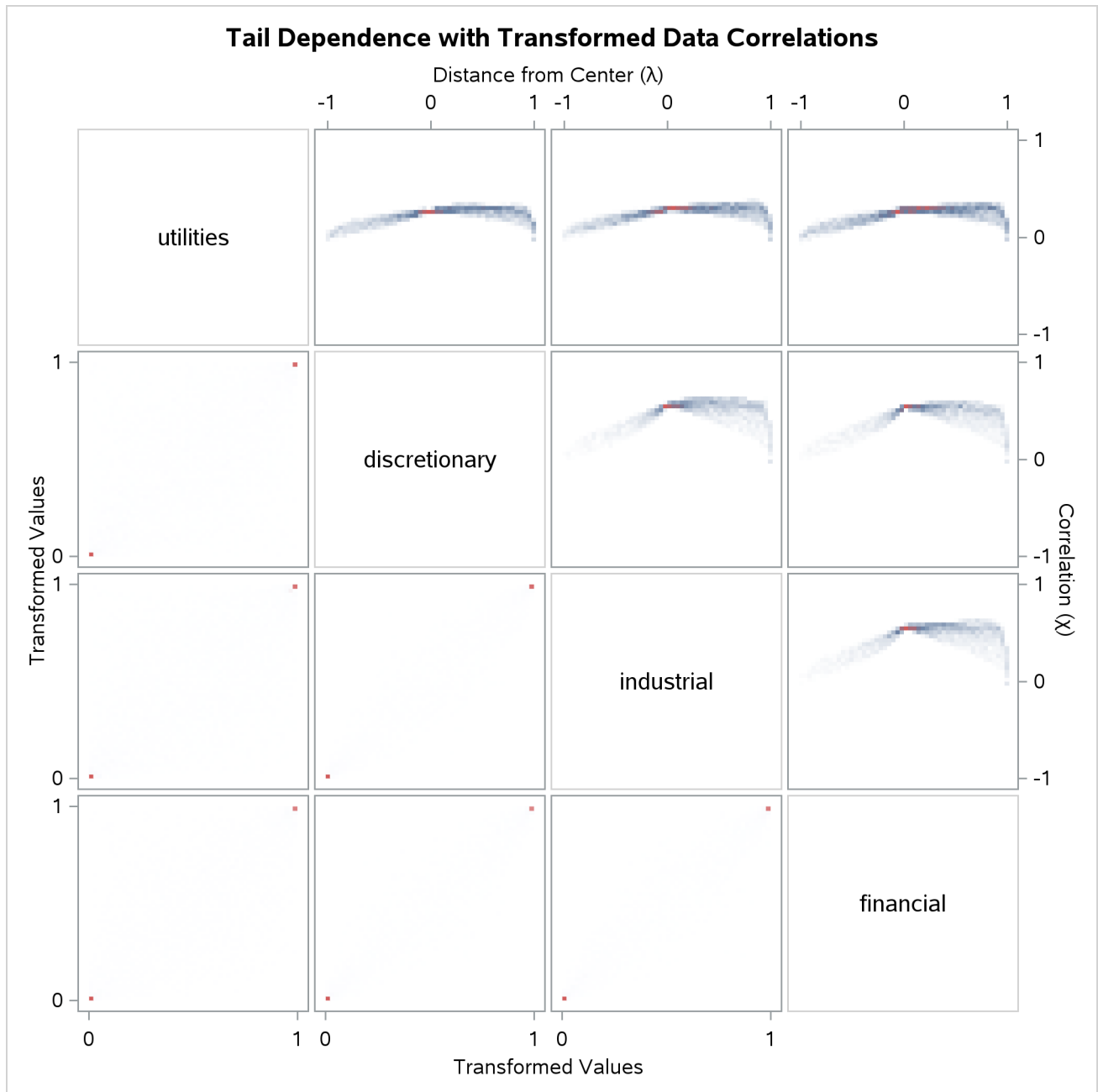
This example analyzes the dependencies among the market sectors in the S&P 500 index. The S&P 500 index consists of 11 sectors: Information Technology, Health Care, Financials, Consumer Discretionary, Communication Services, Industrials, Consumer Staples, Energy, Utilities, Real Estate, and Materials. This analysis considers daily returns for these sectors that have been corrected for stock splits and dividend payments over the years 1999 to 2020.

To facilitate visualization of the return data, this analysis is limited to the Utilities, Consumer Discretionary, Industrials, and Financials sectors. The following code fits a t copula and performs a simulation of the estimated t copula:

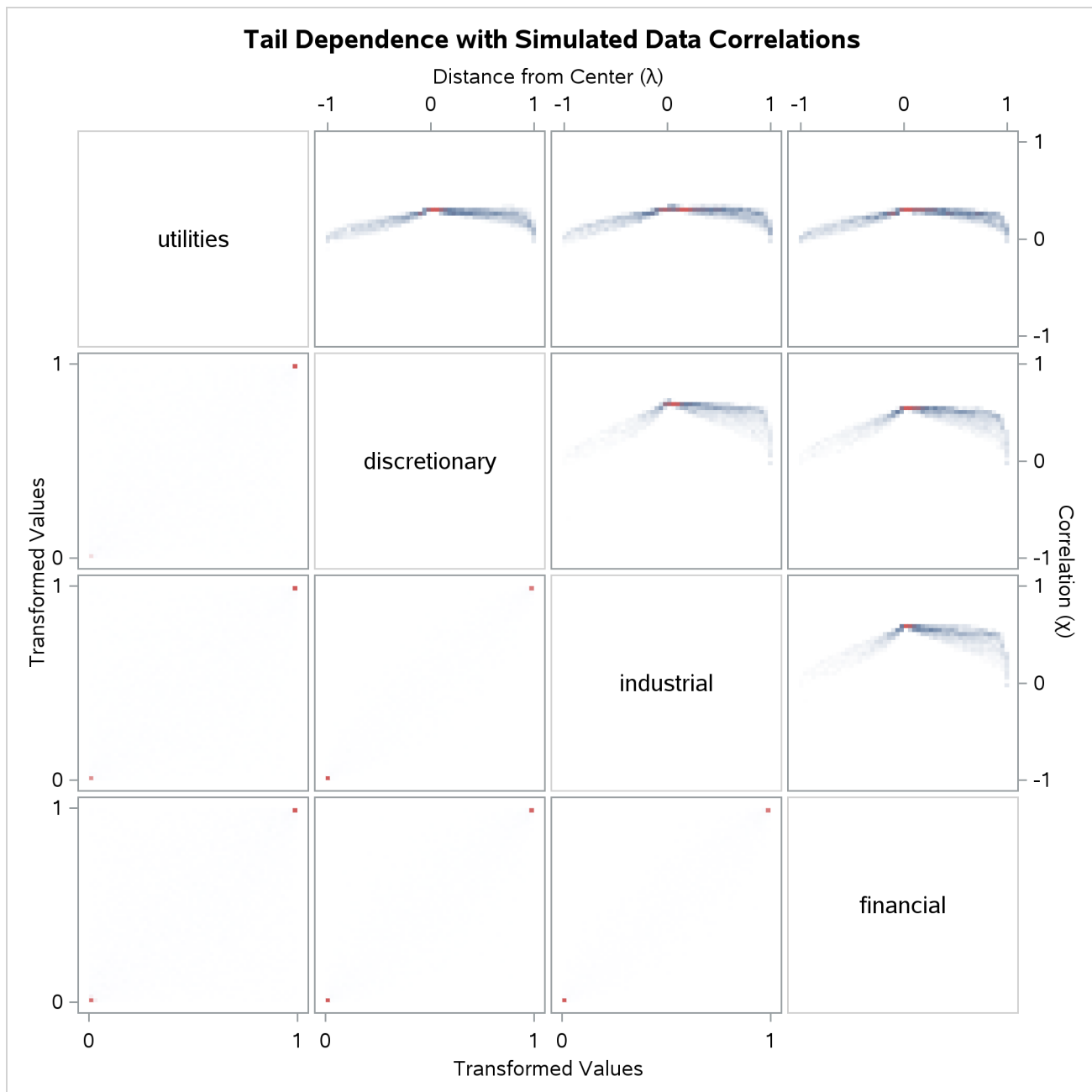
```
/* PROC CCOPULA visualization of sector returns */
proc ccopula data=mylib.sectret;
  var utilities discretionary industrial financial;
  fit t / marginals=empirical plots(tail kendall)=(datatype=uniform)
        store=mylib.sectstor;
  simulate / seed=1 ndraws=5635 outuniform=mylib.outuniform
        plots(tail kendall)=(datatype=uniform);
run;
```

[Output 8.3.1](#) compares the tail plots of the actual return data to plots of the simulated data. Both plots show similar patterns for the dependencies among the various sectors. In particular, the Utilities sector appears to have a consistently low correlation with the other sectors for returns near the mean and for both the high and low extreme values. For the other three sectors, higher correlations are apparent among the returns. Based on the correlation and tail plots in [Output 8.3.1](#), you would be likely to conclude that the estimated t copula represents the dependencies among returns well.

Output 8.3.1 Actual (Upper) and Simulated (Lower) Tail Plots

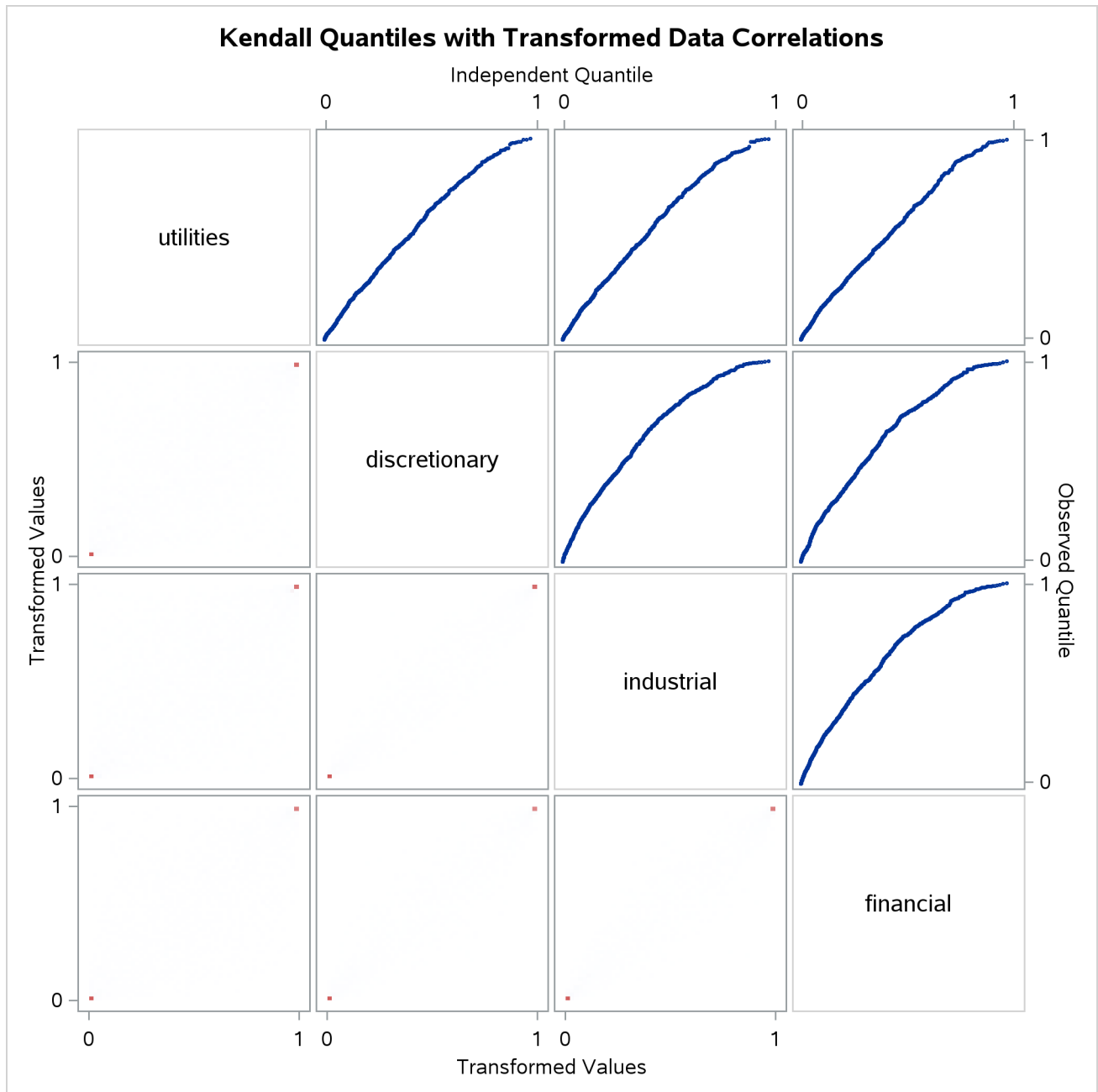


Output 8.3.1 *continued*

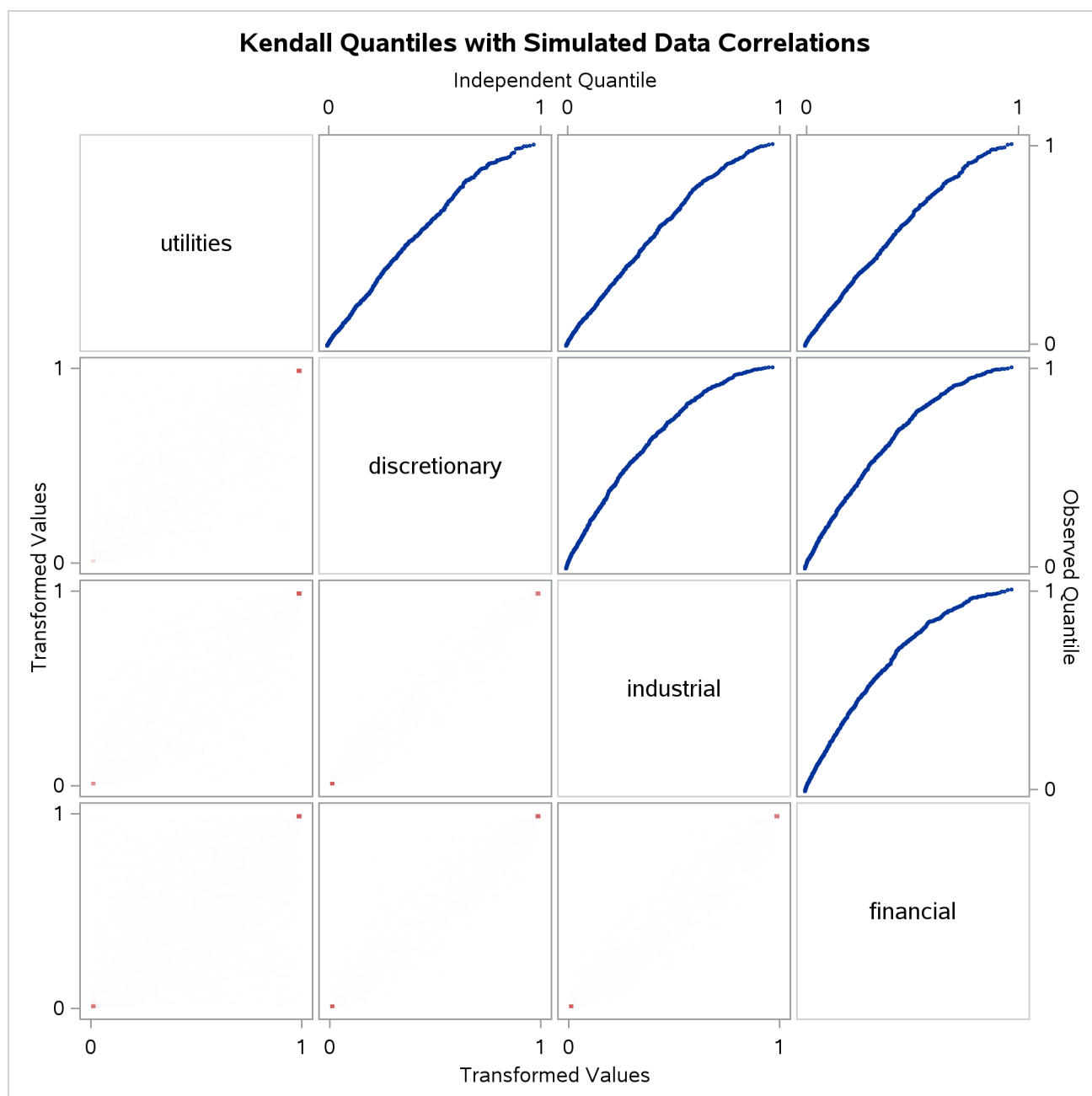


Output 8.3.2 compares the Kendall plots of the actual return data to plots of the simulated data. The Kendall plots for these sectors show that the t copula models the range of returns well. Plots of both the observed and simulated data relative to a uniform bivariate distributions show similar patterns.

Output 8.3.2 Actual (Upper) and Simulated (Lower) Kendall Plots



Output 8.3.2 continued



References

- Bentley, J. (1980). "Multidimensional Divide-and-Conquer." *Communications of the ACM* 23:214–229.
- Cherubini, U., Luciano, E., and Vecchiato, W. (2004). *Copula Methods in Finance*. Chichester, UK: John Wiley & Sons.

- Devroye, L. (1986). *Non-uniform Random Variate Generation*. New York: Springer-Verlag. <http://luc.devroye.org/rnbookindex.html>.
- Fisher, N. I., and Switzer, P. (2001). “Graphical Assessment of Dependence: Is a Picture Worth 100 Tests?” *American Statistician* 55:233–239.
- Genest, C., and Boies, J.-C. (2003). “Detecting Dependence with Kendall Plots.” *American Statistician* 57:275–284.
- Langrené, N., and Warin, X. (2020). “Fast Multivariate Empirical Cumulative Distribution Function with Connection to Kernel Density Estimation.” <https://arxiv.org/abs/2005.03246>.
- Marshall, A. W., and Olkin, I. (1988). “Families of Multivariate Distributions.” *Journal of the American Statistical Association* 83:834–841.
- McNeil, A., Frey, R., and Embrechts, P. (2005). *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton, NJ: Princeton University Press.
- Nolan, J. P. (2010). *Stable Distributions: Models for Heavy Tailed Data*. Boston: Birkhäuser.
- Sklar, A. (1959). “Fonctions de répartition à n dimensions et leurs marges [Distribution functions with n dimensions and their margins].” *Publications de l’Institut de Statistique de L’Université de Paris* 8:229–231.
- Wu, F., Valdez, E., and Sherris, M. (2007). “Simulating from Exchangeable Archimedean Copulas.” *Communications in Statistics—Simulation and Computation* 36:1019–1034.

Chapter 9

CESM Procedure

Contents

Overview: CESM Procedure	338
Using CAS Sessions and CAS Engine Librefs	338
Getting Started: CESM Procedure	339
Syntax: CESM Procedure	340
Functional Summary	340
PROC CESM Statement	341
BY Statement	343
FORECAST Statement	343
ID Statement	346
Details: CESM Procedure	350
Accumulation	350
Missing Value Interpretation	352
Transformations	352
Parameter Estimation	352
Missing Value Modeling Issues	353
Forecasting	353
Inverse Transformations	353
Data Table Output	353
Printed Output	357
Examples: CESM Procedure	358
Example 9.1: Forecasting of Time Series Data	358
Example 9.2: Forecasting of Transactional Data	366
Example 9.3: Specifying the Forecasting Model	370

Overview: CESM Procedure

The CESM procedure generates forecasts by using exponential smoothing models with optimized smoothing weights for one or more time series. Time series data have observations that are equally spaced by a specific time interval (for example, monthly or weekly).

PROC CESM can also generate these forecasts for transactional data, whose observations are not spaced with respect to any particular time interval. The procedure accumulates transactional data on the basis of a specified (or default) time interval to form a time series prior to modeling and forecasting. Typical examples of transactional data are internet, inventory, sales, and other similar data.

For typical time series, you can use the following smoothing models: simple, double, linear, damped trend, seasonal, and Winters method (additive or multiplicative). In addition, transformed versions of the following models are provided: log, square root, logistic, and Box-Cox.

The CESM procedure writes the following to output data tables: the time series that are extrapolated by the forecasts, the series summary statistics, the forecasts and confidence limits, and the parameter estimates.

PROC CESM requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the `CASHOST` and `CASPORT` of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the `TERMINATE` option in the `CAS` statement as follows:

```
cas mysess terminate;
```

For more information about the `CAS` and `LIBNAME` statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: CESM Procedure

The CESM procedure is simple to use and does not require in-depth knowledge of forecasting methods. It can provide results in output data tables. The examples in this section are more fully illustrated in “[Examples: CESM Procedure](#)” on page 358.

The following statements generate forecasts for every numeric variable in the time series in the input data table `mylib.sales`, which contains sales data that are recorded monthly:

```
proc cesm data=mylib.sales outfor=mylib.nextyear;  
  id date interval=month;  
  forecast _numeric_ / lead=12;  
run;
```

The `DATA=` option specifies `mylib.sales` as the input data table, and the `OUTFOR=` option requests that the forecast be output to the `mylib.nextYear` table. The `ID` statement specifies the variable that represents time (`Date`) and specifies that the time series data are spaced monthly. The `_NUMERIC_` specification in the `FORECAST` statement requests that every numeric variable in the input data table be forecast, and the `LEAD=12` options requests that the time series be forecast for the next 12 months.

The following statements show how you can forecast data that contain transactional variables that are not recorded at any specific frequency:

```
proc cesm data=mylib.websites outfor=mylib.nextweek;  
  id time interval=dtday accumulate=total;  
  forecast boats cars planes / lead=7;  
run;
```

The `DATA=` option specifies the input data table `mylib.websites`, which contains three variables (`Boats`, `Cars`, and `Planes`) in internet data that are recorded at no particular time interval. The `OUTFOR=` option requests that forecasts for the total daily values be recorded in the `mylib.nextWeek` data table. The `ID` statement specifies `Time` (which records the time of a website hit) as the variable that represents time and requests that the data be accumulated into a daily time series. The `FORECAST` statement requests that forecasts for the `Boats`, `Cars`, and `Planes` variables be generated. The `LEAD=7` option requests that forecasts be generated for the next seven days. Because the `METHOD=` option is not specified in the `FORECAST` statement, `PROC CESM` fits each series by selecting the best exponential smoothing model (which is the default model) according to the default criterion of root mean square error (RMSE).

Syntax: CESM Procedure

The following statements are available in the CESM procedure:

```
PROC CESM options ;
  BY variables ;
  ID variable INTERVAL= interval < options > ;
  FORECAST variable-list / < options > ;
```

Functional Summary

Table 9.1 summarizes the statements and options that are available in the CESM procedure.

Table 9.1 Functional Summary

Description	Statement	Option
Statements		
Specifies data tables and options	PROC CESM	
Specifies BY-group processing	BY	
Specifies variables to forecast	FORECAST	
Specifies the time ID variable	ID	
Data Set Options		
Specifies the input data table	PROC CESM	DATA=
Specifies the output data table	PROC CESM	OUT=
Specifies the parameter output data table	PROC CESM	OUTEST=
Specifies the forecast output data table	PROC CESM	OUTFOR=
Specify statistics output data table	PROC CESM	OUTSTAT=
Specifies the summary output data table	PROC CESM	OUTSUM=
Accumulation and Seasonality Options		
Specifies the accumulation frequency	ID	INTERVAL=
Specifies the length of seasonal cycle	PROC CESM	SEASONALITY=
Specifies the interval alignment	ID	ALIGN=
Specifies the starting time ID value	ID	START=
Specifies the ending time ID value	ID	END=
Specifies the accumulation statistic	ID, FORECAST	ACCUMULATE=
Specifies how to interpret missing values	ID, FORECAST	SETMISSING=
Forecasting Horizon, Holdback Options		
Specifies the data to hold back	FORECAST	BACK=
Specifies the forecast horizon or lead	FORECAST	LEAD=
Forecasting Model Options		
Specifies the confidence limit width	FORECAST	ALPHA=
Specifies the model selection criterion	FORECAST	CRITERION=

Table 9.1 *continued*

Description	Statement	Option
Specifies the forecast model	FORECAST	METHOD=
Specifies the median forecasts	FORECAST	MEDIAN
Specifies the model transformation	FORECAST	TRANSFORM=
Miscellaneous Options		
Specifies the time ID format	ID	FORMAT=
Specifies the method for trimming the data in the BY groups	ID	TRIMID=

The following sections describe the PROC CESM statement and then describe the other statements in alphabetical order.

PROC CESM Statement

PROC CESM *options* ;

You can specify the following *options*:

DATA=*libref.data-table*

names the input data table for PROC CESM to use. The default is the most recently created data table. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 338.

data-table specifies the name of the input data table.

OUT=*libref.data-table*

names the output data table to contain the forecasts of the variables that are specified in subsequent FORECAST statements. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 338.

The *variable* that is specified in the ID statement is also included in the output data set. The values are accumulated based on the ACCUMULATE= option. The OUT= data table is particularly useful in extending the independent variables.

OUTEST=libref.data-table

names the output data table to contain the model parameter estimates and the associated test statistics and probability values. The OUTEST= data table is useful for evaluating the significance of the model parameters and for understanding the model dynamics.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 338.

OUTFOR=libref.data-table

names the output data table to contain the forecast time series components (actual, predicted, lower confidence limit, upper confidence limit, prediction error, and prediction standard error). The OUTFOR= data table is useful for displaying the forecasts in tabular or graphical form.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 338.

OUTSTAT=libref.data-table

names the output data table to contain the statistics of fit (or goodness-of-fit statistics). The OUTSTAT= data table is useful for evaluating how well the model fits the series. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 338.

OUTSUM=libref.data-table

names the output data table to contain the summary statistics and the forecast summation. The summary statistics are based on the accumulated time series when the ACCUMULATE= or SETMISSING= option is specified in the FORECAST statement. The OUTSUM= data table is useful when large numbers of series are forecasted and a summary of the results is needed.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 338.

SEASONALITY=number

specifies the length of the seasonal cycle. For example, SEASONALITY=3 means that every group of three observations forms a seasonal cycle. This option is applicable only for seasonal forecasting models. By default, the length of the seasonal cycle is 1 (no seasonality) or the length that is implied by the INTERVAL= option in the ID statement. For example, INTERVAL=MONTH implies that the length of the seasonal cycle is 12.

BY Statement

BY *variables* ;

You can use a BY statement to obtain separate dummy variable definitions for groups of observations that are defined by the BY variables.

FORECAST Statement

FORECAST *variable-list* / < *options* > ;

The FORECAST statement lists the numeric variables in the input data table whose accumulated values represent time series to be modeled and forecast. (The input data table is specified in the DATA= option in the PROC CESM statement.)

You must specify a *variable-list* that contains one or more numeric variables. For more information about the *variable-list*, see the section “SAS Variable Lists” in *SAS Programmers Guide: Essentials*.

You can specify any number of FORECAST statements, but you cannot specify the same variable in more than one of them.

You can specify the following *options* to specify which forecast model to use:

ACCUMULATE=*option*

specifies how to accumulate the data table observations within each time period for the variables in the *variable-list*. If the ACCUMULATE= option is not specified in the FORECAST statement, accumulation is determined by the ACCUMULATE= option in the ID statement. Use the ACCUMULATE= option with multiple FORECAST statements when you want different accumulation specifications for different variables. For more information, see the [ACCUMULATE=](#) option in the ID statement.

ALPHA=*number*

specifies the significance level to use in computing the confidence limits of the forecast, where *number* must be between 0 and 1. By default, ALPHA=0.05, which produces 95% confidence intervals.

BACK=*n*

specifies the number of observations before the end of the data where the multistep forecasts are to begin. By default, BACK=0.

CRITERION=*criterion*

specifies the model selection criterion (statistic of fit) to use to select from several candidate models. The default is CRITERION=RMSE. The following statistics of fit are provided:

SSE	sum of square error
MSE	mean squared error
RMSE	root mean squared error
UMSE	unbiased mean squared error
URMSE	unbiased root mean squared error
MAXPE	maximum percent error

MINPE	minimum percent error
MPE	mean percent error
MAPE	mean absolute percent error
MDAPE	median percent error
GMAPE	geometric mean percent error
MAPES	mean absolute error percent of standard deviation
MDAPES	median absolute error percent of standard deviation
GMAPES	geometric mean absolute error percent of standard deviation
MINPPE	minimum predictive percent error
MAXPPE	maximum predictive percent error
MPPE	mean predictive percent error
MAPPE	symmetric mean absolute predictive percent error
MDAPPE	median predictive percent error
GMAPPE	geometric mean predictive percent error
MINSPE	minimum symmetric percent error
MAXSPE	maximum symmetric percent error
MSPE	mean symmetric percent error
SMAPE	symmetric mean absolute percent error
MDASPE	median symmetric percent error
GMASPE	geometric mean symmetric percent error
MINRE	minimum relative error
MAXRE	maximum relative error
MRE	mean relative error
MRAE	mean relative absolute error
MDRAE	median relative absolute error
GMRAE	geometric mean relative absolute error
MAXERR	maximum error
MINERR	minimum error
ME	mean error
MAE	mean absolute error
MASE	mean absolute scaled error
RSQUARE	R-square
ADJRSQ	adjusted R-square
AADJRSQ	Amemiya's adjusted R-square
RWRSQ	random walk R-square

AIC	Akaike information criterion
AICC	Akaike information corrected criterion
SBC	Schwarz Bayesian information criterion
APC	Amemiya's prediction criterion

LEAD=*n*

specifies the number of periods ahead to forecast (the forecast lead or horizon).

The value *n* is not relative to the last nonmissing observation of a particular series, but is instead relative to the BACK= option specification to the last observation in the input data table or the accumulated series. Thus, if a series has missing values at the end, the actual number of forecasts computed for that series is greater than *n*.

By default, LEAD=0.

MEDIAN

estimates the median forecast values and uses those values for forecasting. (By default, PROC CESM uses mean values for forecasting.) If you do not specify the TRANSFORM= option, no transformation is applied to the time series, so the mean and median forecast values are identical.

METHOD=*model-name*

specifies the forecasting model to use to forecast the time series. You can specify the following forecasting *model-names*:

ADDWINTERS	requests the Winters additive method.
BEST	requests the best candidate smoothing model among the SIMPLE, LINEAR, DAMPTREND, SEASONAL, ADDWINTERS, or WINTERS methods.
BESTN	requests the best candidate nonseasonal smoothing model among the SIMPLE, LINEAR, or DAMPTREND methods.
BESTS	requests the best candidate seasonal smoothing model among the SEASONAL, ADDWINTERS, or WINTERS methods.
DAMPTREND	requests damped trend exponential smoothing.
DOUBLE	requests second-order exponential smoothing.
LINEAR	requests linear (Holt) exponential smoothing.
MULTSEASONAL	requests multiplicative seasonal exponential smoothing.
SEASONAL	requests additive seasonal exponential smoothing.
SIMPLE	requests simple (single) exponential smoothing.
WINTERS	requests Winters multiplicative method.

By default, METHOD=BEST.

SETMISSING=*option* | *number*

specifies how to assign missing values (either input or accumulated) in the accumulated time series for variables in the *variable-list*. If the SETMISSING= option is not specified in the FORECAST statement, missing values are set according to the value of the SETMISSING= option in the ID statement. For more information, see the SETMISSING= option in the ID statement.

TRANSFORM=*option*

specifies the time series transformation to be applied to the input or accumulated time series. You can specify the following values for *option*:

AUTO	automatically chooses between NONE and LOG on the basis of the model selection criteria.
BOXCOX (<i>n</i>)	performs Box-Cox transformation with parameter number (<i>n</i>), where <i>n</i> must be between –5 and 5.
LOG	performs logarithmic transformation.
LOGISTIC	performs logistic transformation.
NONE	performs no transformation.
SQRT	performs square-root transformation.

By default, TRANSFORM=NONE.

When the TRANSFORM= option is specified, the time series must be strictly positive. After the time series is transformed, the model parameters are estimated by using the transformed series. The forecasts of the transformed series are then computed, and finally the transformed series forecasts are inverse-transformed. The inverse transform produces either mean or median forecasts depending on whether the MEDIAN option is specified. For more information, see the sections “[Transformations](#)” on page 352 and “[Inverse Transformations](#)” on page 353.

ID Statement

ID *variable* **INTERVAL=***interval* < *options* > ;

The ID statement names a numeric *variable* that identifies observations in the input and output data tables. The values of the ID *variable* are assumed to be SAS date or datetime values. In addition, the ID statement specifies (in the INTERVAL= option) the frequency that is associated with the time series. The *options* also specify how to accumulate the observations and how to align the time ID values to form the time series to be forecast. The specified information affects all variables that are specified in subsequent FORECAST statements. If an ID statement is not specified, the observation number (with respect to the BY group) is used as the time ID.

You must specify the following argument:

INTERVAL=*interval*

specifies the frequency of the input time series or of the time series to be accumulated from the input data. For example, if the input data table consists of quarterly observations, then **INTERVAL=QTR** should be used. If the **SEASONALITY=** option is not specified, the length of the seasonal cycle is implied by the **INTERVAL=** option. For example, **INTERVAL=QTR** implies a seasonal cycle of length 4. If the **ACCUMULATE=** option is also specified, the **INTERVAL=** option determines the time periods for the accumulation of observations.

The basic intervals are **YEAR**, **SEMIYEAR**, **QTR**, **MONTH**, **SEMIMONTH**, **TENDAY**, **WEEK**, **WEEKDAY**, **DAY**, **HOURL**, **MINUTE**, and **SECOND**. For more information about the intervals that can be specified, see the chapter “Date Intervals, Formats, and Functions” in *SAS/ETS User’s Guide*.

You can also specify the following *options*:

ACCUMULATE=*option*

specifies how to accumulate the data table observations within each time period. The frequency (width of each time interval) is specified in the **INTERVAL=** option. The **ID variable** contains the time ID values. Each value of the time ID *variable* value corresponds to a specific time period. The accumulated values form the time series, which is used in subsequent model fitting and forecasting.

The **ACCUMULATE=** option is particularly useful when there are gaps in the input data or when there are multiple input observations that coincide with a particular time period (for example, transactional data). The **EXPAND** procedure in *SAS/ETS User’s Guide* offers additional frequency conversions and transformations that can also be useful in creating a time series.

You can specify the following *options* to determine how to accumulate the observations within each time period based on the **ID variable** and the frequency that is specified in the **INTERVAL=** option:

TOTAL SUM	accumulates observations based on the total sum of their values.
AVERAGE AVG	accumulates observations based on the average of their values.
MINIMUM MIN	accumulates observations based on the minimum of their values.
MAXIMUM MAX	accumulates observations based on the maximum of their values.
N	accumulates observations based on the number of nonmissing observations.
NMISS	accumulates observations based on the number of missing observations.
STDDEV STD	accumulates observations based on the standard deviation of their values.
CSS	accumulates observations based on the corrected sum of squares of their values.
USS	accumulates observations based on the uncorrected sum of squares of their values.

By default, **ACCUMULATE=TOTAL**.

If the **ACCUMULATE=** option is specified, the **SETMISSING=** option is useful for specifying how to treat accumulated missing values. If missing values should be interpreted as 0, then you should specify **SETMISSING=0**. For more information about accumulation, see the section “[Accumulation](#)” on page 350.

ALIGN=option

controls the alignment of SAS date or datetime values that are used to identify the time period of output observations. Although any date or datetime value within the time period can identify the time period, this option requests that the representative date or datetime for the time period be calculated as the beginning date or datetime of the time period, the ending date or datetime of the time period, or the middle date or datetime of the time period. In addition to aligning the time ID values consistently for observations that are supplied by the user, this option specifies the method for calculating the time ID values for observations in the forecast and backcast time periods, which often are not supplied by the user. You can specify the following *options*:

- BEGINNING | BEG | B** represents each time period by using the beginning SAS date or datetime value of the time period.
- ENDING | END | E** represents each time period by using the ending SAS date or datetime value of the time period.
- MIDDLE | MID | M** represents each time period by using the middle SAS date or datetime value of the time period. The middle is calculated as the average of the beginning and ending values.

By default, ALIGN=BEGINNING.

END=date | datetime

specifies a SAS date or datetime literal value that represents the end of the data. If the value of the last time ID variable is less than the END= value, the series is extended with missing values. If the value of the last time ID variable is greater than the END= value, the series is truncated. For example, **END='1jan2008'D** requests that data for time periods after the first of January 2008 not be used. The option **END='&sysdate'D** uses the automatic macro variable SYSDATE to extend or truncate the series to the current date. You can use this option and the START= option to ensure that data that are associated with each BY group contain the same number of observations.

FORMAT=format

specifies the SAS format for the time ID values. If the FORMAT= option is not specified, the default format is implied by the INTERVAL= option.

SETMISSING=n | option

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series. You can specify either a number (*n*) or an *option* to determine how to interpret missing values:

- n* interprets a missing value as having the value *n*. You can specify any number for *n*, but not a missing value. If a missing value indicates a 0 value, specify SETMISSING=0. You typically use SETMISSING=0 for transactional data because no recorded data usually implies no activity.
- MISSING** interprets a missing value as a missing value. Use this option if a missing value indicates an unknown value.
- AVERAGE | AVG** interprets a missing value as the average value of all accumulated nonmissing values in the span of the series.
- MINIMUM | MIN** interprets a missing value as the minimum value of all accumulated nonmissing values in the span of the series.

MEDIAN MED	interprets a missing value as the median value of all accumulated nonmissing values in the span of the series.
MAXIMUM MAX	interprets a missing value as the maximum value of all accumulated nonmissing values in the span of the series.
FIRST	interprets a missing value as the first nonmissing value of all accumulated nonmissing values in the span of the series.
LAST	interprets a missing value as the last nonmissing value of all accumulated nonmissing values in the span of the series.
PREVIOUS PREV	interprets a missing value as the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.
NEXT	interprets a missing value as the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.

By default, SETMISSING=MISSING.

START=*date* | *datetime*

specifies a SAS date or datetime literal value that represents the beginning of the data. If the value of the first time ID variable is greater than the START= value, the series is prefixed with missing values. If the value of the first time ID variable is less than the START= value, the series is truncated. You can use this option and the END= option to ensure that data that are associated with each BY group contain the same number of observations.

TRIMID=*method*

specifies the method for trimming the data in the BY groups. The output time ID variable span that is calculated by the *method* is dependent on the input time ID variable span, irrespective of missing values of the time series variables. Depending on the *method* and the input time ID variable data, leading or trailing missing values can be added to the time series variables.

After the output time ID variable span is calculated by the *method*, the ending value of the output time ID variable is recalculated according to the value of the LEAD= option in the FORECAST statement (if the LEAD= option is specified).

You can specify one of the following *methods*:

NONE	uses the same starting and ending values of the output time ID variable for all BY groups. The span of the output time ID variable includes all values that are input as a time ID value for all BY groups. The time series variables are extended with leading or trailing missing values as required.
LEFT	uses the identifying date for the first time period that is input for the BY group as the starting value of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading missing values. The ending value of the time ID variable is the same for all BY groups. The time series variables are extended with trailing missing values as required.
RIGHT	uses the identifying date for the last time period that is input for the BY group as the ending value of the output time ID variable for each BY group. The time series values in each BY group are not extended with trailing missing values. The starting value of the time ID

variable is the same for all BY groups. The time series variables are extended with leading missing values as required.

BOTH uses the span of the input time ID variable for the BY group as the span of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading or trailing missing values.

By default, TRIMID=NONE.

Details: CESM Procedure

You can use the CESM procedure to forecast both time series data and transactional data. If the data are transactional, then the procedure must first accumulate the data into a time series before it can forecast them. [Table 9.2](#) shows the sequential steps that the procedure uses to produce forecasts; the options that control each step and the statements where those options can be specified are shown in the third and fourth columns, respectively.

Table 9.2 CESM Processing Steps and Control Options

Step	Operation	Option	Statements
1	Accumulation	ACCUMULATE=	ID, FORECAST
2	Missing value interpretation	SETMISSING=	ID, FORECAST
3	Transformations	TRANSFORM=	FORECAST
4	Parameter estimation	METHOD=	FORECAST
5	Forecasting	METHOD=, LEAD=	FORECAST
6	Inverse transformation	TRANSFORM, MEDIAN	FORECAST

Each of the steps shown in [Table 9.2](#) is described in the following sections.

Accumulation

If the ACCUMULATE= option is specified in the ID statement, data table observations are accumulated within each time period. The frequency (width of each time interval) is specified in the INTERVAL= option in the ID statement, and the variable that is specified in the ID statement contains the time ID values. Each time ID value corresponds to a specific time period. Accumulation is particularly useful when the input data table contains transactional data, whose observations are not spaced with respect to any particular time interval. The accumulated values form the time series that is used in subsequent analyses by the CESM procedure.

For example, suppose a data table contains the following observations:

```
19MAR1999    10
19MAR1999    30
11MAY1999    50
12MAY1999    20
```


23MAY1999 20

If the INTERVAL=MONTH option is specified in the ID statement, all the preceding observations fall within three time periods: March 1999, April 1999, and May 1999. The observations are accumulated within each time period as follows.

- If the ACCUMULATE=NONE option is specified, an error is generated because the ID variable values are not equally spaced with respect to the specified frequency (MONTH).
- If the ACCUMULATE=TOTAL option is specified, the resulting time series is

O1MAR1999	40
O1APR1999	.
O1MAY1999	90

- If the ACCUMULATE=AVERAGE option is specified, the resulting time series is

O1MAR1999	20
O1APR1999	.
O1MAY1999	30

- If the ACCUMULATE=MINIMUM option is specified, the resulting time series is

O1MAR1999	10
O1APR1999	.
O1MAY1999	20

- If the ACCUMULATE=MEDIAN option is specified, the resulting time series is

O1MAR1999	20
O1APR1999	.
O1MAY1999	20

- If the ACCUMULATE=MAXIMUM option is specified, the resulting time series is

O1MAR1999	30
O1APR1999	.
O1MAY1999	50

- If the ACCUMULATE=FIRST option is specified, the resulting time series is

O1MAR1999	10
O1APR1999	.
O1MAY1999	50

- If the ACCUMULATE=LAST option is specified, the resulting time series is

O1MAR1999	30
O1APR1999	.
O1MAY1999	20

- If the ACCUMULATE=STDDEV option is specified, the resulting time series is

O1MAR1999	14.14
O1APR1999	.
O1MAY1999	17.32

The preceding examples demonstrate that even when the data observations contain no missing values, the accumulated time series can have missing values.

Missing Value Interpretation

Sometimes missing values should be interpreted as truly unknown values and should be retained as missing values in the data table. The forecasting models used by the CESM procedure can effectively handle missing values; see the section “[Missing Value Modeling Issues](#)” on page 353. However, sometimes missing values are known, such as when missing values are created from accumulation and represent no observed values for the variable. In this case, the value for the period should be interpreted as 0 (no values), and the SETMISSING=0 option should be used to cause PROC CESM to recode missing values as 0. In other cases, missing values should be interpreted as global values, such as minimum or maximum values of the accumulated series. The accumulated and missing-value-recoded time series is used in subsequent analyses in PROC CESM.

Transformations

If the TRANSFORM= option is specified in the FORECAST statement, the time series is transformed prior to model parameter estimation and forecasting. Only strictly positive series can be transformed. An error is generated when the TRANSFORM= option is used with a nonpositive series.

Parameter Estimation

All the parameters (smoothing weights) that are associated with the exponential smoothing model that is used to forecast the time series (as specified by the METHOD= option in the PROC CESM statement) are optimized based on the data, with the default parameter restrictions imposed. If the TRANSFORM= option is specified in the FORECAST statement, the transformed time series data are used to estimate the model parameters.

Missing Value Modeling Issues

The treatment of missing values varies with the forecasting model. Missing values after the start of the series are replaced with one-step-ahead predicted values, and the predicted values are used in the smoothing equations.

The treatment of missing values can also be specified in the SETMISSING= option, which changes the missing values prior to modeling.

NOTE: Even if all the observed data are nonmissing, the ACCUMULATE= option can create missing values in the accumulated series (when the data contain no observations for some of the time periods that are specified in the INTERVAL= option in the ID statement).

Forecasting

After the model parameters are estimated, one-step-ahead forecasts are generated for the full range of the accumulated and optionally transformed time series data, and multistep forecasts are generated from the end of the time series to the future time period that is specified in the LEAD= option in the PROC CESM statement. If there are missing values at the end of the time series, the forecast horizon will be greater than that specified by the LEAD= option.

Inverse Transformations

If the TRANSFORM= option is specified in the FORECAST statement, the forecasts of the transformed time series are inverse-transformed. By default, forecasts of the mean (expected value) are generated. If the MEDIAN option is specified in the PROC CESM statement, median forecasts are generated.

Data Table Output

The CESM procedure can create the OUT=, OUTEST=, OUTFOR=, OUTSTAT=, and OUTSUM= data tables. These data tables contain the variables that are specified in the BY statement and statistics that are related to the variables that are listed in the FORECAST statement. In general, if a forecasting step that is related to an output data table fails, the values of this step are not recorded or are set to missing in the related output data table and appropriate error or warning (or both) messages are recorded in the log.

OUT= Data Table

The OUT= data table contains the variables that are specified in the BY, ID, and FORECAST statements. If the ID statement is specified, the ID variable values are aligned and extended based on the ALIGN= and INTERVAL= options in the ID statement. The values of the variables that are specified in the FORECAST statements are accumulated based on the ACCUMULATE= option in the FORECAST statement, and missing values are interpreted based on the SETMISSING= option in the FORECAST statement.

If any of the forecasting steps fail for a particular variable, the variable is extended by missing values.

OUTEST= Data Table

The OUTEST= data table contains observations that are related to the parameter estimation step. The column names of the table are the BY variables and the following names:

NAME	variable name
MODEL	forecasting model
MODELVAR	model variable
DSVAR	data variable
VARTYPE	variable type
TRANSFORM	transformation
COMPONENT	component
COMPMODEL	component model
PARM	parameter name
FACTOR	factor
LAG	lag
SHIFT	shift
EST	value of parameter estimate
STDERR	standard error of parameter estimate
TVALUE	<i>t</i> values of parameter estimate
PVALUE	<i>p</i> -values of parameter estimate

If the parameter estimation step fails for a particular variable, no observations are output to the OUTEST= data table for that variable.

OUTFOR= Data Table

The OUTFOR= data table contains observations that are related to the forecasting step. The column names of the table are the BY variables and the following names:

NAME	variable name
ID Variable	values of the variable specified in the ID statement
ACTUAL	actual values
PREDICT	predicted values
ERROR	prediction errors
STD	prediction standard errors
UPPER	upper confidence limits
LOWER	lower confidence limits

If the forecasting step fails for a particular variable, no observations are recorded in the OUTFOR= data table for that variable. If the TRANSFORM= option is specified in the FORECAST statement, the values in the preceding variables are the inverse-transform forecasts. If the MEDIAN option is specified in the FORECAST statement, the median forecasts are stored; otherwise, the mean forecasts are stored.

OUTSTAT= Data Set

The OUTSTAT= data table contains observations that are related to the statistics of fit. The column names of the table are the BY variables and the following names:

NAME	variable name
REGION	region in which the statistics are calculated. Statistics that are calculated in the fit region are indicated by FIT. Statistics that are calculated in the forecast region (which happens only if the value of the BACK= option is greater than zero) are indicated by FORECAST.
MODEL	model name
DFE	degrees of freedom error
N	number of observations
NOBS	number of observations used
NMISSA	number of missing actuals
NMISSP	number of missing predicted values
NPARMS	number of parameters
TSS	total sum of squares
SST	corrected total sum of squares
SSE	sum of square error
MSE	mean square error
RMSE	root mean square error
UMSE	unbiased mean square error
URMSE	unbiased root mean square error
MAPE	mean absolute percentage error
MAE	mean absolute error
RSQUARE	R-square
ADJRSQ	adjusted R-square
AADJRSQ	Amemiya's adjusted R-square
RWRSQ	random walk R-square
AIC	Akaike's information criterion
AICC	finite sample corrected AIC
SBC	Schwarz Bayesian information criterion
APC	Amemiya's prediction criterion
MAXERR	maximum error
MINERR	minimum error
MAXPE	maximum percentage error
MINPE	minimum percentage error
ME	mean error

MPE	mean percentage error
MDAPE	median absolute percentage error
GMAPE	geometric mean absolute percentage error
MINPPE	minimum predictive percentage error
MAXPPE	maximum predictive percentage error
MPPE	mean predictive percentage error
MAPPE	symmetric mean absolute predictive percentage error
MDAPPE	median absolute predictive percentage error
GMAPPE	geometric mean absolute predictive percentage error
MINSPE	minimum symmetric percentage error
MAXSPE	maximum symmetric percentage error
MSPE	mean symmetric percentage error
SMAPE	symmetric mean absolute percentage error
MDASPE	median absolute symmetric percentage error
GMASPE	geometric mean absolute symmetric percentage error
MINRE	minimum relative error
MAXRE	maximum relative error
MRE	mean relative error
MRAE	mean relative absolute error
MDRAE	median relative absolute error
GMRAE	geometric mean relative absolute error
MASE	mean absolute scaled error
MINAPES	minimum absolute error percentage of standard deviation
MAXAPES	maximum absolute error percentage of standard deviation
MAPES	mean absolute error percentage of standard deviation
MDAPES	median absolute error percentage of standard deviation
GMAPES	geometric mean absolute error percentage of standard deviation

If the statistics of fit cannot be computed for a particular variable, no observations are recorded in the OUTSTAT= data table for that variable. If the TRANSFORM= option is specified, the values in the preceding variables are computed on the basis of the inverse transform forecasts. If the MEDIAN option is specified, the median forecasts are the basis; otherwise, the mean forecasts are the basis.

OUTSUM= Data Table

The OUTSUM= data table contains the variables that are specified in the BY statement in addition to the variables that are listed in this section. The OUTSUM= data table records the summary statistics for each variable that is specified in a FORECAST statement.

The following variables, which are related to summary statistics, are based on the ACCUMULATE= and SETMISSING= options:

NAME	variable name
NOBS	number of observations
N	number of nonmissing observations
NMISS	number of missing observations
MIN	minimum value
MAX	maximum value
MEAN	mean value
STDDEV	standard deviation
STATUS	forecasting status; nonzero values imply that no forecast was generated for the series
START	the starting date of each series
END	the ending date of each series
STARTOBS	the beginning observation number of each series
ENDOBS	the ending observation number of each series

Printed Output

The CESM procedure always prints a summary of the processing that is performed on the time series data. This is extremely useful for gauging the work that is performed by the CAS server when it executes PROC CESM. Printing of other results is best accomplished by the use of targeted data queries to subset and display the information from the tables that are produced by the CESM procedure. For example, you might want to print the OUTFOR= results only for BY groups that have a particular _STATUS_ value in the OUTSUM= table.

Examples: CESM Procedure

Example 9.1: Forecasting of Time Series Data

This example uses retail sales data to illustrate how you can use the CESM procedure to forecast time series data.

The statements in this example assume that your SAS library is named mylib, but you can substitute any appropriately defined SAS library.

The following DATA step creates a data table from data that were recorded monthly at numerous points of sale. The data table, mylib.sales, contains a variable Date (which represents time) and a variable for each sales item. Each value of the Date variable is recorded in ascending order, and the values of each of the other variables represent a single time series.

```
data mylib.sales;
    format date date9.;
    input date : date9. shoes socks laces dresses
          coats shirts ties belts hats blouses;

    datalines;
01JAN1994 3557 3718 6368.80 575 987 10.8200 15.0000 102.600 12410 15013
01FEB1994 5128 4174 8123.20 565 1000 12.1200 15.1000 99.900 13556 12413
01MAR1994 5222 4482 7807.20 406 1005 11.7800 15.3000 102.000 11063 12752

    ... more lines ...

01DEC1998 5399 4795 6075.30 614 1239 21.2000 1.9000 79.700 23004 22044
01JAN1999 6405 4981 6812.10 607 1196 20.7000 14.9000 99.900 20583 26093
;
```

The following CESM procedure statements forecast each of the monthly time series by using the simple exponential smoothing model:

```
proc cesm data=mylib.sales outfor=mylib.nextyear;
    id date interval=month;
    forecast _numeric_ / lead=12 method=simple;
run;
```

The preceding statements generate forecasts for every numeric variable in the input data table mylib.sales for the next 12 months and store these forecasts in the output data table mylib.nextYear. The summary of time series processing is output in [Output 9.1.1](#).

Output 9.1.1 Summary of Time Series Processing
The CESM Procedure

Processing Summary	
CAS table	SALES
Number of analysis variables	10
Number of rows read	122
Number of groups read	1
Memory for group packages (KB)	33
Time to load groups (seconds)	0.0119960308
Minimum time ID	JAN1994
Maximum time ID	JAN1999
Minimum time periods	61
Maximum time periods	61
Number of nodes run	4
Number of nodes with data	3
Number of nodes with groups	1
Number of threads budgeted	1
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	1
Maximum threads active	1
Number of groups processed by submitted code	1
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0730199814

The following statements plot the forecasts for the sale of belts, blouses, and laces:

```
%macro plotActualPredict (ds,timeId,var,refValue,
                        xStart,xEnd,xBy,xLabel,yStart,yEnd,yBy,yLabel);
    proc sgplot data=&ds. (where=(_name_="&var."));
        series x=&timeId. y=actual / markers lineattrs=(color=red)
            markerattrs=(symbol=circlefilled color=red);
        series x=&timeId. y=predict / markers lineattrs=(color=blue)
            markerattrs=(symbol=asterisk color=blue);
        reffline &refValue. / axis=x;
        xaxis values=(&xStart. to &xEnd. by &xBy.) label=&xLabel;
        yaxis values=(&yStart. to &yEnd. by &yBy.) label=&yLabel minor;
    run;
%mend;

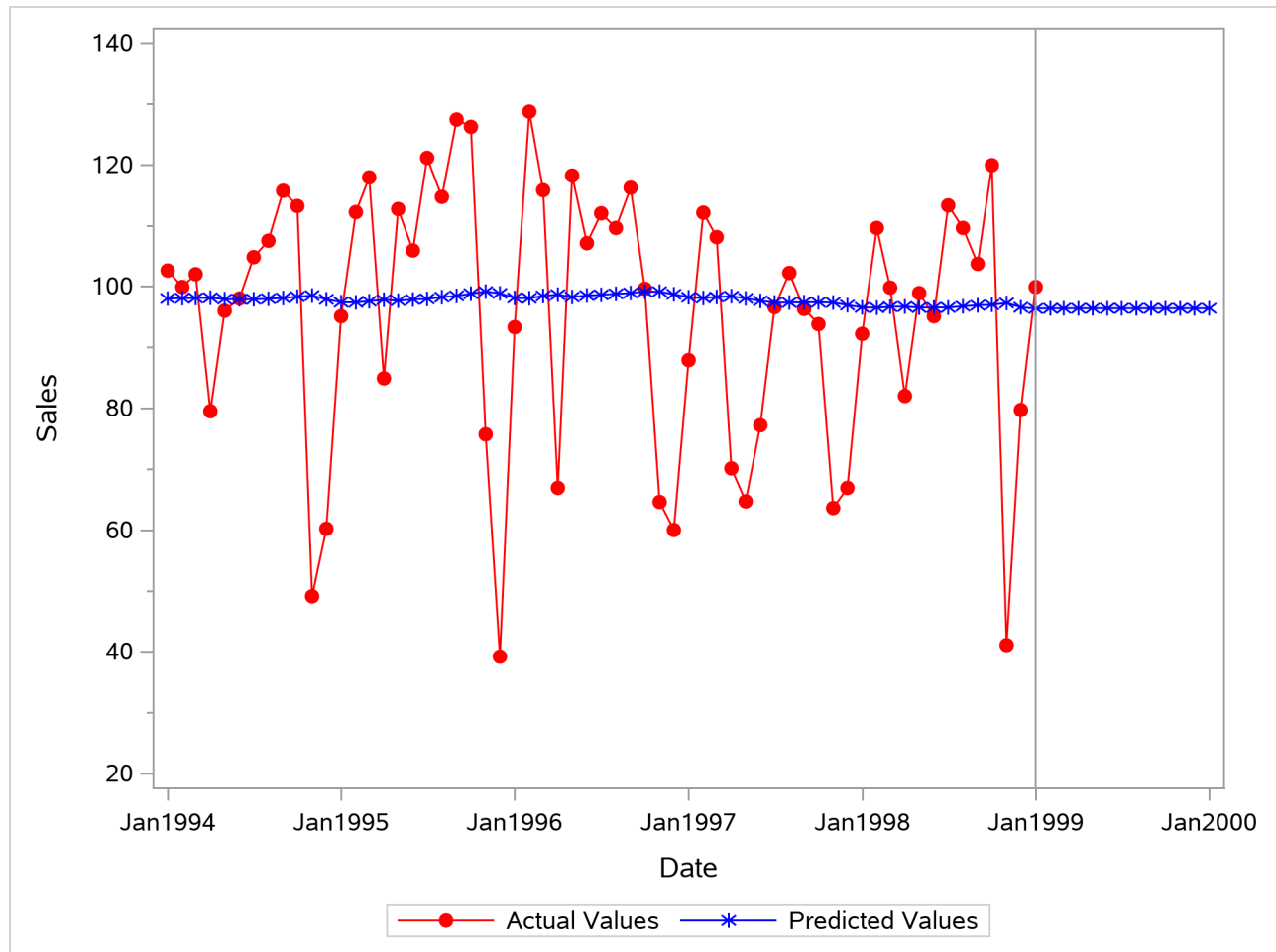
%plotActualPredict(mylib.nextyear, date, belts, '01JAN1999'd,
    '01JAN1994'd, '01DEC2000'd, year, 'Date',
    20, 140, 20, 'Sales');

%plotActualPredict(mylib.nextyear, date, blouses, '01JAN1999'd,
    '01JAN1994'd, '01DEC2000'd, year, 'Date',
    8000, 36000, 4000, 'Sales');
```

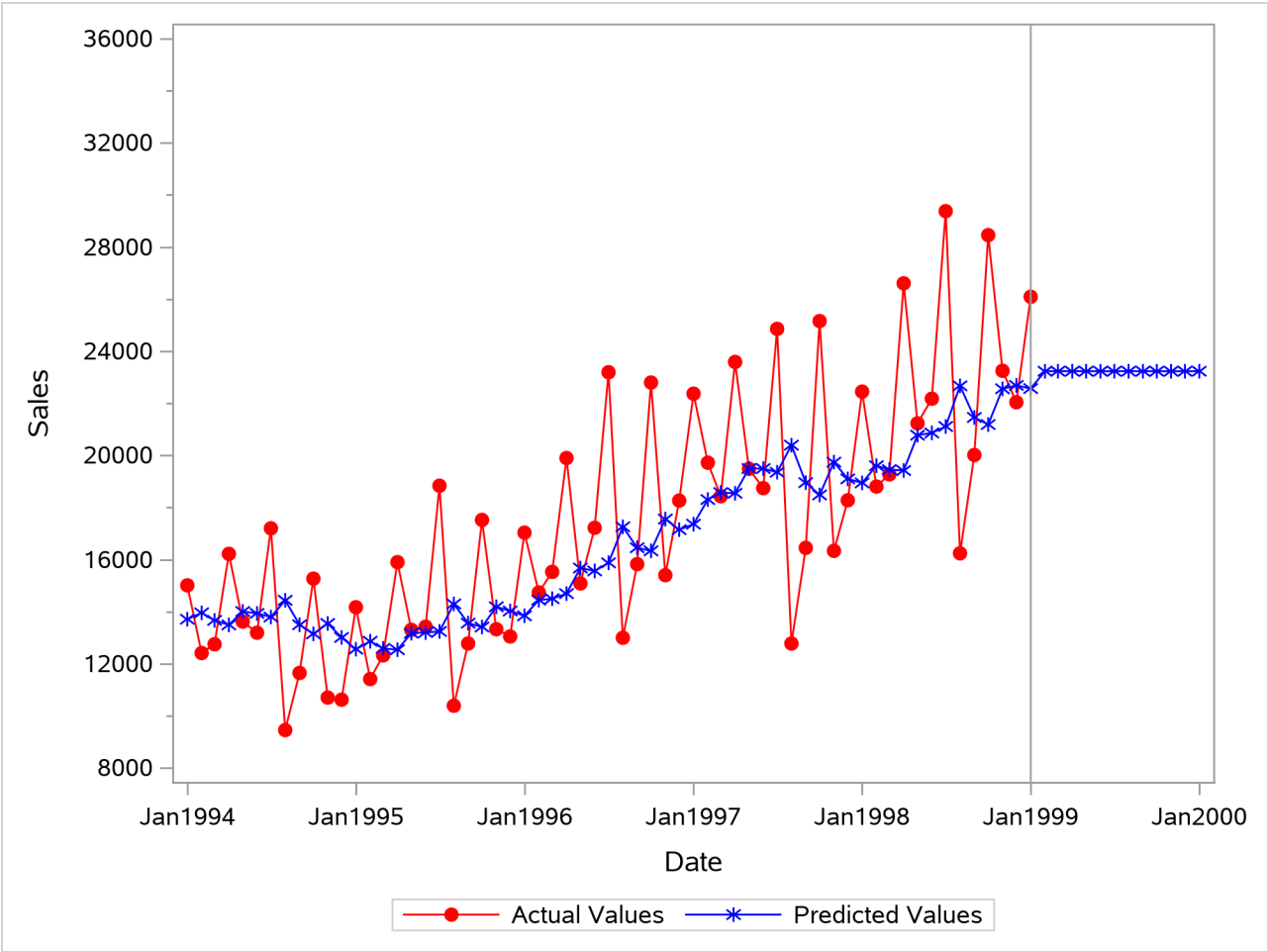
```
%plotActualPredict(mylib.nextyear, date, laces, '01JAN1999'd,
'01JAN1994'd, '01DEC2000'd, year, 'Date',
5000, 10000, 1000, 'Sales');
```

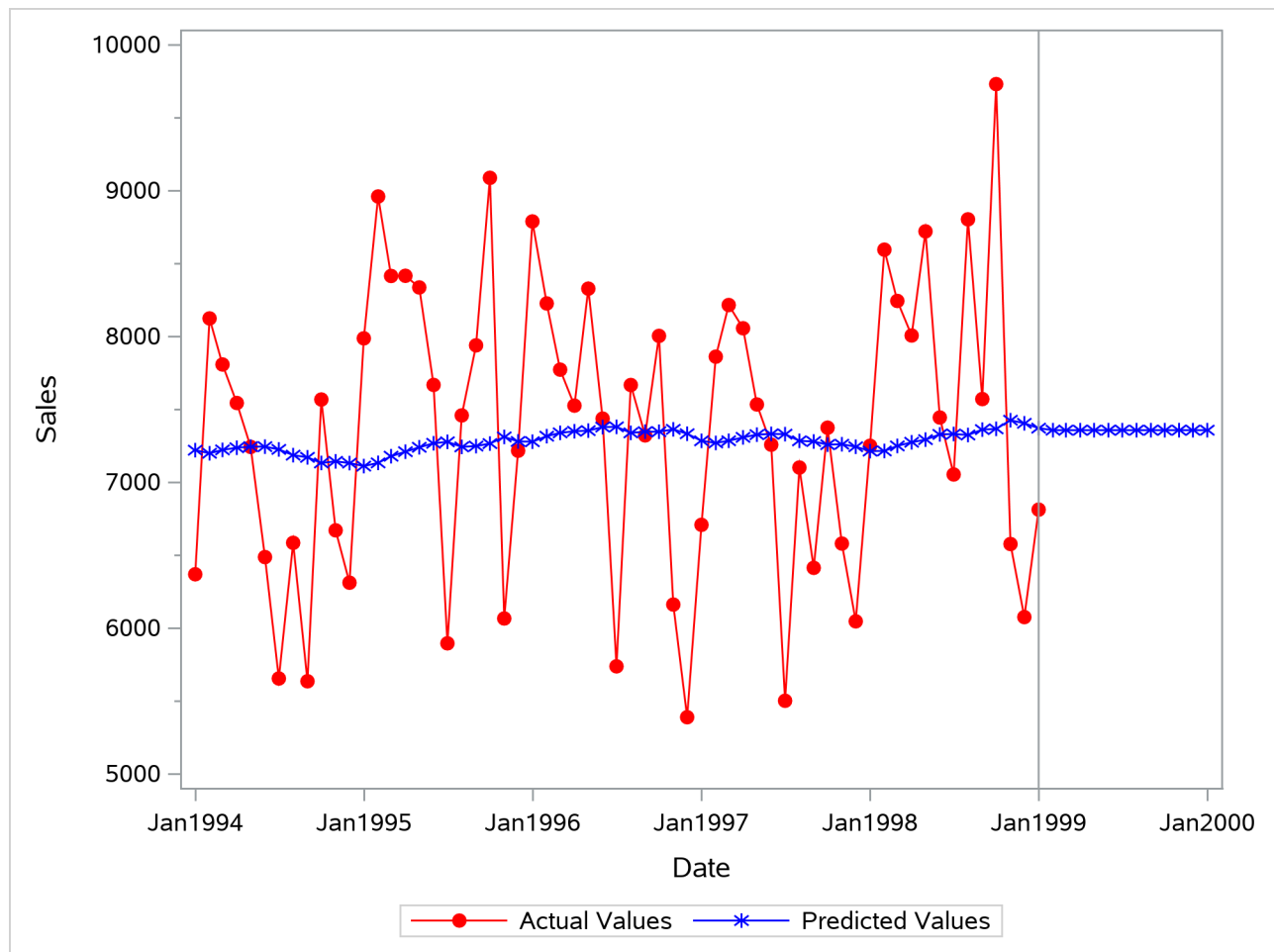
The plots are shown in [Output 9.1.2](#), [Output 9.1.3](#), and [Output 9.1.4](#). The historical actual values and their predicted values are shown to the left of the reference line, and the forecasts for the next 12 monthly periods are shown to the right. For simple exponential smoothing models, the forecasts are constant. The plots seem to indicate that the fit of the predicted values to the actual values is not good.

Output 9.1.2 Retail Sales Forecast Plots (Belts)



Output 9.1.3 Retail Sales Forecast Plots (Blouses)



Output 9.1.4 Retail Sales Forecast Plots (Laces)

The following statements forecast each of the monthly time series by using the default METHOD=BEST option:

```
proc cesm data=mylib.sales outfor=mylib.nextyear outest=mylib.salesEst;
  id date interval=month;
  forecast _numeric_ / lead=12;
run;
```

The best method for each variable is selected according to the criterion (RMSE, the default value, in this case). The following statements plot the forecasts for the sale of belts, blouses, and laces:

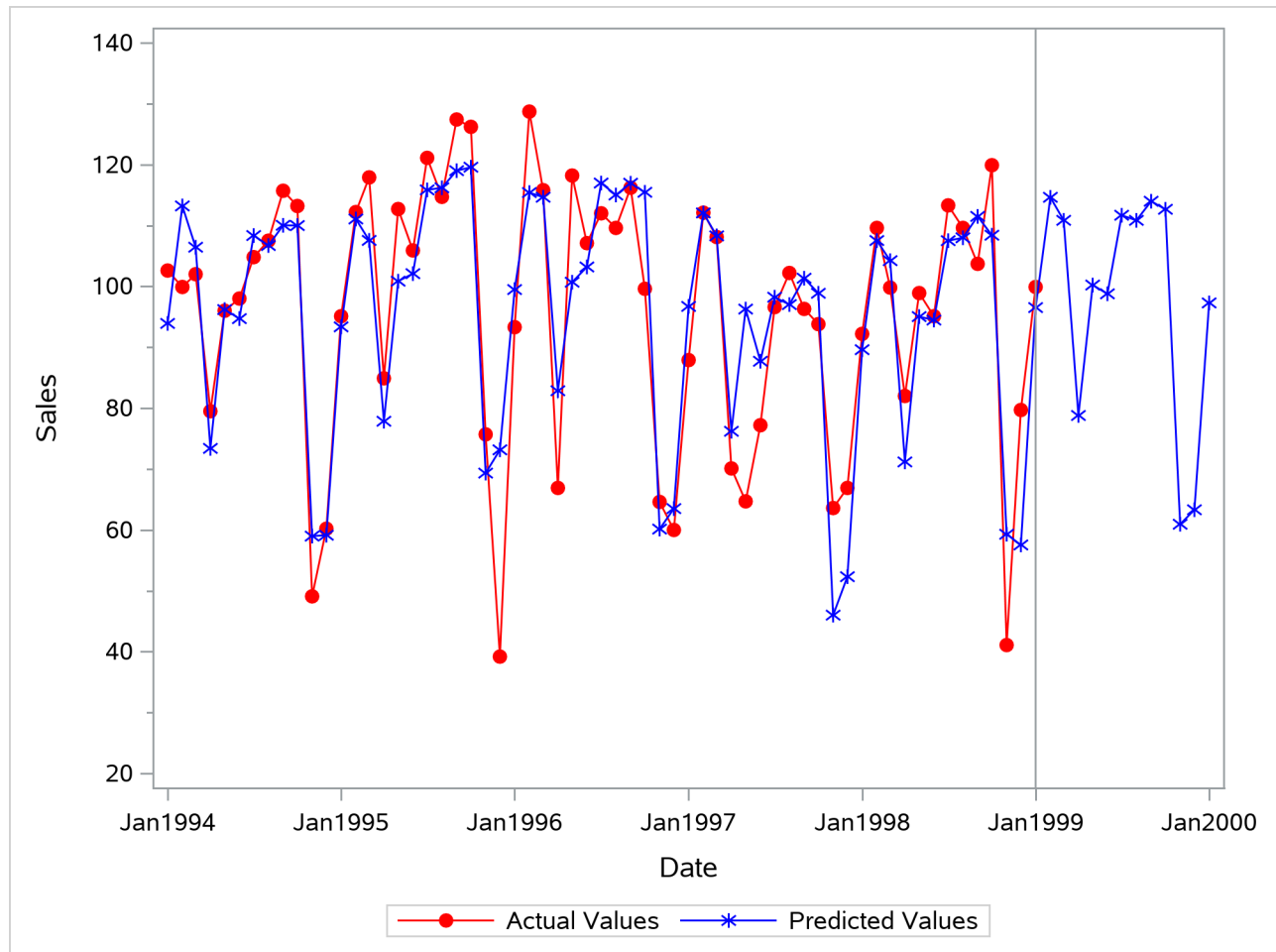
```
%plotActualPredict(mylib.nextyear, date, belts, '01JAN1999'd,
  '01JAN1994'd, '01DEC2000'd, year, 'Date',
  20, 140, 20, 'Sales');

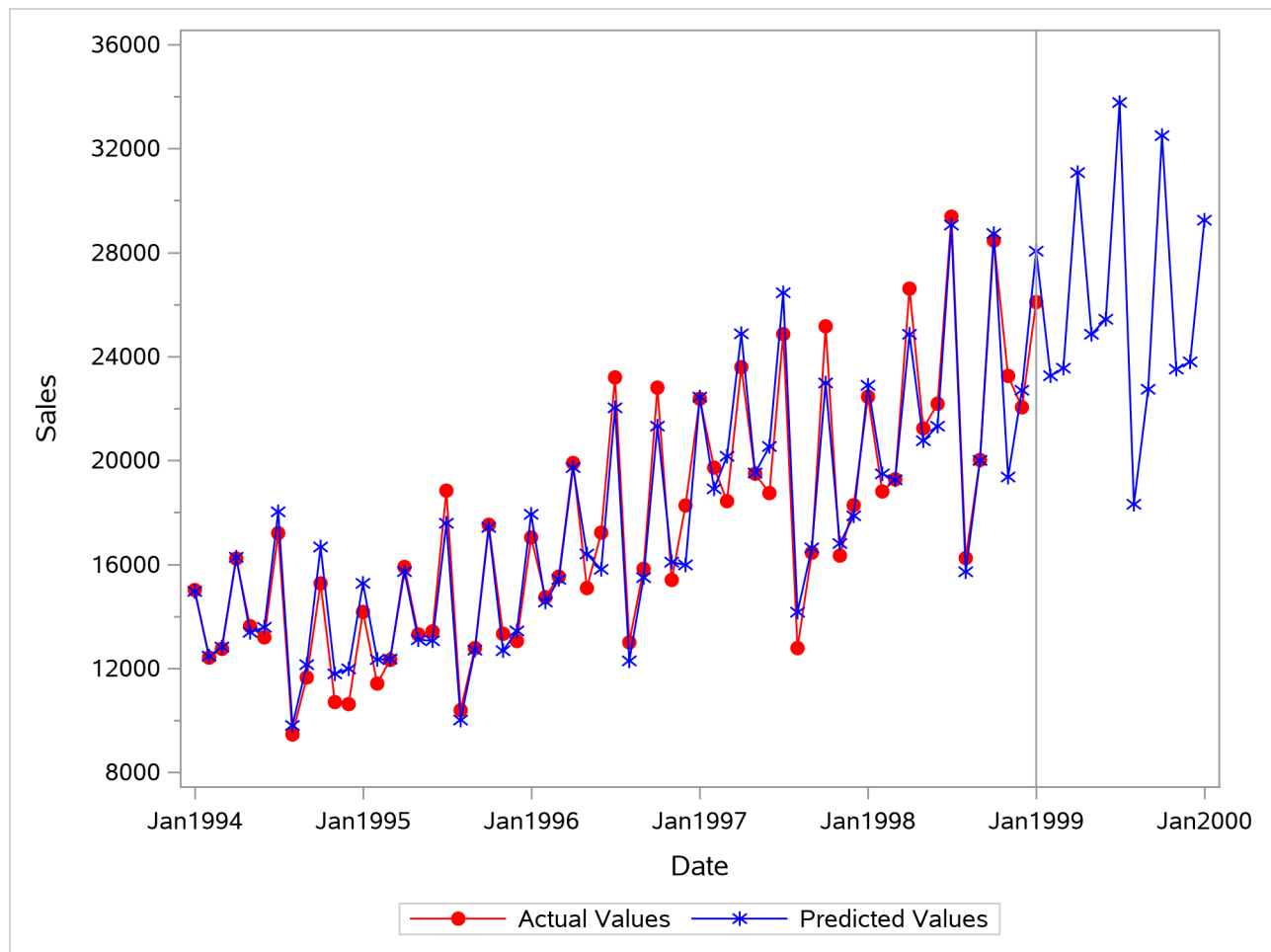
%plotActualPredict(mylib.nextyear, date, blouses, '01JAN1999'd,
  '01JAN1994'd, '01DEC2000'd, year, 'Date',
  8000, 36000, 4000, 'Sales');
```

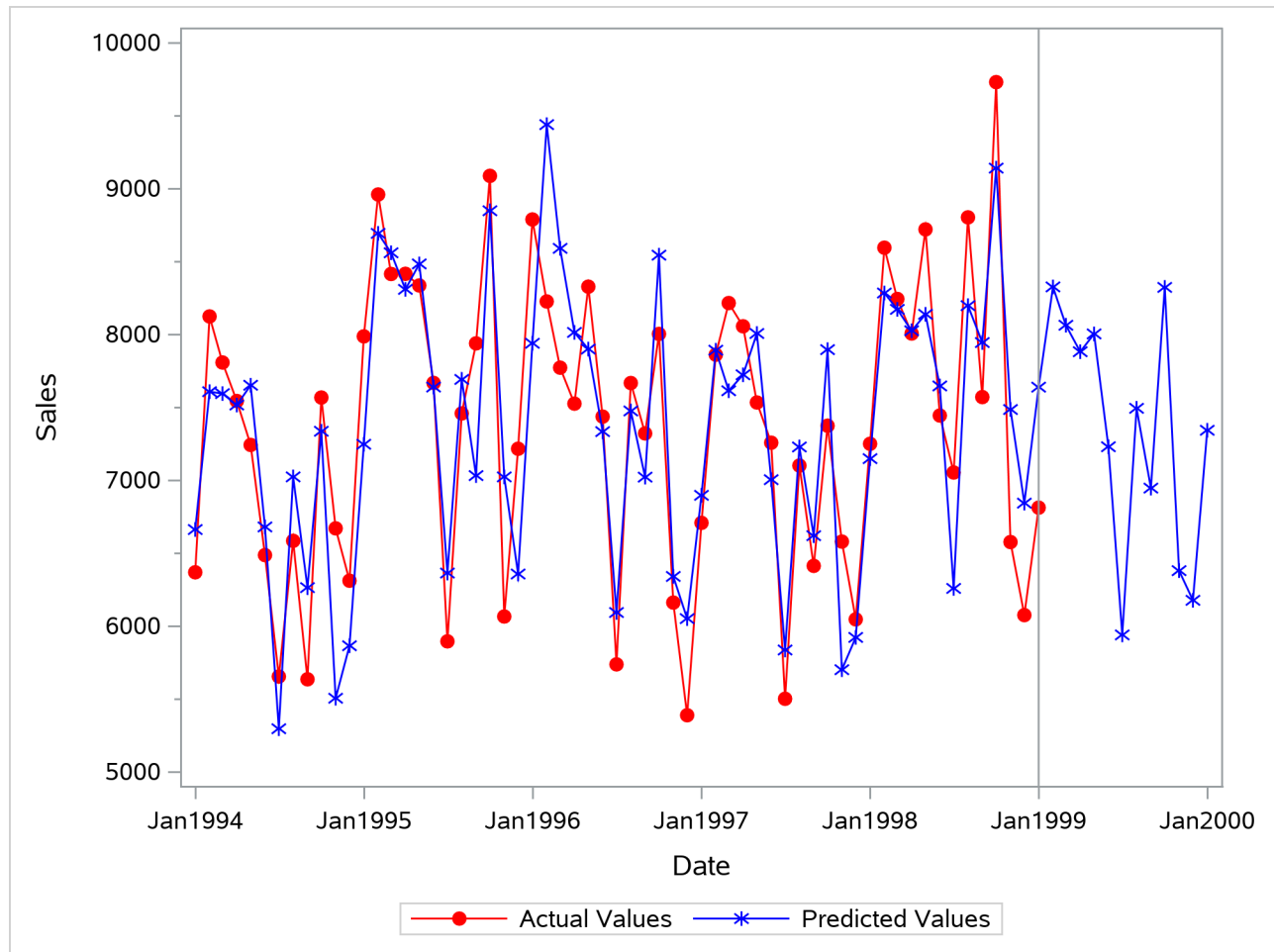
```
%plotActualPredict(mylib.nextyear, date, laces, '01JAN1999'd,
'01JAN1994'd, '01DEC2000'd, year, 'Date',
5000, 10000, 1000, 'Sales');
```

The plots are shown in [Output 9.1.5](#), [Output 9.1.6](#), and [Output 9.1.7](#). The fit of the predicted values to the actual values seems much better, so you might have more confidence in these forecasts.

Output 9.1.5 Retail Sales Forecast Plots (Belts)



Output 9.1.6 Retail Sales Forecast Plots (Blouses)

Output 9.1.7 Retail Sales Forecast Plots (Laces)

The following statements print the methods that are selected for each variable, as shown in [Output 9.1.8](#):

```
data salesEst; set mylib.salesEst; run;
proc sort data=salesEst nodupkey; by _name_ _compmodel_; run;
proc print data=salesEst label noobs;
  var _name_ _compmodel_;
  label _name_='Variable' _compmodel_='Method';
run;
```

Output 9.1.8 Best Method for Each Variable

Variable	Method
belts	SEASONAL
blouses	WINTERS
coats	ADDWINTERS
dressses	SEASONAL
hats	ADDWINTERS
laces	ADDWINTERS
shirts	ADDWINTERS
shoes	ADDWINTERS
socks	ADDWINTERS
ties	SEASONAL

Example 9.2: Forecasting of Transactional Data

This example illustrates how you can use the CESM procedure to forecast transactional data.

The statements in this example assume that your SAS library is named mylib, but you can substitute any appropriately defined SAS library.

The following DATA step creates a data table from data that were recorded at several internet websites. The data table mylib.websites contains a variable Time (which represents time) and the variables Engine, Boats, Cars, and Planes, which represent internet website data. Each value of the Time variable is recorded in ascending order, and the values of each of the other variables represent a transactional data series.

```
data mylib.websites(label="Transactional internet data");

    /*- time variable definition -*/
    keep time;
    format time datetime.;
    label time="Time of Web Hit";
    starttime = '12mar2000:00:00:00'dt; /*- Sunday -*/
    seedtime = 1234321;

    /*- cars variable definition -*/
    keep cars;
    format cars best12.;
    label cars="Number of Car Web Hits";
    seedcar = 1234321;

    /*- boats variable definition -*/
    keep boats;
    format boats best12.;
    label boats="Number of Boat Web Hits";
    seedboat = 1234321;

    /*- planes variable definition -*/
    keep planes;
    format planes best12.;
    label planes="Number of Planes Web Hits";
```



```

seedplane = 1234321;

/*- engines variable definition -*/
keep engines;
format engines best12.;
label engines="Number of Engine Web Hits";
seedengine = 1234321;

/*- simulate the data -*/
do day = 1 to 30;
    season = abs(4 - mod(day,7));
    nhits = ceil(10*ranuni(seedtime));
    intv = 24*3600*ranuni(seedtime)/nhits;
    do hits = 1 to nhits;

        /*- randomly generate the next time -*/
        intv = intv + 24*3600*ranuni(seedtime)/nhits;
        intv = int(intv);
        time = intnx( 'DTDAY', starttime, day );
        time = intnx( 'DTSECOND', time, intv );

        /*- randomly generate car data -*/
        cars = 1000 + 600*day + 1000*season
            + 10*rannor(seedcar);
        cars = int(cars);

        /*- randomly generate boats data -*/
        boats = 1000 + 1000*season +
            + 10*rannor(seedboat);
        boats = int(boats);

        /*- randomly generate planes data -*/
        planes = 1000 - 10*day +
            + 10*rannor(seedplane);
        planes = int(planes);

        /*- randomly generate engines data -*/
        engines = 1000 + 1*cars - 2*boats + 4*planes
            + 10*rannor(seedengine);
        engines = int(engines);

        output;
    end;
end;

run;

```

The following CESM procedure statements forecast each of the transactional data series:

```

proc cesm data=mylib.websites outfor=mylib.nextweek;
    id time interval=dtday accumulate=total;
    forecast boats cars planes / lead=7 method=simple;
run;

```

The preceding statements accumulate the data into a daily time series, generate forecasts for the Boats, Cars,

and Planes variables in the input data table mylib.websites for the next week, and store the forecasts in the mylib.nextWeek data table, which is specified in the OUTFOR= option.

The following statements plot the forecasts that are related to the internet data:

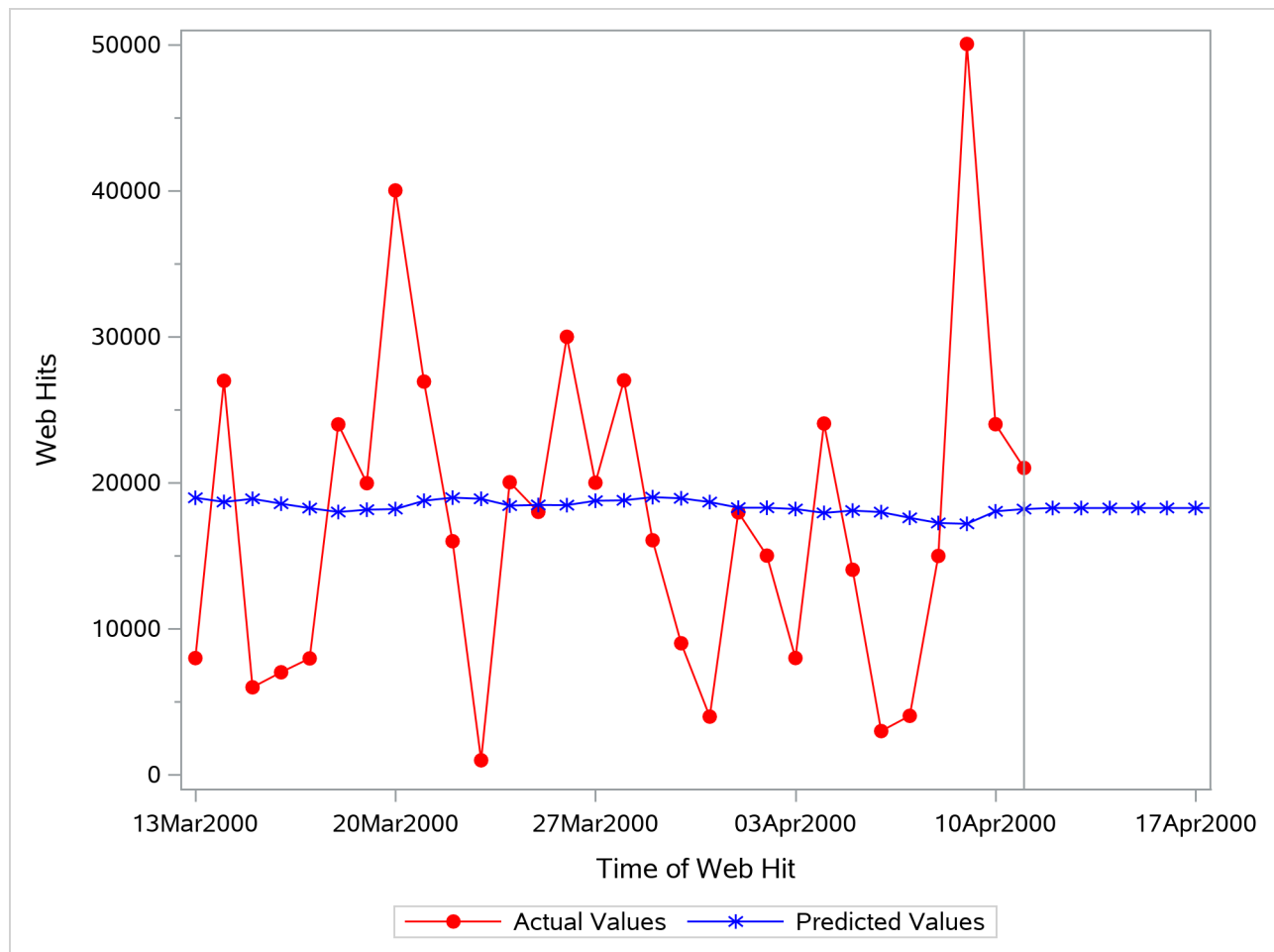
```
%plotActualPredict(mylib.nextweek, time, boats, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 50000, 10000, 'Web Hits');

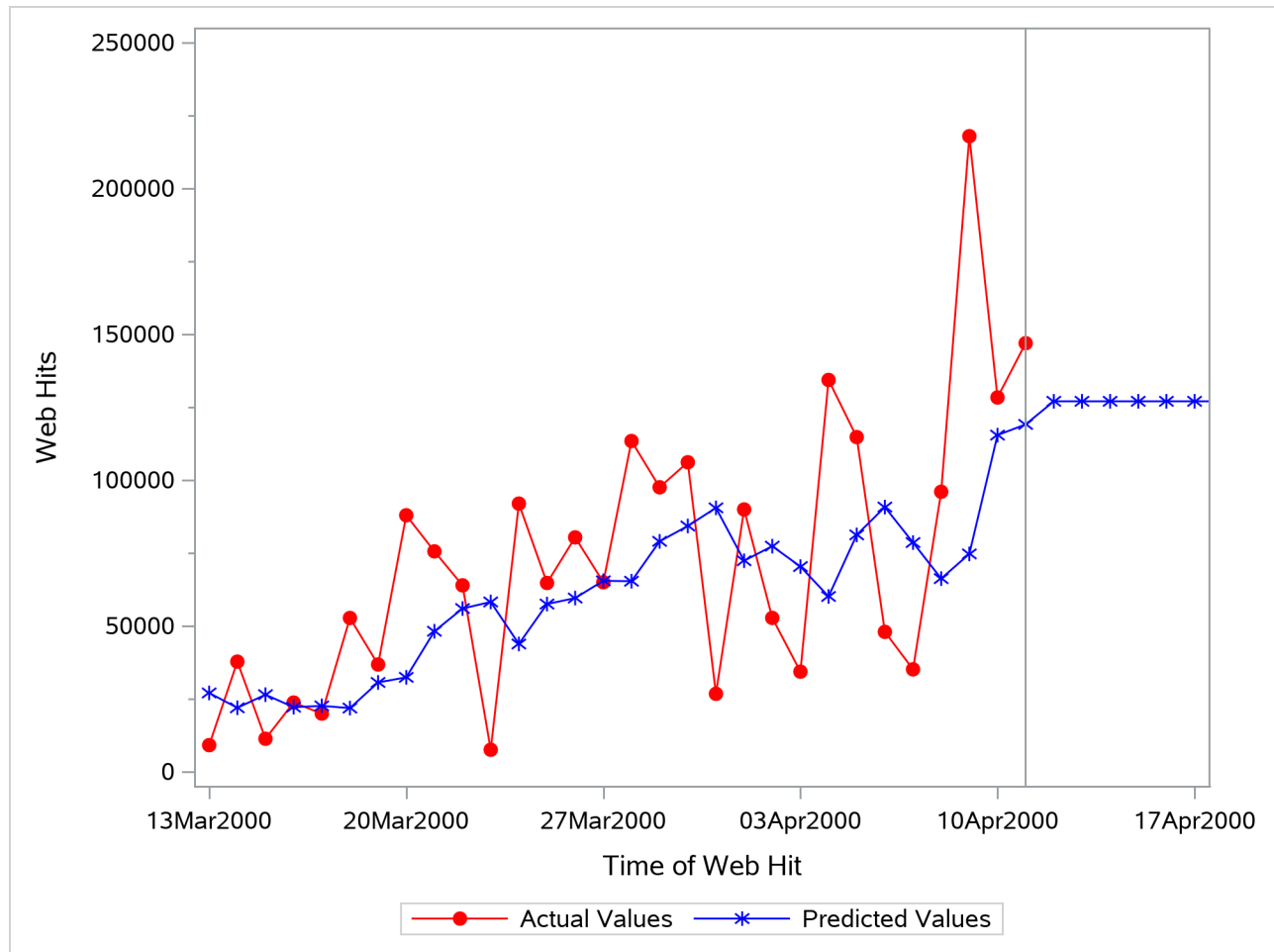
%plotActualPredict(mylib.nextweek, time, cars, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 250000, 50000, 'Web Hits');

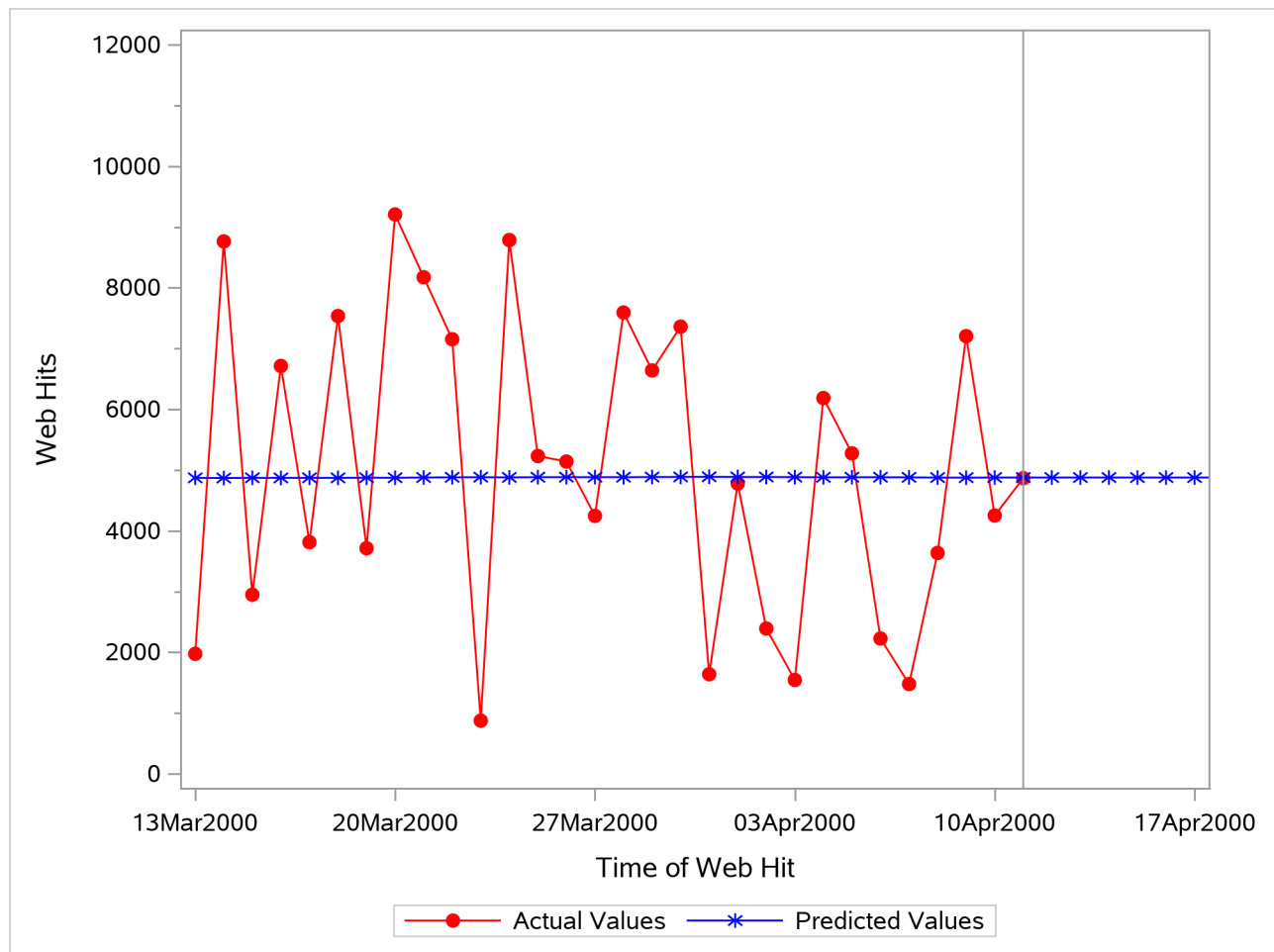
%plotActualPredict(mylib.nextweek, time, planes, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 12000, 2000, 'Web Hits');
```

The plots are shown in [Output 9.2.1](#), [Output 9.2.2](#), and [Output 9.2.3](#). The historical data and their fits are shown to the left of the reference line, and the forecasts for the next seven days are shown to the right.

Output 9.2.1 Internet Data Forecast Plots (Boats)



Output 9.2.2 Internet Data Forecast Plots (Cars)

Output 9.2.3 Internet Data Forecast Plots (Planes)

Example 9.3: Specifying the Forecasting Model

This example illustrates how you can use the CESM procedure to specify different models for different series. Internet data from the previous example are used for this illustration.

The statements in this example assume that your SAS library is named `mylib`, but you can substitute any appropriately defined SAS library.

This example forecasts the Boats variable by using the seasonal exponential smoothing model (SEASONAL), the Cars variable by using the Winters (multiplicative) model (WINTERS), and the Planes variable by using the log Winters (additive) model (ADDWINTERS). The following CESM procedure statements forecast each of the transactional data series based on these requirements:

```
proc cesm data=mylib.websites outfor=mylib.nextweek;
  id time interval=dtday accumulate=total;
  forecast boats / lead=7 method=seasonal;
  forecast cars / lead=7 method=winters;
  forecast planes / lead=7 method=addwinters transform=log;
run;
```

The following statements plot the forecasts that are related to the internet data:

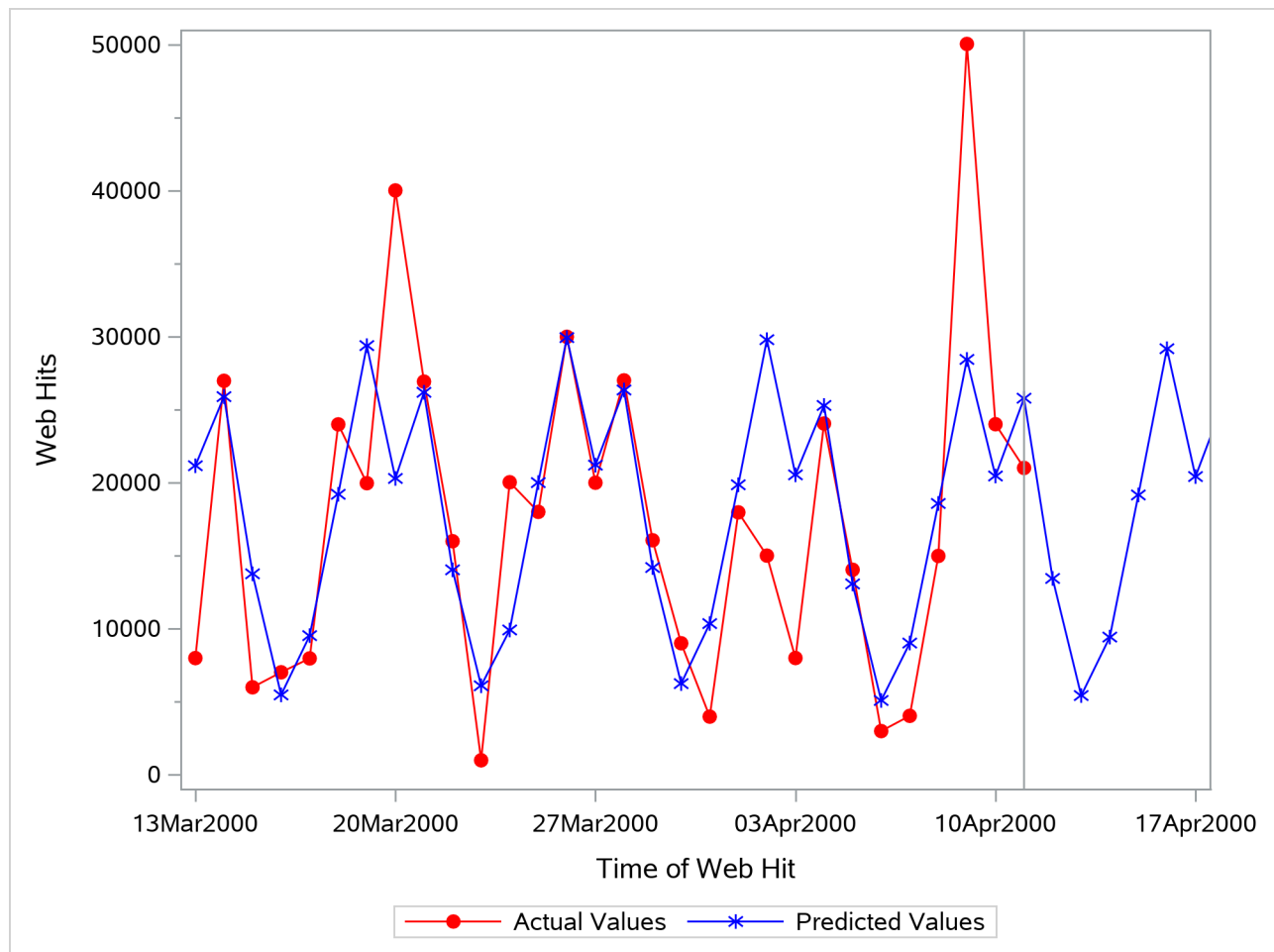
```
%plotActualPredict(mylib.nextweek, time, boats, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 50000, 10000, 'Web Hits');

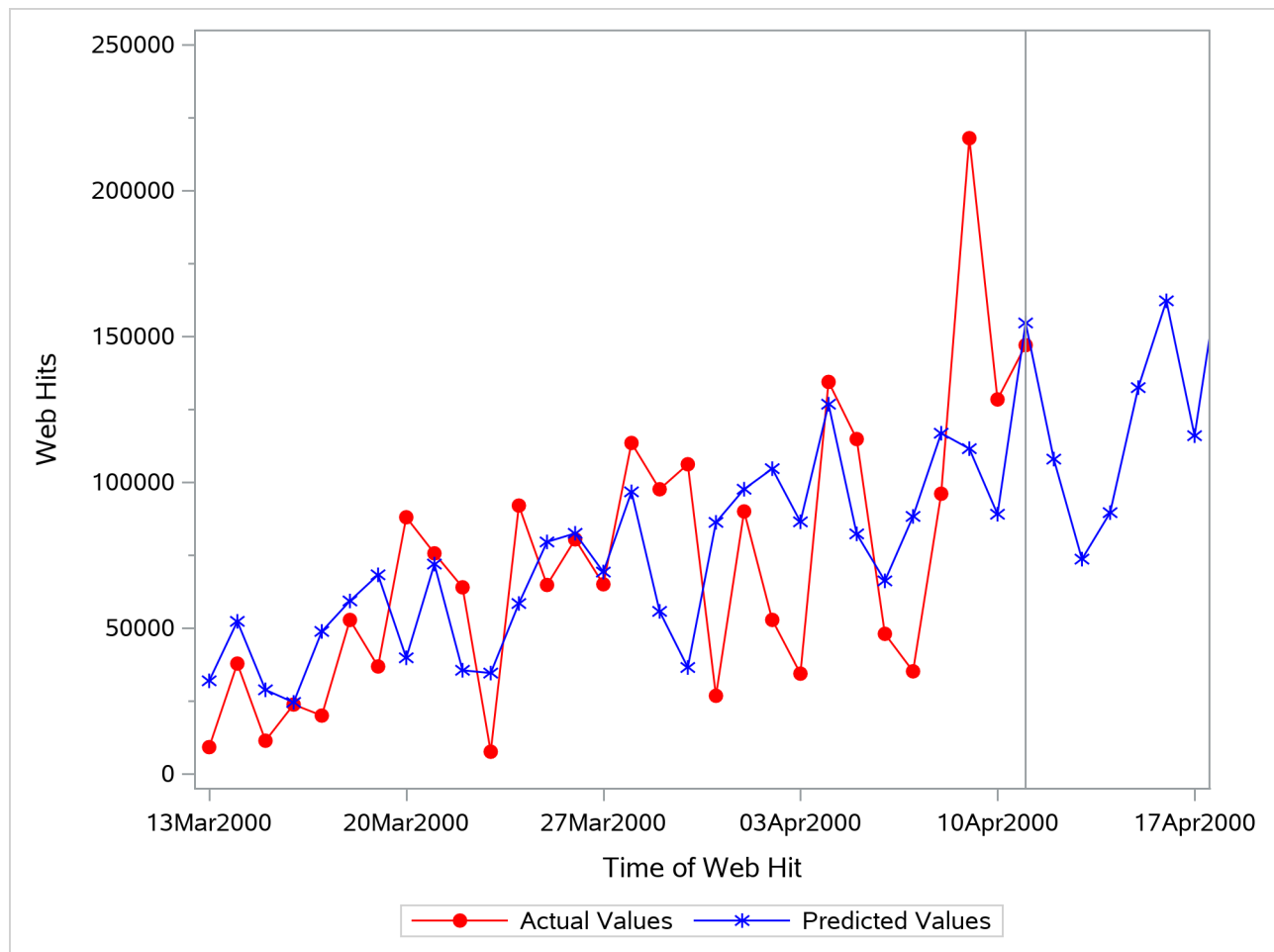
%plotActualPredict(mylib.nextweek, time, cars, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 250000, 50000, 'Web Hits');

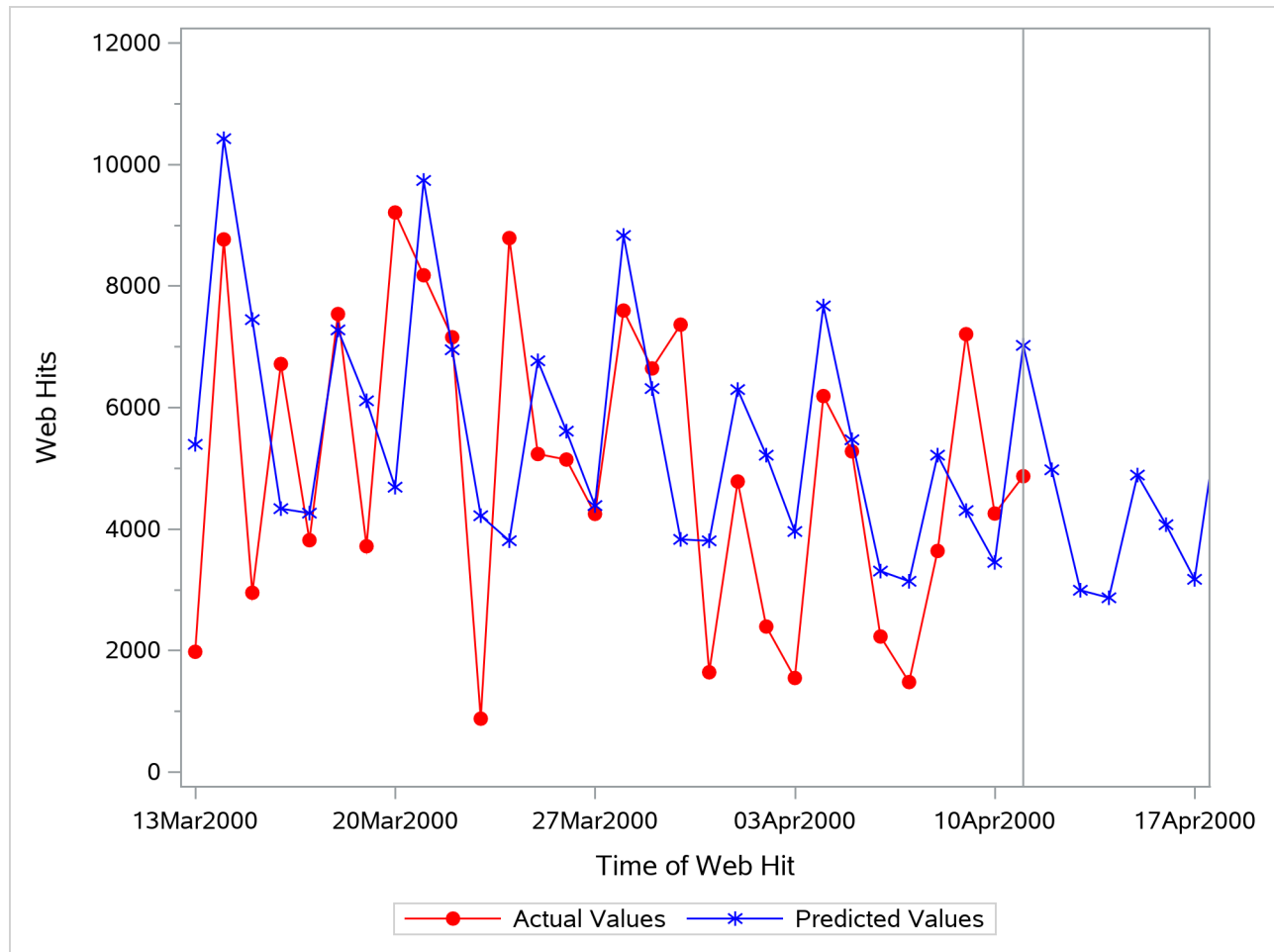
%plotActualPredict(mylib.nextweek, time, planes, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 12000, 2000, 'Web Hits');
```

The plots are shown in [Output 9.3.1](#), [Output 9.3.2](#), and [Output 9.3.3](#). The historical data and their fits are shown to the left of the reference line, and the forecasts for the next seven days are shown to the right.

Output 9.3.1 Internet Data Forecast Plots (Boats)



Output 9.3.2 Internet Data Forecast Plots (Cars)

Output 9.3.3 Internet Data Forecast Plots (Planes)

Chapter 10

CNTSELECT Procedure

Contents

Overview: CNTSELECT Procedure	376
PROC CNTSELECT Features	377
PROC CNTSELECT Compared with Other SAS Procedures	378
Using CAS Sessions and CAS Engine Librefs	378
Getting Started: CNTSELECT Procedure	379
Syntax: CNTSELECT Procedure	382
Functional Summary	382
PROC CNTSELECT Statement	386
BAYES Statement	388
BOUNDS Statement	397
BY Statement	398
CLASS Statement	398
DISPLAY Statement	398
DISPLAYOUT Statement	399
DISPMODEL Statement	400
EFFECT Statement	401
FREQ Statement	402
INIT Statement	403
MODEL Statement	403
OUTPUT Statement	405
PRIOR Statement	407
RESTRICT Statement	413
SELECTION Statement	415
TEST Statement	417
VIEWSTORE Statement	418
WEIGHT Statement	419
ZEROMODEL Statement	419
Details: CNTSELECT Procedure	420
Missing Values	420
Poisson Regression	421
Conway-Maxwell-Poisson Regression	422
Negative Binomial Regression	426
Zero-Inflated Count Regression Overview	428
Zero-Inflated Poisson Regression	429
Zero-Inflated Conway-Maxwell-Poisson Regression	430
Zero-Inflated Negative Binomial Regression	432

Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, INIT, and PRIOR Statements	433
Computational Resources	437
Covariance Matrix Types	437
Tests on Parameters	438
Bayesian Analysis	439
Prior Distributions	441
Displayed Output	443
OUTPUT OUT= Data Table	444
ODS Table Names	445
Examples: The CNTSELECT Procedure	447
Example 10.1: Zero-Inflated Poisson Model with CLASS Statement	447
Example 10.2: Zero-Inflated Poisson Model with BAYES and PRIOR Statements	450
Example 10.3: Bayesian Analysis Using NUTS with Multiple Chains	453
References	458

Overview: CNTSELECT Procedure

The CNTSELECT (count regression) procedure analyzes regression models in which the dependent variable takes nonnegative integer or count values. The dependent variable is usually an *event count*, which refers to the number of times an event occurs. For example, an event count might represent the number of ship accidents per year for a given fleet. In count regression, the conditional mean $E(y_i | \mathbf{x}_i)$ of the dependent variable y_i is assumed to be a function of a vector of covariates $\mathbf{x}_i$.

The models that PROC CNTSELECT supports can contain main effects that consist of both continuous and classification variables and interaction effects of these variables. The models can also include constructed effects such as splines. The procedure offers effect-selection methods along with capabilities for customizing the model selection process by using various selection and stopping criteria.

The Poisson (log-linear) regression model is the most basic model that explicitly takes into account the nonnegative integer-valued aspect of the outcome. For this model, the probability of an event count is determined by a Poisson distribution, where the conditional mean of the distribution is a function of a vector of covariates. However, the basic Poisson regression model is limited because it forces the conditional mean of the outcome to equal the conditional variance. This assumption is often violated in real-life data. Negative binomial regression is an extension of Poisson regression in which the conditional variance can exceed the conditional mean. Also, a common characteristic of count data is that the number of zeros in the sample exceeds the number of zeros that are predicted by either the Poisson or negative binomial model. Zero-inflated Poisson (ZIP) and zero-inflated negative binomial (ZINB) models explicitly model the production of zero counts to account for excess zeros and also enable the conditional variance of the outcome to differ from the conditional mean.

In zero-inflated models, additional zeros occur with probability φ_i , which is determined by a separate model, $\varphi_i = F(\mathbf{z}_i' \boldsymbol{\gamma})$, where F is the normal or logistic distribution function that results in a probit or logistic model and $\mathbf{z}_i$ is a set of covariates.

PROC CNTSELECT Features

The CNTSELECT procedure estimates the parameters of a count regression model by using maximum likelihood or Bayesian techniques.

The CNTSELECT procedure supports the following models for count data:

- Poisson regression
- Conway-Maxwell-Poisson regression
- negative binomial regression with quadratic and linear variance functions (Cameron and Trivedi 1986)
- zero-inflated Poisson (ZIP) model (Lambert 1992)
- zero-inflated Conway-Maxwell-Poisson (ZICMP) model
- zero-inflated negative binomial (ZINB) model
- fixed-effects and random-effects Poisson models for panel data
- fixed-effects and random-effects negative binomial models for panel data

The following list summarizes some basic features of the CNTSELECT procedure:

- has model-building syntax that uses **CLASS** and effect-based **MODEL** statements familiar from SAS/Econometrics analytic procedures
- performs maximum likelihood estimation
- supports multiple link functions
- uses the **WEIGHT** statement for weighted analysis
- uses the **FREQ** statement for grouped analysis
- uses the **OUTPUT** statement to produce a data table that contains predicted probabilities and other observationwise statistics
- uses the **BAYES** statement to perform Bayesian estimation for all models except Conway-Maxwell-Poisson models and panel models
- uses the **PRIOR** statement to set prior distributions for Bayesian estimation
- supports effect-selection methods

PROC CNTSELECT Compared with Other SAS Procedures

The CNTSELECT procedure provides count-data modeling functionality comparable to that of the HPCOUNTREG and COUNTREG procedures in SAS/ETS software.

PROC CNTSELECT Compared with the HPCOUNTREG Procedure

The functionality of the CNTSELECT procedure closely resembles that of the HPCOUNTREG procedure, which is a high-performance procedure. The CNTSELECT procedure is the next generation of the HPCOUNTREG procedure, and it was developed specifically for SAS Viya. Both procedures are capable of running on a cluster of machines that distribute the data and the computations. Both procedures perform computations in multiple threads on multiple machines.

The CNTSELECT procedure provides all the functionality of the HPCOUNTREG procedure and also provides Bayesian estimation techniques.

PROC CNTSELECT Compared with the COUNTREG Procedure

This release of the CNTSELECT procedure contains some, but not all, of the functionality present in the COUNTREG procedure. The COUNTREG procedure provides the following features, which are not present in the CNTSELECT procedure:

- spatial effect models
- scoring by using an item store

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: CNTSELECT Procedure

The CNTSELECT procedure is similar in use to other regression model procedures in the SAS System. For example, the following statements are used to estimate a Poisson regression model:

```
proc cntselect data=mylib.one;
  model y = x / dist=poisson;
run;
```

The response variable `y` is numeric and has nonnegative integer values.

This section illustrates two simple examples that use PROC CNTSELECT. The data are taken from Long (1997). This study examines how factors such as gender (`fem`), marital status (`mar`), number of young children (`kid5`), prestige of the graduate program (`phd`), and number of articles published by a scientist’s mentor (`ment`) affect the number of articles (`art`) published by the scientist.

The first 10 observations are shown in [Figure 10.1](#).

Figure 10.1 Article Count Data

Obs	art	fem	mar	kid5	phd	ment
1	3	0	1	2	1.38000	8.0000
2	1	0	1	1	3.59000	19.0000
3	0	0	1	1	2.12000	10.0000
4	3	0	1	1	1.80000	4.0000
5	1	0	1	1	3.41000	10.0000
6	2	0	1	0	2.10000	2.0000
7	0	0	1	1	2.26000	5.0000
8	6	0	1	1	3.85000	16.0000
9	2	0	0	0	2.26000	2.0000
10	1	0	1	2	4.29000	10.0000

The following SAS statements estimate the Poisson regression model.

```

/*-- Poisson Regression --*/
proc cntselect data=mylib.long97data;
  model art = fem mar kid5 phd ment / dist=poisson method=quanew;
run;

```

The “Model Fit Summary” table that is shown in [Figure 10.2](#) lists several details about the model. By default, the CNTSELECT procedure uses the Newton-Raphson optimization technique. The maximum log-likelihood value is shown, in addition to two information measures—Akaike’s information criterion (AIC) and Schwarz’s Bayesian information criterion (SBC)—which can be used to compare competing Poisson models. Smaller values of these criteria indicate better models.

Figure 10.2 Estimation Summary Table for a Poisson Regression

The CNTSELECT Procedure

Model Fit Summary	
Dependent Variable	art
Number of Observations	915
Data Set	LONG97DATA
Model	Poisson
Log Likelihood	-1651.06
Maximum Absolute Gradient	0.001295
Number of Iterations	11
Optimization Method	Quasi-Newton
AIC	3314.113
SBC	3343.026
Covariance Estimation	Hessian

[Figure 10.3](#) shows the parameter estimates of the model and their standard errors. All covariates are significant predictors of the number of articles, except for the prestige of the program (phd), which has a *p*-value of 0.6271.

Figure 10.3 Parameter Estimates of Poisson Regression

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	0.304617	0.102982	2.96	0.0031
fem	1	-0.224595	0.054614	-4.11	<.0001
mar	1	0.155243	0.061375	2.53	0.0114
kid5	1	-0.184882	0.040127	-4.61	<.0001
phd	1	0.012823	0.026397	0.49	0.6271
ment	1	0.025543	0.002006	12.73	<.0001

To allow for variance greater than the mean, you can fit the negative binomial model instead of the Poisson model by specifying the DIST=NEGBIN option, as shown in the following statements. Whereas the Poisson model requires that the conditional mean and conditional variance be equal, the negative binomial model allows for overdispersion, in which the conditional variance can exceed the conditional mean.

```

/*-- Negative Binomial Regression --*/
proc cntselect data=mylib.long97data;
  model art = fem mar kid5 phd ment / dist=negbin(p=2) method=quanew;
run;

```

Figure 10.4 shows the fit summary and Figure 10.5 shows the parameter estimates.

Figure 10.4 Estimation Summary Table for a Negative Binomial Regression

The CNTSELECT Procedure

Model Fit Summary	
Dependent Variable	art
Number of Observations	915
Data Set	LONG97DATA
Model	NegBin(p=2)
Log Likelihood	-1560.96
Maximum Absolute Gradient	0.070336
Number of Iterations	12
Optimization Method	Quasi-Newton
AIC	3135.917
SBC	3169.649
Covariance Estimation	Hessian

Figure 10.5 Parameter Estimates of Negative Binomial Regression

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	0.256154	0.138560	1.85	0.0645
fem	1	-0.216404	0.072672	-2.98	0.0029
mar	1	0.150485	0.082106	1.83	0.0668
kid5	1	-0.176423	0.053060	-3.32	0.0009
phd	1	0.015266	0.036040	0.42	0.6719
ment	1	0.029083	0.003470	8.38	<.0001
_Alpha	1	0.441618	0.052967	8.34	<.0001

The parameter estimate for _Alpha of 0.4416 is an estimate of the dispersion parameter in the negative binomial distribution. A t test for the hypothesis $H_0 : \alpha = 0$ is provided. It is highly significant, indicating overdispersion ($p < 0.0001$).

The null hypothesis $H_0 : \alpha = 0$ can be also tested against the alternative $\alpha > 0$ by using the likelihood ratio test, as described by Cameron and Trivedi (1998, pp. 45, 77–78). The likelihood ratio test statistic is equal to $-2(\mathcal{L}_P - \mathcal{L}_{NB}) = -2(-1651 + 1561) = 180$, which is highly significant, providing strong evidence of overdispersion.

Syntax: CNTSELECT Procedure

The following statements are available in the CNTSELECT procedure. Items within angle brackets (< >) or square brackets ([]) are optional.

```

PROC CNTSELECT <options> ;
  BAYES <options> ;
  BOUNDS bound1 [ , bound2 ... ] ;
  BY variables ;
  CLASS variable <(options)> ... <variable <(options)> > </ global-options> ;
  DISPLAY <table-list> </ options> ;
  DISPLAYOUT table-spec-list </ options> ;
  DISPMODEL dependent variable ~ <dispersion-related regressors> ;
  EFFECT name=effect-type(variables </ options>) ;
  FREQ freq-variable ;
  INIT initialization1 < , initialization2 ... > ;
  MODEL dependent-variable = regressors </ options> ;
  OUTPUT <output-options> ;
  PRIOR parameter-list ~ distribution ;
  RESTRICT restriction1 [ , restriction2 ... ] ;
  SELECTION < METHOD=method <(method-options)> > <options> ;
  TEST equation1 < , equation2 ... > </ test-options> ;
  VIEWSTORE / options ;
  WEIGHT variable </ option> ;
  ZEROMODEL dependent-variable ~ zero-inflated-regressors </ options> ;

```

There can be only one MODEL statement. Any ZEROMODEL, DISPMODEL, OUTPUT, INIT, BOUNDS, RESTRICT, or TEST statement, if used, must appear after the MODEL statement. There can be no more than one ZEROMODEL statement and no more than one DISPMODEL statement. Multiple INIT, BOUNDS, RESTRICT, and TEST statements are allowed. The BAYES statement is not compatible with the BOUNDS, DISPMODEL, RESTRICT, SELECTION, or TEST statement. If a FREQ or WEIGHT statement is specified more than once, the variable specified in the first instance is used.

Functional Summary

Table 10.1 summarizes the statements and options used with the CNTSELECT procedure.

Table 10.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CNTSELECT	DATA=
Specifies the identification variable for panel data analysis	PROC CNTSELECT	GROUPID=
Writes estimates to an output data table	OUTPUT	OUT=
Specifies BY-group processing	BY	

Table 10.1 *continued*

Description	Statement	Option
Specifies the classification variables	CLASS	
Specifies an optional frequency variable	FREQ	
Specifies an optional weight variable	WEIGHT	
Printing Control Options		
Prints the correlation matrix of the estimates	PROC CNTSELECT	CORRB
Prints the covariance matrix of the estimates	PROC CNTSELECT	COVB
Prints the parameter labels without truncating	PROC CNTSELECT	EXTENDPARMLENGTH
Suppresses the normal printed output	PROC CNTSELECT	NOPRINT
Requests all printing options	PROC CNTSELECT	PRINTALL
Prints timing information	PROC CNTSELECT	PRINTTIMING
Prints the names used internally for the parameters	PROC CNTSELECT	PRINTINTERNALNAMES
Options to Control the Optimization Process		
Selects the iterative minimization method to use	PROC CNTSELECT	METHOD=
Specifies maximum number of iterations allowed	PROC CNTSELECT	MAXITER=
Specifies maximum number of function calls	PROC CNTSELECT	MAXFUNC=
Specifies the upper limit of CPU time in seconds	PROC CNTSELECT	MAXTIME=
Sets boundary restrictions on parameters	BOUNDS	
Sets initial values for parameters	INIT	
Sets linear restrictions on parameters	RESTRICT	
Model Estimation Options		
Specifies the dispersion variables	DISPMODEL	
Specifies the type of model	PROC CNTSELECT	DIST=
Specifies the type of covariance matrix	PROC CNTSELECT	COVEST=
Specifies the type of error components model for panel data	MODEL	ERRORCOMP=
Suppresses the intercept parameter	MODEL	NOINT
Specifies the offset variable	MODEL	OFFSET=
Specifies the parameterization for the Conway-Maxwell-Poisson (CMP) model	MODEL	PARAMETER=
Specifies the zero-inflated offset variable	ZEROMODEL	OFFSET=
Specifies the zero-inflated link function	ZEROMODEL	LINK=
Specifies the constructed regression effects	EFFECT	
Regression Effect-Selection Options		
Specifies the selection method	SELECTION	METHOD=
Specifies how to apply the STOP= criterion	SELECTION	STOPHORIZON=
Regression Effect-Selection Method Options		
Specifies a criterion for choosing the best model at each step	SELECTION	CHOOSE=

Table 10.1 *continued*

Description	Statement	Option
Specifies that the competitive form of the stepwise selection method be used	SELECTION	COMPETITIVE
Specifies the maximum number of effects in the model	SELECTION	MAXEFFECTS=
Specifies the maximum number of selection steps	SELECTION	MAXSTEPS=
Specifies the minimum number of effects in the model	SELECTION	MINEFFECTS=
Specifies a criterion to decide which effects enter or leave the model at each step	SELECTION	SELECT=
Specifies a criterion for stopping the selection process	SELECTION	STOP=
Bayesian Markov Chain Monte Carlo (MCMC) Options		
Specifies the number of burn-in iterations	BAYES	NBURNIN=
Specifies the number of Markov chains to run during the sampling phase	BAYES	NCHAIN=
Specifies the number of iterations to run during the sampling phase	BAYES	NSAMPLE=
Specifies the MCMC sampler and its options	BAYES	SAMPLER=
Specifies the random number generator seed	BAYES	SEED=
Controls the thinning of the Markov chain	BAYES	THIN=
Bayesian Output Tables		
Displays MCMC diagnostics	BAYES	DIAGNOSTICS=
Displays MCMC diagnostics for each Markov chain	BAYES	DIAGNOSTICS(BYCHAIN)=
Displays the settings used for all MCMC diagnostics	BAYES	DIAGNOSTICS(SETTINGS)=
Displays a summary of all parameters and their priors	BAYES	PRIORSUMMARY
Displays the settings used for the MCMC sampler	BAYES	SAMPLERSETTINGS
Displays a summary of the MCMC sampler used	BAYES	SAMPLERSUMMARY
Displays posterior summary statistics	BAYES	STATISTICS=
Displays posterior summary statistics for each Markov chain	BAYES	STATISTICS(BYCHAIN)=
Bayesian Posterior Sample		
Specifies options for saving a SAS data table for the posterior sample	BAYES	OUTPOST=
Bayesian Prior Options		
Specifies the Cauchy prior distribution	PRIOR	CAUCHY()
Specifies the gamma prior distribution	PRIOR	GAMMA()
Specifies the inverse gamma prior distribution	PRIOR	IGAMMA()

Table 10.1 *continued*

Description	Statement	Option
Specifies the normal prior distribution	PRIOR	NORMAL()
Specifies the square root gamma prior distribution	PRIOR	SQRTGAMMA()
Specifies the square root inverse gamma prior distribution	PRIOR	SQRTIGAMMA()
Specifies the t prior distribution	PRIOR	T()
Specifies the uniform prior distribution	PRIOR	UNIFORM()
Output Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	
Specifies the output item store to preserve the properties of the model and the results of fitting the model	PROC CNTSELECT	STORE=
Outputs SAS variables to the output data table	OUTPUT	COPYVAR=
Outputs the estimates of dispersion for the CMP model	OUTPUT	DISPERSION
Outputs the estimates of $G\Delta = \mathbf{g}_i' \boldsymbol{\delta}$ for the CMP model	OUTPUT	GDELTA=
Outputs the estimates of λ for the CMP model	OUTPUT	LAMBDA=
Outputs the estimates of ν for the CMP model	OUTPUT	NU=
Outputs the estimates of μ for the CMP model	OUTPUT	MU=
Outputs the estimates of mode for the CMP model	OUTPUT	MODE=
Outputs the probability that the response variable will take the current value	OUTPUT	PROB=
Outputs probabilities for particular response values	OUTPUT	PROBCOUNT()
Outputs expected value of response variable	OUTPUT	PRED=
Outputs the estimates of variance for the CMP model	OUTPUT	VARIANCE=
Outputs estimates of $\mathbf{XBeta} = \mathbf{x}_i' \boldsymbol{\beta}$	OUTPUT	XBETA=
Outputs estimates of $\mathbf{ZGamma} = \mathbf{z}_i' \boldsymbol{\gamma}$	OUTPUT	ZGAMMA=
Outputs probability of a zero value as a result of the zero-generating process	OUTPUT	PROBZERO=

PROC CNTSELECT Statement

PROC CNTSELECT <options> ;

The following *options* can be used in the PROC CNTSELECT statement.

Input Data Table Options

DATA=*libref.data-table*

names the input data table for PROC CNTSELECT to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 378.

data-table specifies the name of the input data table.

GROUPID=*variable*

specifies an identification variable when a panel data model is estimated. The identification variable is used as a cross-sectional ID variable.

Item Store Option

STORE=*SAS-item-store*

specifies the item store that preserves the properties of the model and the fit results. No information about the model is preserved unless you specify this option.

Printing Options

You can specify the following options in either the PROC CNTSELECT statement or the MODEL statement:

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

EXTENDPARMLENGTH

prints the parameter labels without truncating.

NOPRINT

suppresses all printed output.

PRINTALL

requests all printing options.

PRINTTIMING

prints a timing report.

PRINTINTERNALNAMES

prints internal names assigned to parameters.

Estimation Control Options

You can specify the following options in either the PROC CNTSELECT statement or the MODEL statement:

COVEST=HESSIAN | OP | QML

specifies the type of covariance matrix for the parameter estimates.

The default is COVEST=HESSIAN. You can specify the following values:

HESSIAN	specifies the covariance from the Hessian matrix.
OP	specifies the covariance from the outer product matrix.
QML	specifies the covariance from the outer product and Hessian matrices.

Optimization Control Options

PROC CNTSELECT uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. You can specify the following *options* in either the PROC CNTSELECT statement or the MODEL statement.

MAXFUNC=*i***MAXFU=*i***

specifies the maximum number of function calls in the optimization process. The default is 1,000.

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually performed can exceed the number of calls that are specified by this option.

MAXITER=*i***MAXIT=*i***

specifies the maximum number of iterations in the optimization process. The default is 200.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. By default, MAXTIME=200. The time that you specify in this option is checked only once at the end of each iteration. Therefore, the actual run time can be much longer than *r*. The actual run time includes the remaining time that is needed to finish the iteration and the time that is needed to generate the output of the results.

METHOD=*value*

specifies the iterative minimization method to use. The default is METHOD=NEWRAP. You can specify the following *values*:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double-dogleg method.
NEWRAP	specifies the Newton-Raphson method (this is the default).
NONE	specifies that no optimization be performed beyond using the ordinary least squares method to compute the parameter estimates.
NRRIDG	specifies the Newton-Raphson Ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

BAYES Statement

BAYES < *options* > ;

The BAYES statement controls the Markov chain Monte Carlo sampling scheme that is used to obtain samples from the posterior distribution of the underlying model and data. The BAYES statement is incompatible with the BOUNDS, DISPMODEL, RESTRICT, SELECTION, and TEST statements and with Conway-Maxwell-Poisson and panel models. You can specify the following *options*:

DIAGNOSTICS < (*diag-options*) >=ALL | NONE | *keyword-list* | (*keyword-list*)

DIAG < (*diag-options*) >=ALL | NONE | *keyword-list* | (*keyword-list*)

specifies which diagnostics to produce. If the value of the NCHAIN= option is greater than 1, then instead of the diagnostic tables being produced for each Markov chain, by default the results of the diagnostics for all the chains are appropriately combined and summarized. For more information, see the section “[Using Multiple Chains](#)” on page 37 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or both of the following *diag-options*:

BYCHAIN

produces diagnostic tables for each Markov chain. If NCHAIN=1, then the BYCHAIN option does nothing.

SETTINGS

produces a table of all MCMC diagnostic settings.

If you specify both the BYCHAIN and SETTINGS options, separate them by using a space or a comma. To produce all diagnostics, specify DIAGNOSTICS=ALL. To produce no diagnostics, specify DIAGNOSTICS=NONE. To produce some but not all diagnostics, or to change certain settings of these diagnostics, specify one or more of the following keywords. If you specify multiple keywords, separate them by using spaces.

AUTOCORR <(LAGS=*numeric-list*)>

AC <(LAGS=*numeric-list*)>

computes the autocorrelations at lags that you specify in the *numeric-list*. For more information, see the section “[Autocorrelations](#)” on page 34 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” Elements in the *numeric-list* must be positive integers, and values can be separated either by spaces or by commas. Repeated values are removed. If you omit the LAGS= option, autocorrelations of lags 1, 5, 10, and 50 are computed.

GEWEKE <(geweke-options)>

GEW <(geweke-options)>

computes the Geweke spectral density diagnostics. For more information, see the section “[Geweke Diagnostics](#)” on page 28 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *geweke-options*:

ALPHA=*value*

specifies the α level ($0 < \alpha < 1$) for the Geweke test. By default, ALPHA=0.05.

FRAC1=*value*

F1=*value*

specifies the fraction f_1 for the first window. By default, FRAC1=0.1.

FRAC2=*value*

F2=*value*

specifies the fraction f_2 for the second window. By default, FRAC2=0.5.

HEIDELBERGER <(heidel-options)>

HEIDEL <(heidel-options)>

HB <(heidel-options)>

computes the Heidelberg-Welch diagnostic for each variable. For more information, see the section “[Heidelberg-Welch Diagnostics](#)” on page 30 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or more of the following *heidel-options*:

EPSILON=*value*

EPS=*value*

specifies a positive number ϵ such that if the halfwidth is less than ϵ times the sample mean of the retained iterates, the halfwidth test is passed. By default, EPSILON=0.1.

HALFWIDTH_ALPHA=*value*

HALPHA=*value*

specifies the α level ($0 < \alpha < 1$) for the halfwidth test. By default, HALFWIDTH_ALPHA=0.05.

STATIONARY_ALPHA=*value*

SALPHA=*value*

specifies the α level ($0 < \alpha < 1$) for the stationarity test. By default, STATIONARY_ALPHA=0.05.

RAFTERY <(raftery-options)>

RL <(raftery-options)>

computes the Raftery-Lewis diagnostics. For more information, see the section “[Raftery-Lewis Diagnostics](#)” on page 31 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or more of the following *raftery-options*:

ACCURACY=*value*

ACC=*value*

R=*value*

specifies a small positive number *r* as the margin of error for measuring the accuracy of estimation of the quantile. By default, ACCURACY=0.005.

EPSILON=*value*

EPS=*value*

specifies the tolerance level (ϵ , a small positive number) for the stationary test. By default, EPSILON=0.001.

PROBABILITY=*value*

PROB=*value*

S=*value*

specifies the probability *s* of attaining the accuracy of the estimation of the quantile. By default, PROBABILITY=0.95.

QUANTILE=*value*

QUANT=*value*

Q=*value*

specifies the order (*q*, a value between 0 and 1) of the quantile of interest. By default, QUANTILE=0.025.

SUMMARIES <(MAXLAG=*value*)>

SUMMARY <(MAXLAG=*value*)>

SUM <(MAXLAG=*value*)>

computes a summary of MCMC diagnostics for each parameter in the Markov chain, including effective sample size (ESS), Monte Carlo standard error (MCSE), and related diagnostics. For more information about ESS, see the section “[Effective Sample Size](#)” on page 34 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#)”; for more information about MCSE, see the section “[Standard Error of the Mean Estimate](#)” on page 36 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” MAXLAG=*value* specifies the maximum autocorrelation lag to use for computing ESS, which is used to compute MCSE, where *value* must be a positive integer. By default, MAXLAG=250.

By default, DIAGNOSTICS=SUMMARIES.

DISTRIBUTED**DISTRIBUTE**

keeps the data in the input table distributed on the CAS server when the server runs in massively parallel processing (MPP) mode. If you omit this option, instead the data are copied onto each worker for Bayesian inference, and multiple Markov chains are run in parallel. This option is ignored when the CAS server runs in symmetric multiprocessing (SMP) mode. For more information, see the section “[Distributed Data and MCMC](#)” on page 440.

NBURNIN=*number*

BURNIN=*number*

NBI=*number*

specifies the number of burn-in iterations to perform in each Markov chain before the chains are saved. For more information, see the section “[Burn-In, Thinning, and Markov Chain Samples](#)” on page 27 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, NBURNIN=0.

NCHAIN=*number*

NCHAINS=*number*

CHAINS=*number*

specifies the number of Markov chains to run. By default, NCHAIN=1.

NSAMPLE=*number*

NSAM=*number*

NMC=*number*

specifies the number of iterations after tuning and burn-in to save in each Markov chain. For more information, see the section “[Burn-In, Thinning, and Markov Chain Samples](#)” on page 27 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, NSAMPLE=1000.

OUTPOST<(*outpost-options*)>=*libref.data-table*

names the SAS data table to contain the posterior sample. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 378. You can specify the following *outpost-options*:

DETAILED

saves any relevant additional variables, such as auxiliary variables, in the SAS data table. The random walk Metropolis (RWM) sampler without the [TRANSFORM](#) option has no additional variables to save. With the TRANSFORM option, the SAS data table also includes the transformed version of each parameter and the log posterior of the transformed parameters. When you use the No-U-Turn Sampler (NUTS) by specifying the [SAMPLER=NUTS](#) option, the SAS data table includes those additional variables, as well as the tree height that is achieved at every iteration.

WARMUP

saves warmup (tuning and burn-in) iterations in the SAS data table. This option is not available for the random walk Metropolis (RWM) sampler.

PRIORSUMMARY <(priorsum-option)>

PRIORSUM <(priorsum-option)>

produces a summary of all parameters in the model and their prior distributions. You can specify the following *priorsum-option*:

SHOWNAMES

NAMES

adds a column that displays the names of all parameters in the model. These are the names that you use to specify prior distributions on those parameters.

SAMPLER=NUTS <(nuts-options)> | **RWM** <(rwm-options)>

SAMPLING=NUTS <(nuts-options)> | **RWM** <(rwm-options)>

specifies how to sample from the posterior distribution. By default, SAMPLER=RWM.

NUTS <(nuts-options)>

samples from the posterior distribution by using the No-U-Turn Sampler. You can specify the following *nuts-options*:

DELTA=number

TARGETRATE=number

RATE=number

specifies the target Metropolis acceptance rate δ for the No-U-Turn Sampler. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. By default, DELTA=0.8.

MASSTUNE=NONE | **WINDOWED** <(windowed-options)>

specifies the mass matrix tuning method for the No-U-Turn Sampler. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. By default, MASSTUNE=WINDOWED. You can specify the following values:

NONE

specifies no mass matrix tuning. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440.

WINDOWED <(windowed-options)>

WINDOW <(windowed-options)>

WIN <(windowed-options)>

specifies the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. You can specify the following *windowed-options*:

EXPANDRATE=number

EXPRATE=number

RATE=number

specifies the window expansion rate γ for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. By default, EXPANDRATE=2.

FINALWINDOW=number

FINAL=number

specifies the size of the final tuning window n_K for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. By default, FINALWINDOW=50.

INITWINDOW=number

INIT=number

specifies the size of the initial tuning window n_1 for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. By default, INITWINDOW=75.

MIDWINDOW=number

MID=number

specifies the size of the first middle tuning window n_2 for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. By default, MIDWINDOW=25.

OVERLAPPING

OVERLAP

OVER

uses overlapping windows for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440.

MAXHEIGHT=number

MAXDEPTH=number

MAXTREE=number

specifies the maximum height of the binary tree $J_{\max}$ for the No-U-Turn Sampler. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. To see how many times the algorithm reaches the maximum tree height during sampling, use the [SAMPLERSUMMARY](#) option. By default, MAXHEIGHT=10.

MAXHEIGHTKILL=k | INFINITY

MAXDEPTHKILL=k | INFINITY

MAXTREEKILL=k | INFINITY

specifies the maximum number of times the No-U-Turn Sampler can reach the maximum binary tree height before the simulation stops. To see how many times the algorithm reaches the maximum tree height during sampling, use the [SAMPLERSUMMARY](#) option. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. You can specify the following values:

k

specifies the maximum number of times. If $k=0$, then there is no maximum.

INFINITY**INFTY****INF**

specifies no maximum number of times.

By default, MAXHEIGHTKILL=0.

NTUNE=number**NTU=number**

specifies the number of iterations n_{tune} for the No-U-Turn Sampler. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440. By default, NTUNE=2000.

RWM <(rwm-options)>

samples from the posterior distribution by using the random walk Metropolis scheme. You can specify the following *rwm-options*:

ACCTOL=value**ATOL=value**

specifies the tolerance ϵ for the acceptance rate during the tuning process of the random walk Metropolis (RWM) sampler. For more information, see the section “[Scale Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, ACCTOL=0.075.

INITSCALE=value**SCALE=value**

specifies the initial scale parameter(s) c^{cur} of the RWM proposal distribution. For more information, see the section “[Scale Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, INITSCALE=2.38.

NTUNE=number**NTU=number**

specifies the number of iterations n_{tune} for each tuning stage of the random walk Metropolis sampler. For more information, see the section “[Tuning the Random Walk Metropolis Sampler](#)” on page 439. By default, NTUNE=500.

NTUNESTAGE=number

specifies the number of tuning stages n_{stage} . For more information, see the section “[Tuning the Random Walk Metropolis Sampler](#)” on page 439. By default, NTUNESTAGE=24.

TARGETRATE=value**RATE=value****DELTA=value**

specifies the target acceptance rate δ of the random walk Metropolis sampler. For more information, see the section “[Tuning Random Walk Metropolis Algorithms](#)” on page 19 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, TARGETRATE=0.234.

TRANSFORM**TRANS**

transforms all parameters to an unconstrained space and implements the random walk Metropolis sampler on the unconstrained space. For more information, see the section “[Parameter Transformation](#)” on page 25 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).”

TUNEWEIGHT=*value*

WEIGHT=*value*

WT=*value*

specifies the tuning weight w to use during the tuning phase of the random walk Metropolis sampler to tune the proposal covariance matrix. For more information, see the section “[Covariance Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, TUNEWEIGHT=0.75.

SAMPLERSETTINGS**SAMSETTINGS****SAMSET**

produces a summary of the MCMC sampler settings that are used to sample from the posterior distribution, including tuning settings.

SAMPLERSUMMARY<(VERBOSE)>**SAMSUMMARY**<(VERBOSE)>**SAMSUM**<(VERBOSE)>

produces a summary of the MCMC sampler that is used to sample from the posterior distribution. For the RWM sampler, this option produces a table of the Metropolis acceptance rates. For NUTS, this option produces a table of the Metropolis acceptance rates as well as a table of NUTS diagnostics. You can specify the following suboption:

VERBOSE

produces a more detailed summary of the MCMC sampler. For the RWM sampler, this option does nothing. For NUTS, this option produces a table of the final mass matrix that is used after tuning. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 440.

SEED=*number*

specifies an integer seed in the range 1 to $2^{31} - 1$ for the random number generator in the simulation. Specifying a seed enables you to reproduce identical Markov chains for the same specification. If you do not specify this option, or if you specify a nonpositive seed, a random seed is generated by reading the time of day from the computer’s clock.

STATISTICS<(stat-option)>=**ALL** | **NONE** | *keyword* | (*keyword-list*)

STATISTIC<(stat-option)>=**ALL** | **NONE** | *keyword* | (*keyword-list*)

STATS<(stat-option)>=**ALL** | **NONE** | *keyword* | (*keyword-list*)

STAT<(stat-option)>=**ALL** | **NONE** | *keyword* | (*keyword-list*)

specifies which posterior summary statistics to produce. If the value of the NCHAIN= option is greater than 1, then instead of the posterior summary statistics being computed separately for each Markov chain, by default the posterior summary statistics are computed by using all chains. For more information, see the section “[Using Multiple Chains](#)” on page 37 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *stat-option*:

BYCHAIN

produces posterior summary statistics tables for each Markov chain. If NCHAIN=1, then the BYCHAIN option does nothing.

To produce all statistics, specify STATISTICS=ALL. To produce no statistics, specify STATISTICS=NONE. To produce some but not all statistics, or to change certain settings of these statistics, specify one or more of the following keywords. If you specify multiple keywords, separate them by using spaces.

CORR**COR**

produces the posterior correlation matrix.

COV

produces the posterior covariance matrix.

INTERVAL <(interval-option)>**INTERVALS** <(interval-option)>**INT** <(interval-option)>

produces equal-tail posterior credible intervals and highest posterior density (HPD) credible intervals. For more information, see the sections “[Interval Estimation](#)” on page 10 and “[Summary Statistics](#)” on page 35 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *interval-option*:

ALPHA=*numeric-list*

ALPHAS=*numeric-list*

computes posterior credible intervals at the levels that you specify in the *numeric-list*. Elements in the *numeric-list* must be between 0 and 1, and values can be separated either by spaces or by commas. Repeated values are removed. Each value in the *numeric-list* produces a pair of 100(1-*value*)% equal-tail and HPD intervals for each parameter. By default, ALPHA=0.05, which yields the 95% credible intervals for each parameter.

SUMMARIES <(summary-option)>**SUMMARY** <(summary-option)>**SUM** <(summary-option)>

produces the means, standard deviations, and percentiles for the posterior sample. For more information, see the section “[Summary Statistics](#)” on page 35 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *summary-option*:

PERCENTILES=*numeric-list*

PERCENTILE=*numeric-list*

PERCENT=*numeric-list*

PERC=*numeric-list*

PCT=*numeric-list*

computes the percentiles of the posterior sample that you specify in the *numeric-list*. Elements in the *numeric-list* must be between 0 and 100, and values can be separated either by spaces or by commas. Repeated values are removed. By default, PERCENTILES=(2.5 25 50 75 97.5), which yields the 2.5th, 25th, 50th, 75th, and 97.5th percentiles, respectively, for each parameter.

By default, STATISTICS=SUMMARIES.

THIN=*k*

THINNING=*k*

thins the Markov chain by using only one of every *k* iterations. For more information, see the section “Burn-In, Thinning, and Markov Chain Samples” on page 27 in Chapter 2, “Introduction to Bayesian Analysis Procedures.” By default, THIN=1.

BOUNDS Statement

BOUNDS *bound1* [, *bound2* ...] ;

The BOUNDS statement imposes simple boundary constraints on the parameter estimates. You can specify any number of BOUNDS statements.

Each *bound* is composed of parameter names, constants, and inequality operators as follows:

item operator item [*operator item* [*operator item* ...]]

Each *item* is a constant, a parameter name, or a list of parameter names. Each *operator* is <, >, <=, or >=. Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as '*', '-', '(', or ')'), then you must use the internal name of the parameter in order to refer to that parameter in the BOUNDS statement. You can specify the PRINTINTERNALNAMES option in the PROC CNTSELECT statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column is added to the “Parameter Estimates” table, which shows the internal name of each parameter.

For more information about how parameters are named, see the section “Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, INIT, and PRIOR Statements” on page 433.

You can use both the BOUNDS statement and the RESTRICT statement to impose boundary constraints. However, the BOUNDS statement provides a simpler syntax for specifying these kinds of constraints. For more information, see the section “RESTRICT Statement” on page 413.

The following BOUNDS statement illustrates the use of parameter lists to specify boundary constraints. It constrains the estimates of the parameter for *z* to be negative, the parameters for *x1* through *x10* to be between 0 and 1, and the parameter for *x1* in the zero-inflation model to be less than 1.

```
bounds z < 0,
       0 < x1-x10 < 1,
       Inf_x1 < 1;
```

The BOUNDS statement is not supported if a BAYES statement is also specified. In Bayesian analysis, the restrictions on parameters are usually introduced through the prior distribution.

BY Statement

BY *variables* ;

A BY statement can be used with PROC CNTSELECT to obtain separate analyses on observations in groups that are defined by the BY *variables*.

CLASS Statement

CLASS *variable* <(options)>... <*variable* <(options)>> </global-options> ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. You can list the response variable for binary models in the CLASS statement, but this is not required. Table 10.2 summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “CLASS Statement” on page 56 in Chapter 4, “Shared Concepts.”

Table 10.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable
SPLIT	Splits levels of CLASS variables into independent effects

DISPLAY Statement

DISPLAY <*table-list*> </options> ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 445. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 445. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

DISPMODEL Statement

DISPMODEL *dependent-variable* ~ < *dispersion-related-regressors* > ;

The DISPMODEL statement specifies the *dispersion-related-regressors* that are used to model dispersion. This statement is ignored unless you specify either DIST=CMPOISSON or DIST=ZICMPOISSON in the MODEL statement. The *dependent-variable* in the DISPMODEL statement must be the same as the *dependent-variable* in the MODEL statement. The DISPMODEL statement is not compatible with the BAYES statement.

The *dependent-variable* that appears in the DISPMODEL statement is directly used to model dispersion. Each of the q variables to the right of the tilde ($\sim$) has a parameter to be estimated in the regression. For example, let $\mathbf{g}_i'$ be the i th observation's $1 \times (q + 1)$ vector of values of the q dispersion explanatory variables (q_0 is set to 1 for the intercept term). Then the dispersion is a function of $\mathbf{g}_i' \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ is the $(q + 1) \times 1$ vector of parameters to be estimated, the dispersion model intercept is δ_0 , and the coefficients for the q dispersion covariates are $\delta_1, \dots, \delta_q$. If you specify DISP=CMPOISSON in the MODEL statement but do not include a DISPMODEL statement, then only the intercept term δ_0 is estimated. The “Parameter Estimates” table in the displayed output shows the estimates for the dispersion intercept and dispersion explanatory variables; they are labeled with the prefix “Disp_”. For example, the dispersion intercept is labeled “Disp_Intercept”. If you specify Age (a variable in your data table) as a dispersion explanatory variable, then the “Parameter Estimates” table labels the corresponding parameter estimate “Disp_Age”. The following statements fit a Conway-Maxwell-Poisson model by using the regressors SEX, ILLNESS, and INCOME and by using AGE as a dispersion-related regressor:

```
proc cntselect data=mylib.docvisit;
  model doctorvisits=sex illness income / dist=cmpoisson;
  dispmode doctorvisits ~ age;
run;
```

EFFECT Statement

EFFECT *name=effect-type (variables < / options>)* ;

The EFFECT statement enables you to construct special collections of columns for design matrices. These collections are referred to as *constructed effects* to distinguish them from the usual model effects that are formed from continuous or classification variables, as discussed in the section “GLM Parameterization of Classification Variables and Effects” on page 94 in Chapter 4, “Shared Concepts.”

You can specify the following *effect-types*:

COLLECTION	specifies a collection effect that defines one or more variables as a single effect that has multiple degrees of freedom. The variables in a collection are considered as a unit for purposes of estimation and inference.
MULTIMEMBER MM	specifies a multimember classification effect whose levels are determined by one or more variables that appear in a CLASS statement.
POLYNOMIAL POLY	specifies a multivariate polynomial effect in the specified numeric variables.
SPLINE	specifies a regression spline effect whose columns are univariate spline expansions of one or more variables. A spline expansion replaces the original variable with an expanded or larger set of new variables.

Table 10.3 summarizes the *options* available in the EFFECT statement.

Table 10.3 EFFECT Statement Options

Option	Description
Collection Effects Options	
DETAILS	Displays the constituents of the collection effect
Multimember Effects Options	
DETAILS	Displays the levels of the multimember effect
NOEFFECT	Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns
STDIZE	Standardizes the design matrix entries so that each observation has a sum of 1
WEIGHT=	Specifies the weight variable for the contributions of each classification effect
Polynomial Effects Options	
DEGREE=	Specifies the degree of the polynomial
DETAILS	Displays details of the specified polynomial
MDEGREE=	Specifies the maximum degree of any variable in a term of the polynomial
NOSEPARATE	Treats the polynomial as a single effect with multiple degrees of freedom
STANDARDIZE=	Specifies centering and scaling suboptions for the variables that define the polynomial

Table 10.3 *continued*

Option	Description
Spline Effects Options	
BASIS=	Specifies the type of basis (B-spline basis or truncated power function basis) for the spline effect
DATABOUNDARY	Uses the extremes of the data as boundary knots for a B-spline basis
DEGREE=	Specifies the degree of the spline effect
DETAILS	Displays the knots and locations for each spline basis function
KNOTMAX=	Requests equally spaced right-side boundary knots starting at the variables' maximum and ending at the KNOTMAX= value
KNOTMETHOD=	Specifies how to construct the knots for the spline effect
KNOTMIN=	Requests equally spaced left-side boundary knots starting at the KNOTMIN= value and ending at the variables' minimum value
NATURALCUBIC	Specifies a natural cubic spline basis for the spline effect
SEPARATE	Treats the spline basis for each variable as a separate effect when multiple variables are specified
SPLIT	Treats each design matrix column as a separate effect for selection methods

For more information about the syntax of these *effect-types* and how columns of constructed effects are computed, see the section “**EFFECT Statement**” on page 62 in Chapter 4, “**Shared Concepts**.”

FREQ Statement

FREQ *freq-variable* ;

The FREQ statement identifies a variable (*freq-variable*) that contains the frequency of occurrence of each observation. PROC CNTSELECT treats each observation as if it appears n times, where n is the value of *freq-variable* for the observation. If the value for the observation is not an integer, it is truncated to an integer. If the value is less than 1 or missing, the observation is not used in the model fitting. When you omit the FREQ statement, each observation is assigned a frequency of 1. The FREQ statement is not compatible with panel models that you specify by using the ERRORCOMP= option. For information about combining the FREQ statement with the **BAYES** statement, see the section “**Bayesian Analysis with Weighted Likelihoods**” on page 440.

INIT Statement

INIT *initialization1* < , *initialization2* ... > ;

The INIT statement sets initial values for parameters in the optimization.

Each *initialization* is written as a parameter or parameter list, followed by an optional equal sign (=), followed by a number:

parameter <=> *number*

Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter in order to refer to that parameter in the INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC CNTSELECT statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column is added to the “Parameter Estimates” table, which shows the internal name of each parameter.

For more information about how parameters are named, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, INIT, and PRIOR Statements](#)” on page 433.

MODEL Statement

MODEL *dependent-variable* = *regressors* </ options> ;

The MODEL statement specifies the dependent variable and independent regressor variables for the regression model. The dependent count variable should take only nonnegative integer values from the input data table. PROC CNTSELECT rounds any positive noninteger count value to the nearest integer. PROC CNTSELECT discards any observation that has a negative count.

Only one MODEL statement can be specified. You can specify the following *options* in the MODEL statement after a slash (/):

DIST=*value*

specifies a type of model to be analyzed. You can specify the following *values*:

POISSON P	specifies the Poisson regression model.
CMPOISSON C CMP	specifies a Conway-Maxwell-Poisson regression model.
NEGBIN(P=1)	specifies the negative binomial regression model that uses a linear variance function.
NEGBIN(P=2) NEGBIN	specifies the negative binomial regression model that uses a quadratic variance function.
ZIPOISSON ZIP	specifies zero-inflated Poisson regression.
ZICMPOISSON ZICMP	specifies a zero-inflated Conway-Maxwell-Poisson regression. The ZERO-MODEL statement must be specified when this model type is specified.
ZINEGBIN ZINB	specifies zero-inflated negative binomial regression.

You can also specify the DIST option in the CNTSELECT statement. The BAYES statement is not compatible with the CMPOISSON and ZICMPOISSON options.

ERRORCOMP=FIXED | RANDOM

specifies a type of conditional panel model to be analyzed. You can specify the following model types:

- | | |
|---------------|---|
| FIXED | specifies a fixed-effect error component regression model. |
| RANDOM | specifies a random-effect error component regression model. |

The BAYES, FREQ, and WEIGHT statements are not compatible with the ERRORCOMP= option.

NOINT

suppresses the intercept parameter.

OFFSET=offset-variable

specifies a variable in the input data table to be used as an offset variable. The *offset-variable* is used to allow the observational units to vary across observations. For example, when the number of shipping accidents could be measured across different time periods or the number of students who participate in an activity could be reported across different class sizes, the observational units need to be adjusted to a common denominator by using the offset variable. The offset variable appears as a covariate in the model with its parameter restricted to 1. The offset variable cannot be the response variable, the zero-inflation offset variable (if any), or any of the explanatory variables. The “Model Fit Summary” table gives the name of the data table variable that is used as the offset variable; it is labeled “Offset.”

PARAMETER=MU | LAMBDA

specifies the parameterization for the Conway-Maxwell-Poisson model. The following parameterizations are supported:

- | | |
|---------------|--|
| LAMBDA | estimates the original Conway-Maxwell-Poisson model (Shmueli et al. 2005). |
| MU | reparameterizes λ as documented by Guikema and Coffelt (2008), where $\mu = \lambda^{1/\nu}$ and the integral part of μ represents the mode, which can be considered a measure of central tendency (mean). |

By default, PARAMETER=MU.

Printing Options

You can specify the following options in either the PROC CNTSELECT statement or the MODEL statement:

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

NOPRINT

suppresses all printed output.

PRINTALL

requests all printing options.

OUTPUT Statement

OUTPUT < *output-options* > ;

The OUTPUT statement creates a new data table that includes variables created by the *output-options*. These variables include the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the probability of the response variable taking on the current value. Furthermore, if a zero-inflated model was fit, you can request that the output data table contain the estimates of $\mathbf{z}_i' \boldsymbol{\gamma}$ and the probability that the response is zero as a result of the zero-generating process. For the Conway-Maxwell-Poisson model, the estimates of $\mathbf{g}_i' \boldsymbol{\delta}$, λ , ν , μ , mode, variance, and dispersion are also available. Except for the probability of the current value, these statistics can be computed for all observations in which the regressors are not missing, even if the response is missing. By adding observations that have missing response values to the input data table, you can compute these statistics for new observations or for settings of the regressors that are not present in the data without affecting the model fit.

You can specify only one OUTPUT statement. You can specify the following *output-options*:

OUT=*libref.data-table*

names the output data table for PROC CNTSELECT to use. You must specify this option before any other options. *libref.data-table* is a two-level name, where

<i>libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>libref</i> , see the section “Using CAS Sessions and CAS Engine Librefs” on page 378.
---------------	--

<i>data-table</i>	specifies the name of the output data table.
-------------------	--

COPYVAR=*SAS-variable-names*

COPYVARS=*SAS-variable-names*

adds SAS variables to the output data table.

PRED=*name*

MEAN=*name*

names the variable to contain the predicted value of the response variable.

PROB=*name*

names the variable to contain the probability that the response variable will take the actual value, $\Pr(Y = y_i)$.

PROBCOUNT(*value1* < *value2* ... >)

outputs the probability that the response variable will take particular values. Each *value* should be a nonnegative integer. If you specify a noninteger, it is rounded to the nearest integer. The *value* can also be a list of the form X TO Y BY Z. For example, PROBCOUNT(0 1 2 TO 10 BY 2 15) requests predicted probabilities for counts 0, 1, 2, 4, 5, 6, 8, 10, and 15. This option is not available for the fixed-effects and random-effects panel models.

PROBZERO=*name*

names the variable to contain the value of φ_i , which is the probability that the response variable will take the value of 0 as a result of the zero-generating process. This variable is written to the output file only if the model is zero-inflated.

VARIANCE=*name*

assigns a name to the variable that contains the estimate of variance.

XBETA=*name*

names the variable to contain estimates of $\mathbf{x}_i' \boldsymbol{\beta}$.

ZGAMMA=*name*

names the variable to contain estimates of $\mathbf{z}_i' \boldsymbol{\gamma}$.

You can specify the following additional *output-options* if you specify a Conway-Maxwell-Poisson regression in the MODEL statement:

DISPERSION=*name*

assigns a name to the variable that contains the value of dispersion.

GDELTA=*name*

assigns a name to the variable that contains estimates of $\mathbf{g}_i' \boldsymbol{\delta}$.

LAMBDA=*name*

assigns a name to the variable that contains the estimate of λ .

MODE=*name*

assigns a name to the variable that contains the integral part of μ (mode).

MU=*name*

assigns a name to the variable that contains the estimate of μ .

NU=*name*

assigns a name to the variable that contains the estimate of ν .

PRIOR Statement

PRIOR *parameter-list* ~ *distribution* ;

The PRIOR statement specifies the prior distribution of the model parameters for Bayesian analysis. You must specify a single parameter or a list of parameters, a tilde (~), and then a distribution with its parameters. You can specify multiple PRIOR statements, although PRIOR statements for parameters that do not exist in the model are ignored. For information about how parameters are named so that you can refer to them when specifying prior distributions, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, INIT, and PRIOR Statements](#)” on page 433. Alternatively, specifying the SHOWNAMES suboption of the PRIORSUMMARY option in the BAYES statement displays the names that are used for all parameters in the prior summary table.

You can specify the following *distributions*. All hyperparameters of the distribution must be specified, except LOWER and UPPER, which determine the support of the distribution and allow for truncated versions of many of them. For more information about these *distributions*, see the section “[Standard Prior Distributions](#)” on page 12 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” For more information about truncated prior distributions, see the section “[Truncated Prior Distributions](#)” on page 442.

CAUCHY(*cauchy-options*)

specifies independently and identically distributed [Cauchy distributions](#) for the *parameter-list*.

You must specify the following *cauchy-options*:

LOCATION= μ

LOC= μ

specifies the location parameter of the Cauchy distribution.

SCALE= γ

specifies the scale parameter of the Cauchy distribution, where $\gamma > 0$.

You can also specify the following *cauchy-options*:

LOWER= L | **NEGATIVEINFINITY**

LB= L | **NEGATIVEINFINITY**

specifies the lower bound of the Cauchy distribution, where $L < U$. You can specify the following values:

L

uses L as the lower bound, such that the prior becomes a truncated Cauchy distribution.

NEGATIVEINFINITY

NEGINF

NEGINF

uses negative infinity as the lower bound.

By default, LOWER=NEGATIVEINFINITY.

UPPER=U | INFINITY**UB=U | INFINITY**

specifies the upper bound of the Cauchy distribution, where $U > L$. You can specify the following values:

U

uses *U* as the upper bound, such that the prior becomes a truncated Cauchy distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

GAMMA(gamma-options)

specifies independently and identically distributed [gamma distributions](#) for the *parameter-list*. You must specify the following *gamma-options*:

SCALE=b

specifies the scale parameter of the gamma distribution, where $b > 0$.

SHAPE=a

specifies the shape parameter of the gamma distribution, where $a > 0$.

You can also specify the following *gamma-options*:

LOWER=L**LB=L**

specifies the lower bound of the gamma distribution, where $L \geq 0$ and $L < U$. If $L > 0$, the prior becomes a truncated gamma distribution.

By default, LOWER=0.

UPPER=U | INFINITY**UB=U | INFINITY**

specifies the upper bound of the gamma distribution, where $U > L$. You can specify the following values:

U

uses *U* as the upper bound, such that the prior becomes a truncated gamma distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

IGAMMA(*igamma-options*)

specifies independently and identically distributed **inverse gamma distributions** for the *parameter-list*. You must specify the following *igamma-options*:

SCALE= b

specifies the scale parameter of the inverse gamma distribution, where $b > 0$.

SHAPE= a

specifies the shape parameter of the inverse gamma distribution, where $a > 0$.

You can also specify the following *igamma-options*:

LOWER= L **LB**= L

specifies the lower bound of the inverse gamma distribution, where $L \geq 0$ and $L < U$. If $L > 0$, the prior becomes a truncated inverse gamma distribution.

By default, LOWER=0.

UPPER= U | **INFINITY****UB**= U | **INFINITY**

specifies the upper bound of the inverse gamma distribution, where $U > L$. You can specify the following values:

U

uses U as the upper bound, such that the prior becomes a truncated inverse gamma distribution.

INFINITY

INFTY

INF

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

NORMAL(*normal-options*)

specifies independently and identically distributed **normal distributions** for the *parameter-list*. You must specify the MEAN= *normal-option* and either the SD= or VAR= *normal-option*:

MEAN= μ

specifies the mean parameter of the normal distribution.

SD= σ

specifies the standard deviation parameter of the normal distribution, where $\sigma > 0$.

VAR= σ^2

specifies the variance parameter of the normal distribution, where $\sigma^2 > 0$.

You can also specify the following *normal-options*:

LOWER=L | NEGATIVEINFINITY**LB=L | NEGATIVEINFINITY**

specifies the lower bound of the normal distribution, where $L < U$. You can specify the following values:

L

uses L as the lower bound, such that the prior becomes a truncated normal distribution.

NEGATIVEINFINITY**NEGINF****NEGINF**

uses negative infinity as the lower bound.

By default, LOWER=NEGATIVEINFINITY.

UPPER=U | INFINITY**UB=U | INFINITY**

specifies the upper bound of the normal distribution, where $U > L$. You can specify the following values:

U

uses U as the upper bound, such that the prior becomes a truncated normal distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

SQRTGAMMA(sqrtgamma-options)

specifies independently and identically distributed [square root gamma distributions](#) for the *parameter-list*. You must specify the following *sqrtgamma-options*:

SCALE=b

specifies the scale parameter of the square root gamma distribution, where $b > 0$.

SHAPE=a

specifies the shape parameter of the square root gamma distribution, where $a > 0$.

You can also specify the following *sqrtgamma-options*:

LOWER=L**LB=L**

specifies the lower bound of the square root gamma distribution, where $L \geq 0$ and $L < U$. If $L > 0$, the prior becomes a truncated square root gamma distribution.

By default, LOWER=0.

UPPER=U | INFINITY**UB=U | INFINITY**

specifies the upper bound of the square root gamma distribution, where $U > L$. You can specify the following values:

U

uses U as the upper bound, such that the prior becomes a truncated square root gamma distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

SQRTIGAMMA(*sqrtigamma-options*)

specifies independently and identically distributed [square root inverse gamma distributions](#) for the *parameter-list*. You must specify the following *sqrtigamma-options*:

SCALE=b

specifies the scale parameter of the square root inverse gamma distribution, where $b > 0$.

SHAPE=a

specifies the shape parameter of the square root inverse gamma distribution, where $a > 0$.

You can also specify the following *sqrtigamma-options*:

LOWER=L**LB=L**

specifies the lower bound of the square root inverse gamma distribution, where $L \geq 0$ and $L < U$. If $L > 0$, the prior becomes a truncated square root inverse gamma distribution.

By default, LOWER=0.

UPPER=U | INFINITY**UB=U | INFINITY**

specifies the upper bound of the square root inverse gamma distribution, where $U > L$. You can specify the following values:

U

uses U as the upper bound, such that the prior becomes a truncated square root inverse gamma distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

T(*t-options*)

specifies independently and identically distributed *t* distributions for the *parameter-list*.

You must specify the following *t-options*:

DF= ν

specifies the degrees-of-freedom parameter of the *t* distribution, where $\nu > 0$.

LOCATION= μ

LOC= μ

specifies the location parameter of the *t* distribution.

SCALE= γ

specifies the scale parameter of the *t* distribution, where $\gamma > 0$.

You can also specify the following *t-options*:

LOWER= L | NEGATIVEINFINITY

LB= L | NEGATIVEINFINITY

specifies the lower bound of the *t* distribution, where $L < U$. You can specify the following values:

L

uses *L* as the lower bound, such that the prior becomes a truncated *t* distribution.

NEGATIVEINFINITY

NEGINTY

NEGINF

uses negative infinity as the lower bound.

By default, LOWER=NEGATIVEINFINITY.

UPPER= U | INFINITY

UB= U | INFINITY

specifies the upper bound of the *t* distribution, where $U > L$. You can specify the following values:

U

uses *U* as the upper bound, such that the prior becomes a truncated *t* distribution.

INFINITY

INTY

INF

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

UNIFORM(*uniform-options*)

specifies independently and identically distributed **uniform distributions** for the *parameter-list*. You must specify both the following *uniform-options*:

MAX=*M* | INFINITY

specifies the maximum value of the uniform distribution, where $M < m$. You can specify the following values:

M

uses *M* as the maximum value.

INFINITY

INFTY

INF

uses positive infinity as the maximum value, such that the prior becomes an improper uniform prior.

MIN=*m* | NEGATIVEINFINITY

specifies the minimum value of the uniform distribution, where $m < M$. You can specify the following values:

m

uses *m* as the minimum value.

NEGATIVEINFINITY

NEGINFTY

NEGINF

uses negative infinity as the minimum value, such that the prior becomes an improper uniform prior.

RESTRICT Statement

RESTRICT *restriction1* [, *restriction2* ...] ;

The RESTRICT statement imposes linear restrictions on the parameter estimates. You can specify any number of RESTRICT statements.

Each *restriction* is written either as a single linear equation or as a comma-separated list of two or more linear equations. A restriction equation consists of an *expression*, followed by an equality *operator* (=) or an inequality *operator* (<, >, <=, >=), followed by a second *expression*:

expression operator expression

The *operator* can be =, <, >, <=, or >=.

Lagrange multipliers are reported in the “Parameter Estimates” table for all the active linear constraints. They are identified by the names Restrict1, Restrict2, and so on. The test statistics and *p*-values for these restrictions are computed using the Lagrange multiplier test that is described in the section “[Tests on Parameters](#)” on page 438. Nonactive (nonbinding) restrictions have no effect on the estimation results and are not noted in the output.

RESTRICT Statement Expressions

A restriction expression is composed of parameter names, constants, and the operators times (*), plus (+), and minus (−). Each restriction expression must be a linear function of the parameters in the model. In addition, no grouping symbols (such as parentheses) are allowed, and the constant factor in any product can appear only on the left-hand side of the times (*) operator.

In the following example, the assumption is that you have a data table in which *y* is the count variable and *x1*–*x3* are continuous variables. The PROC CNTSELECT program uses a RESTRICT statement to impose a restriction on the estimate for the parameter that is associated with the variable *x2*. Thus, in any solution that the optimizer finds, the solution must satisfy the condition that the parameter that is associated with the variable *x2* is equal to 1.5:

```
proc cntselect data=mylib.exrestrict;
  model y = x1-x3;
  restrict x2=1.5;
run;
```

It is important to keep in mind that the parameters that are associated with the variables, not the variables themselves, are restricted. Thus, in the preceding RESTRICT statement, the variable name “*x2*” refers to the parameter that is associated with the variable *x2*, not to the variable itself.

Parameter names are shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as “*”, “-”, “(”, or “)”), then you must use the internal name of the parameter in order to refer to that parameter in the RESTRICT statement. You can specify the PRINTINTERNALNAMES option in the PROC CNTSELECT statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column is added to the “Parameter Estimates” table, which shows the internal name of each parameter.

For more information about how parameters are named, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, INIT, and PRIOR Statements](#)” on page 433.

Restrictions should be consistent and not redundant. All restriction equations in all RESTRICT statements are applied jointly.

The RESTRICT statement is not supported if a BAYES statement is also specified. In Bayesian analysis, the restrictions on parameters are usually introduced through the prior distribution.

RESTRICT Statement Examples

Examples of valid RESTRICT statements include the following:

```
restrict x1=0.1;
restrict a+b=1;
restrict a-b=0, b+c=1.5;
restrict 2*f=g+h, intercept+f=0;
```

Examples of invalid RESTRICT statements include the following:

```
restrict x1^2=4;
restrict x1*x3=4;
restrict x1/x3=2;
restrict sin(a)=0;
restrict a*0.5=1;
```



```
restrict 2*(f+h)=1;
```

In the first four examples of invalid RESTRICT statements, the equation is nonlinear. The fifth example is invalid because the constant factor (0.5) cannot appear on the right-hand side of the times (*) operator. The last example is invalid because grouping symbols are not allowed.

The set of restrictions must be consistent. For example, you cannot specify the following because the three restrictions are not consistent:

```
restrict f-g=0,
       f-intercept=0,
       g-intercept=1;
```

The following RESTRICT statement constrains the negative binomial dispersion parameter α to 1, which restricts the conditional variance to be $\mu + \mu^2$:

```
restrict _Alpha = 1;
```

SELECTION Statement

SELECTION <METHOD=*method* <(method-options)> >> options > ;

The SELECTION statement performs model selection by examining whether effects should be added to or removed from the model according to rules that are defined by the selection methods. The statement is fully documented in the section “[SELECTION Statement](#)” on page 78 in Chapter 4, “[Shared Concepts](#).” The CNTSELECT procedure supports only a subset of the *method* values and *method-options* that are listed in that section.

You cannot specify both the SELECTION statement and the BAYES statement.

You can use the following *option* to specify the selection method:

METHOD=BACKWARD | FORWARD | FORWARDSWAP | STEPWISE

specifies the selection method to use. You can specify one of the following methods:

BACKWARD	specifies the backward elimination method.
FORWARD	specifies the forward selection method.
FORWARDSWAP	specifies forward-swap selection, which is an extension of the forward selection method.
STEPWISE	specifies the stepwise regression method.

By default, METHOD=STEPWISE. For more information about each of the selection methods, see the section “[Model Selection Methods](#)” on page 102

PROC CNTSELECT supports a specific set of values for the following *method-options*:

CHOOSE=criterion

chooses from the list of models (at each step of the selection process) the model that yields the best value of the specified criterion. PROC CNTSELECT supports only the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, the value of the CHOOSE= option is the same as the value that you assign to the SELECT= option. For more information, see the detailed description of the [CHOOSE=](#) option.

SELECT=criterion

specifies the criterion that the procedure uses to determine the order in which effects enter or leave at each step of the selection method. PROC CNTSELECT supports the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, SELECT=SBC. For more information, see the detailed description of the [SELECT=](#) option.

STOP=criterion

specifies a criterion to use to stop the selection process. PROC CNTSELECT supports only the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, the value of the STOP= option is the same as the value that you assign to the SELECT= option. For more information, see the detailed description of the [STOP=](#) option.

PROC CNTSELECT supports the following *method-options*, which are described in the section “[SELECTION Statement](#)” on page 78 in Chapter 4, “[Shared Concepts](#)”:

- [COMPETITIVE](#)
- [MAXEFFECTS=](#)
- [MAXSTEPS=](#)
- [MINEFFECTS=](#)

PROC CNTSELECT supports the following statement *options*, which are described in the section “[SELECTION Statement](#)” on page 78 in Chapter 4, “[Shared Concepts](#)”:

- [DETAILS=](#)
- [STOPHORIZON=](#)

TEST Statement

`<label:> TEST <'string'> equation1 <, equation2... > /<test-options> ;`

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the regression parameters that are specified in the preceding MODEL statement.

Each *test* is written either as a single linear *equation* or as a comma-separated list of two or more linear *equations*. A test *equation* specifies a linear hypothesis to be tested and consists of an *expression*, followed by the equality operator (=), followed by a second expression:

expression = *expression*

The rules governing valid test expressions are the same as those for restriction expressions. For more information see the section “[RESTRICT Statement Expressions](#)” on page 414.

All hypotheses in one TEST statement are tested jointly.

You cannot specify both the TEST statement and the BAYES statement.

You can specify the following *test-options* after a slash (/):

ALL

requests Wald, Lagrange multiplier, and likelihood ratio tests.

LM

requests the Lagrange multiplier test.

LR

requests the likelihood ratio test.

WALD

requests the Wald test.

By default, the Wald test is performed.

You can add a label (which is printed in the output) to a TEST statement in two ways: add an unquoted *label* followed by a colon before the TEST keyword, or add a quoted *string* after the TEST keyword. The unquoted *label* cannot contain any spaces. If you include both an unquoted *label* and a quoted *string*, PROC CNTSELECT uses the unquoted *label*. If you specify neither an unquoted *label* nor a quoted *string*, PROC CNTSELECT automatically labels the tests.

The following illustrates the use of the TEST statement:

```
proc cntselect;
  model y = x1 x2 x3;
  test x1 = 0, 1.5 * x2 + 2 * x3 = 0;
  test_int: test intercept = 0, x3 = 0.75;
run;
```

In the example, two separate tests are performed. The first test investigates the joint hypothesis that

$$\beta_1 = 0$$

and

$$1.5\beta_2 + 2\beta_3 = 0$$

The second test is labeled “test_int” and investigates the joint hypothesis that

$$\beta_{\text{Intercept}} = 0$$

and

$$\beta_3 = 0.75$$

VIEWSTORE Statement

VIEWSTORE / *options* ;

The VIEWSTORE statement enables you to print various reports that describe the model fitting information that is preserved in an item store. You must have created the item store by using the STORE= option in the PROC CNTSELECT statement. You can specify the following *options*:

ALL

prints the entire contents of the item store.

BYVARVALUES

prints the values of the BY-group variables that were used to fit the model.

CLASSVARLEVELS

prints the classification variables and levels that were used in the model.

COVARIANCES

prints the covariances matrix.

CORRELATIONS

prints the correlations matrix.

FINALESTIMATES

prints the final parameter estimates.

FITMODELSUMMARY

prints the fit model summary.

INITIALESTIMATES

prints the initial parameter estimates.

INSTORE=SAS-item-store

specifies the item store that you want to view.

MINIMAL

prints only the fit model summary and final parameter estimates.

MODELDEFINITION

prints the statements that defined the model.

NONE

suppresses printing.

OPTIMIZERSETTINGS

prints the optimizer settings that were used to fit the model.

WEIGHT Statement

WEIGHT *variable* </ *option* > ;

The WEIGHT statement specifies a variable to supply weighting values to use for each observation in estimating parameters. The log likelihood for each observation is multiplied by the corresponding weight variable value.

If the weight of an observation is nonpositive, that observation is not used in the estimation.

The following *option* can be added to the WEIGHT statement after a slash (/):

NONNORMALIZE

does not normalize the weights. (By default, the weights are normalized so that they add up to the actual sample size. The weights w_i are normalized by multiplying them by $\frac{n}{\sum_{i=1}^n w_i}$, where n is the sample size.) If the weights are required to be used as they are, then specify the NONNORMALIZE option.

The WEIGHT statement is not compatible with panel models that you specify by using the ERRORCOMP= option. For information about combining the WEIGHT statement with the BAYES statement, see the section “Bayesian Analysis with Weighted Likelihoods” on page 440.

ZEROMODEL Statement

ZEROMODEL *dependent-variable* ~ *zero-inflated-regressors* < / *options* > ;

The ZEROMODEL statement is required if either ZIP or ZINB is specified in the DIST= option in the MODEL statement. If ZIP or ZINB is specified, then the ZEROMODEL statement must follow the MODEL statement. The dependent variable in the ZEROMODEL statement must be the same as the dependent variable in the MODEL statement.

The zero-inflated (ZI) regressors appear in the equation that determines the probability (φ_i) of a zero count. Each of these q variables has a parameter to be estimated in the regression. For example, let $\mathbf{z}'_i$ be the i th observation's $1 \times (q + 1)$ vector of values of the q ZI explanatory variables (w_0 is set to 1 for the intercept term). Then φ_i is a function of $\mathbf{z}'_i \boldsymbol{\gamma}$, where $\boldsymbol{\gamma}$ is the $(q + 1) \times 1$ vector of parameters to be estimated. (The zero-inflated intercept is γ_0 ; the coefficients for the q zero-inflated covariates are $\gamma_1, \dots, \gamma_q$.) If q is equal to 0 (no ZI explanatory variables are provided), then only the intercept term γ_0 is estimated. The “Parameter Estimates” table in the displayed output shows the estimates for the ZI intercept and ZI explanatory variables; they are labeled with the prefix “Inf_”. For example, the ZI intercept is labeled “Inf_intercept”. If you specify

Age (a variable in your data table) as a ZI explanatory variable, then the “Parameter Estimates” table labels the corresponding parameter estimate “Inf_Age”.

You can specify the following *options* in the ZEROMODEL statement after a slash (/):

LINK=LOGISTIC | NORMAL

specifies the distribution function to use to compute probability of zeros. You can specify the following functions:

LOGISTIC specifies logistic distribution.

NORMAL specifies standard normal distribution.

If this option is omitted, then the default ZI link function is logistic.

OFFSET=zero-inflated-offset-variable

specifies a variable in the input data table to be used as a zero-inflated (ZI) offset variable. The ZI offset variable *zero-inflated-offset-variable* is included as a term, with coefficient restricted to 1, in the equation that determines the probability (ϕ_i) of a zero count and represents an adjustment to a common observational unit. The ZI offset variable cannot be the response variable, the offset variable (if any), or any of the explanatory variables. The name of the data table variable that is used as the ZI offset variable is displayed in the “Model Fit Summary” table, where it is labeled as “Inf_offset”.

Details: CNTSELECT Procedure

Missing Values

Any observations in the input data table that have a missing value for one or more of the regressors are ignored by PROC CNTSELECT and not used in the model fit. PROC CNTSELECT rounds any positive noninteger count values to the nearest integer and ignores any observations that have a negative count.

If the input data table contains any observations that have missing response values but nonmissing regressors, PROC CNTSELECT can compute several statistics and store them in an output data table by using the OUTPUT statement. For example, you can request that the output data table contain the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the probability that the response variable will take the current value. Furthermore, if a zero-inflated model was fit, you can request that the output data table contain the estimates of $\mathbf{z}_i' \boldsymbol{\gamma}$, and the probability that the response is 0 as a result of the zero-generating process. Note that the presence of such observations (that have missing response values) does not affect the model fit.

Poisson Regression

The most widely used model for count data analysis is Poisson regression. Poisson regression assumes that y_i , given the vector of covariates $\mathbf{x}_i$, is independently Poisson distributed with

$$P(Y_i = y_i | \mathbf{x}_i) = \frac{e^{-\mu_i} \mu_i^{y_i}}{y_i!}, \quad y_i = 0, 1, 2, \dots$$

and the mean parameter—that is, the mean number of events per period—is given by

$$\mu_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

where $\boldsymbol{\beta}$ is a $(k + 1) \times 1$ parameter vector. (The intercept is β_0 ; the coefficients for the k regressors are $\beta_1, \dots, \beta_k$.) Taking the exponential of $\mathbf{x}_i' \boldsymbol{\beta}$ ensures that the mean parameter μ_i is nonnegative. It can be shown that the conditional mean is given by

$$E(y_i | \mathbf{x}_i) = \mu_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

Note that the conditional variance of the count random variable is equal to the conditional mean in the Poisson regression model:

$$V(y_i | \mathbf{x}_i) = E(y_i | \mathbf{x}_i) = \mu_i$$

The equality of the conditional mean and variance of y_i is known as *equidispersion*.

The standard estimator for the Poisson model is the maximum likelihood estimator (MLE). Because the observations are independent, the log-likelihood function is written as

$$\mathcal{L} = \sum_{i=1}^N (-\mu_i + y_i \ln \mu_i - \ln y_i!) = \sum_{i=1}^N (-e^{\mathbf{x}_i' \boldsymbol{\beta}} + y_i \mathbf{x}_i' \boldsymbol{\beta} - \ln y_i!)$$

For more information about the Poisson regression model, see [SAS/ETS User's Guide](#).

The Poisson model has been criticized for its restrictive property that the conditional variance equals the conditional mean. Real-life data are often characterized by *overdispersion*—that is, the variance exceeds the mean. Allowing for overdispersion can improve model predictions because the Poisson restriction of equal mean and variance results in the underprediction of zeros when overdispersion exists. The most commonly used model that accounts for overdispersion is the negative binomial model. Conway-Maxwell-Poisson regression enables you to model both overdispersion and underdispersion.

Conway-Maxwell-Poisson Regression

The Conway-Maxwell-Poisson (CMP) distribution is a generalization of the Poisson distribution that enables you to model both underdispersed and overdispersed data. It was originally proposed by Conway and Maxwell (1962), but its implementation to model under- and overdispersed count data is attributed to Shmueli et al. (2005).

Recall that y_i , given the vector of covariates $\mathbf{x}_i$, is independently Poisson-distributed as

$$P(Y_i = y_i | \mathbf{x}_i) = \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!}, \quad y_i = 0, 1, 2, \dots$$

The CMP distribution is defined as

$$P(Y_i = y_i | \mathbf{x}_i, \mathbf{g}_i) = \frac{1}{Z(\lambda_i, \nu_i)} \frac{\lambda_i^{y_i}}{(y_i!)^{\nu_i}}, \quad y_i = 0, 1, 2, \dots$$

where the normalization factor is

$$Z(\lambda_i, \nu_i) = \sum_{n=0}^{\infty} \frac{\lambda_i^n}{(n!)^{\nu_i}}$$

and

$$\lambda_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

$$\nu_i = \exp(-\mathbf{g}_i' \boldsymbol{\delta})$$

The vector $\boldsymbol{\beta}$ is a $(k + 1) \times 1$ parameter vector. (The intercept is β_0 , and the coefficients for the k regressors are $\beta_1, \dots, \beta_k$.) The vector $\boldsymbol{\delta}$ is an $(m + 1) \times 1$ parameter vector. (The intercept is represented by δ_0 , and the coefficients for the m regressors are $\delta_1, \dots, \delta_m$.) The covariates are represented by the vectors $\mathbf{x}_i$ and $\mathbf{g}_i$.

One of the restrictive properties of the Poisson model is that the conditional mean and variance must be equal:

$$E(y_i | \mathbf{x}_i) = V(y_i | \mathbf{x}_i) = \lambda_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

The CMP distribution overcomes this restriction by defining an additional parameter, ν , which governs the rate of decay of successive ratios of probabilities such that

$$P(Y_i = y_i - 1) / P(Y_i = y_i) = \frac{(y_i)^{\nu_i}}{\lambda_i}$$

The introduction of the additional parameter, ν , allows for flexibility in modeling the tail behavior of the distribution. If $\nu = 1$, the ratio is equal to the rate of decay of the Poisson distribution. If $\nu < 1$, the rate of decay decreases, enabling you to model processes that have longer tails than the Poisson distribution's (overdispersed data). If $\nu > 1$, the rate of decay increases in a nonlinear fashion, thus shortening the tail of the distribution (underdispersed data).

The CMP distribution has several special cases. If $\lambda < 1$ and $\nu \rightarrow \infty$, the CMP distribution results in the Bernoulli distribution. In this case, the data can take only the values 0 and 1; this represents an extreme

underdispersion. If $\nu = 1$, the Poisson distribution is recovered along with its equidispersion property. When $\nu = 0$ and $\lambda < 1$, the normalization factor is convergent and forms a geometric series,

$$Z(\lambda_i, 0) = \frac{1}{1 - \lambda_i}$$

and the probability density function becomes

$$P(Y = y_i; \lambda_i, \nu_i = 0) = (1 - \lambda_i) \lambda_i^{y_i}$$

The geometric distribution represents a case of severe overdispersion.

Mean, Variance, and Dispersion for the Conway-Maxwell-Poisson Model

The mean and the variance of the Conway-Maxwell-Poisson distribution are defined as

$$E(Y) = \frac{\partial \ln Z}{\partial \ln \lambda}$$

$$V(Y) = \frac{\partial^2 \ln Z}{\partial^2 \ln \lambda}$$

The CMP distribution does not have closed-form expressions for its moments in terms of its parameters λ and ν . However, the moments can be approximated. Shmueli et al. (2005) use asymptotic expressions for Z to derive $E(Y)$ and $V(Y)$ as

$$E(Y) \approx \lambda^{1/\nu} + \frac{1}{2\nu} - \frac{1}{2}$$

$$V(Y) \approx \frac{1}{\nu} \lambda^{1/\nu}$$

Shmueli et al. (2005) noted that these approximations might not be accurate for underdispersed data ($\nu > 1$) or for values of $\lambda^{1/\nu} < 10$.

In the CMP model, the summation of infinite series is evaluated using a logarithmic expansion. The mean and variance are calculated as follows for the Shmueli et al. (2005) model:

$$E(Y) = \frac{1}{Z(\lambda, \nu)} \sum_{j=0}^{\infty} \frac{j \lambda^j}{(j!)^\nu}$$

$$V(Y) = \frac{1}{Z(\lambda, \nu)} \sum_{j=0}^{\infty} \frac{j^2 \lambda^j}{(j!)^\nu} - E(Y)^2$$

The dispersion is defined as

$$D(Y) = \frac{V(Y)}{E(Y)}$$

Log-Likelihood Function for the Conway-Maxwell-Poisson Model

The log-likelihood function for the Conway-Maxwell-Poisson regression model can be written as

$$\begin{aligned}\mathcal{L} &= \sum_{i=1}^N (y_i \ln(\lambda_i) - v_i \ln(y_i!) - \ln[Z(\lambda_i, v_i)]) \\ &= \sum_{i=1}^N (y_i \mathbf{x}_i' \boldsymbol{\beta} - \exp(-\mathbf{g}_i' \boldsymbol{\delta}) \ln(y_i!) - \ln[Z(\lambda_i, v_i)])\end{aligned}$$

The gradients can be written as

$$\begin{aligned}\mathcal{L}_{\boldsymbol{\beta}} &= \sum_{i=1}^N \left(y_i - \frac{Z_{\lambda}(\lambda_i, v_i)}{Z(\lambda_i, v_i)} \lambda_i \right) \mathbf{x}_i \\ \mathcal{L}_{\boldsymbol{\delta}} &= \sum_{i=1}^N \left(\ln(y_i!) + \frac{Z_v(\lambda_i, v_i)}{Z(\lambda_i, v_i)} \right) v_i \mathbf{g}_i\end{aligned}$$

where $Z_{\lambda}(\lambda_i, v_i)$ is the derivative of $Z(\lambda, v)$ with respect to λ evaluated at $\lambda = \lambda_i$ and $v = v_i$, and $Z_v(\lambda_i, v_i)$ is the derivative with respect to v evaluated at $\lambda = \lambda_i$ and $v = v_i$.

Conway-Maxwell-Poisson Regression: Guikema and Coffelt (2008) Reparameterization

Guikema and Coffelt (2008) propose a reparameterization of the Shmueli et al. (2005) Conway-Maxwell-Poisson model to provide a measure of central tendency that can be interpreted in the context of the generalized linear model. By substituting $\lambda = \mu^v$, you can write the Guikema and Coffelt (2008) formulation as

$$P(Y = y_i; \mu_i, v_i) = \frac{1}{S(\mu_i, v_i)} \left(\frac{\mu_i^{y_i}}{y_i!} \right)^{v_i}$$

where the new normalization factor is defined as

$$S(\mu, v) = \sum_{n=0}^{\infty} \left(\frac{\mu^n}{n!} \right)^v$$

and

$$\mu_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

$$v_i = \exp(-\mathbf{g}_i' \boldsymbol{\delta})$$

In terms of their new formulations, the mean and variance of Y are given as

$$E(Y) = \frac{1}{v} \frac{\partial \ln S}{\partial \ln \mu}$$

$$V(Y) = \frac{1}{v^2} \frac{\partial^2 \ln S}{\partial^2 \ln \mu}$$

They can be approximated as

$$E(Y) \approx \mu + \frac{1}{2\nu} - \frac{1}{2}$$

$$V(Y) \approx \frac{\mu}{\nu}$$

When you specify the PRED= option in the OUTPUT statement, the CNTSELECT procedure calculates the predicted value of the response variable by using the approximation for $E(Y)$ if $\mu > 20$; otherwise the exact formula is used. In PROC CNTSELECT, the mean and variance are calculated according to the following formulas for the Guikema and Coffelt (2008) model:

$$E(Y) = \frac{1}{Z(\lambda, \nu)} \sum_{n=0}^{\infty} \frac{n \mu^{\nu n}}{(n!)^{\nu}}$$

$$V(Y) = \frac{1}{Z(\lambda, \nu)} \sum_{n=0}^{\infty} \frac{n^2 \mu^{\nu n}}{(n!)^{\nu}} - E(Y)^2$$

In terms of the new parameter μ , the log-likelihood function is specified as

$$\mathcal{L} = \sum_{i=1}^N (v_i y_i \ln(\mu_i) - v_i \ln(y_i!) - \ln[S(\mu_i, v_i)])$$

and the gradients are calculated as

$$\mathcal{L}_{\beta} = \sum_{i=1}^N \left(v_i y_i - \frac{S_{\mu}(\mu_i, v_i)}{S(\mu_i, v_i)} \mu_i \right) \mathbf{x}_i$$

$$\mathcal{L}_{\delta} = \sum_{i=1}^N \left(\ln(y_i!) + \frac{S_{\nu}(\mu_i, v_i)}{S(\mu_i, v_i)} - y_i \log \mu_i \right) v_i \mathbf{g}_i$$

where $S_{\mu}(\mu_i, v_i)$ is the derivative of $S(\mu, \nu)$ with respect to μ evaluated at $\mu = \mu_i$ and $\nu = v_i$, and $S_{\nu}(\mu_i, v_i)$ is the derivative with respect to ν evaluated at $\mu = \mu_i$ and $\nu = v_i$. By default, PROC CNTSELECT uses the Guikema and Coffelt (2008) specification. You can estimate the Shmueli et al. (2005) model by specifying the PARAMETER=LAMBDA option in the MODEL statement. If you specify DISP=CMPOISSON in the MODEL statement and you omit the DISPMODEL statement, the model is estimated according to the Lord, Guikema, and Geedipally (2008) specification, where ν represents a single parameter that does not depend on any covariates. The Lord, Guikema, and Geedipally (2008) specification makes the model comparable to the negative binomial model because it has only one parameter.

The dispersion is defined as

$$D(Y) = \frac{V(Y)}{E(Y)}$$

Using the Guikema and Coffelt (2008) specification results in the integral part of μ representing the mode, which is a reasonable approximation of the mean. The dispersion can be written as

$$D(Y) = \frac{V(Y)}{E(Y)} \approx \frac{\frac{\mu}{\nu}}{\mu + \frac{1}{2\nu} - \frac{1}{2}} \approx \frac{1}{\nu}$$

When $\nu < 1$, the variance can be shown to be greater than the mean and the dispersion can be shown to be greater than 1. This is a result of overdispersed data. When $\nu = 1$ and the mean and variance are equal, the dispersion is equal to 1 (Poisson model). When $\nu > 1$, the variance is smaller than the mean and the dispersion is less than 1. This is a result of underdispersed data.

All Conway-Maxwell-Poisson models in PROC CNTSELECT are parameterized in terms of dispersion, where

$$-\ln(\nu) = \delta_0 + \sum_{n=1}^q \delta_n g_n$$

Negative values of $\ln(\nu)$ indicate that the data are approximately overdispersed, and positive values of $\ln(\nu)$ indicate that the data are approximately underdispersed.

Negative Binomial Regression

The Poisson regression model can be generalized by introducing an unobserved heterogeneity term for observation i . Thus, the individuals are assumed to differ randomly in a manner that is not fully accounted for by the observed covariates. This is formulated as

$$E(y_i | \mathbf{x}_i, \tau_i) = \mu_i \tau_i = e^{\mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i}$$

where the unobserved heterogeneity term $\tau_i = e^{\epsilon_i}$ is independent of the vector of regressors $\mathbf{x}_i$. Then the distribution of y_i conditional on $\mathbf{x}_i$ and τ_i is Poisson with conditional mean and conditional variance $\mu_i \tau_i$:

$$f(y_i | \mathbf{x}_i, \tau_i) = \frac{\exp(-\mu_i \tau_i) (\mu_i \tau_i)^{y_i}}{y_i!}$$

Let $g(\tau_i)$ be the probability density function of τ_i . Then, the distribution $f(y_i | \mathbf{x}_i)$ (no longer conditional on τ_i) is obtained by integrating $f(y_i | \mathbf{x}_i, \tau_i)$ with respect to τ_i :

$$f(y_i | \mathbf{x}_i) = \int_0^{\infty} f(y_i | \mathbf{x}_i, \tau_i) g(\tau_i) d\tau_i$$

An analytical solution to this integral exists when τ_i is assumed to follow a gamma distribution. This solution is the negative binomial distribution. If the model contains a constant term, then in order to identify the mean of the distribution, it is necessary to assume that $E(e^{\epsilon_i}) = E(\tau_i) = 1$. Thus, it is assumed that τ_i follows a gamma(θ, θ) distribution with $E(\tau_i) = 1$ and $V(\tau_i) = 1/\theta$,

$$g(\tau_i) = \frac{\theta^\theta}{\Gamma(\theta)} \tau_i^{\theta-1} \exp(-\theta \tau_i)$$

where $\Gamma(x) = \int_0^\infty z^{x-1} \exp(-z) dz$ is the gamma function and θ is a positive parameter. Then, the density of y_i given $\mathbf{x}_i$ is derived as

$$\begin{aligned} f(y_i|\mathbf{x}_i) &= \int_0^\infty f(y_i|\mathbf{x}_i, \tau_i) g(\tau_i) d\tau_i \\ &= \frac{\theta^\theta \mu_i^{y_i}}{y_i! \Gamma(\theta)} \int_0^\infty e^{-(\mu_i + \theta)\tau_i} \tau_i^{\theta + y_i - 1} d\tau_i \\ &= \frac{\theta^\theta \mu_i^{y_i} \Gamma(y_i + \theta)}{y_i! \Gamma(\theta) (\theta + \mu_i)^{\theta + y_i}} \\ &= \frac{\Gamma(y_i + \theta)}{y_i! \Gamma(\theta)} \left(\frac{\theta}{\theta + \mu_i} \right)^\theta \left(\frac{\mu_i}{\theta + \mu_i} \right)^{y_i} \end{aligned}$$

If you make the substitution $\alpha = \frac{1}{\theta}$ ($\alpha > 0$), the negative binomial distribution can then be rewritten as

$$f(y_i|\mathbf{x}_i) = \frac{\Gamma(y_i + \alpha^{-1})}{y_i! \Gamma(\alpha^{-1})} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \left(\frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}, \quad y_i = 0, 1, 2, \dots$$

Thus, the negative binomial distribution is derived as a gamma mixture of Poisson random variables. It has the conditional mean

$$E(y_i|\mathbf{x}_i) = \mu_i = e^{\mathbf{x}_i' \boldsymbol{\beta}}$$

and the conditional variance

$$V(y_i|\mathbf{x}_i) = \mu_i \left[1 + \frac{1}{\theta} \mu_i \right] = \mu_i [1 + \alpha \mu_i] > E(y_i|\mathbf{x}_i)$$

The conditional variance of the negative binomial distribution exceeds the conditional mean. Overdispersion results from neglected unobserved heterogeneity. The negative binomial model with variance function $V(y_i|\mathbf{x}_i) = \mu_i + \alpha \mu_i^2$, which is quadratic in the mean, is referred to as the NEGBIN2 model Cameron and Trivedi (1986). To estimate this model, specify DIST=NEGBIN(P=2) in the MODEL statement. The Poisson distribution is a special case of the negative binomial distribution where $\alpha = 0$. A test of the Poisson distribution can be carried out by testing the hypothesis that $\alpha = \frac{1}{\theta_i} = 0$. A Wald test of this hypothesis is provided (it is the reported t statistic for the estimated α in the negative binomial model).

The log-likelihood function of the negative binomial regression model (NEGBIN2) is given by

$$\begin{aligned} \mathcal{L} &= \sum_{i=1}^N \left\{ \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1}) - \ln(y_i!) \right. \\ &\quad \left. - (y_i + \alpha^{-1}) \ln(1 + \alpha \exp(\mathbf{x}_i' \boldsymbol{\beta})) + y_i \ln(\alpha) + y_i \mathbf{x}_i' \boldsymbol{\beta} \right\} \end{aligned}$$

where use of the following fact is made if y is an integer:

$$\Gamma(y + a) / \Gamma(a) = \prod_{j=0}^{y-1} (j + a)$$

Cameron and Trivedi (1986) consider a general class of negative binomial models that have mean μ_i and variance function $\mu_i + \alpha\mu_i^p$. The NEGBIN2 model, with $p = 2$, is the standard formulation of the negative binomial model. Models that have other values of p , $-\infty < p < \infty$, have the same density $f(y_i|\mathbf{x}_i)$, except that α^{-1} is replaced everywhere by $\alpha^{-1}\mu_i^{2-p}$. The negative binomial model NEGBIN1, which sets $p = 1$, has the variance function $V(y_i|\mathbf{x}_i) = \mu_i + \alpha\mu_i$, which is linear in the mean. To estimate this model, specify `DIST=NEGBIN(P=1)` in the `MODEL` statement.

The log-likelihood function of the NEGBIN1 regression model is given by

$$\mathcal{L} = \sum_{i=1}^N \left\{ \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1} \exp(\mathbf{x}_i' \boldsymbol{\beta})) - \ln(y_i!) - (y_i + \alpha^{-1} \exp(\mathbf{x}_i' \boldsymbol{\beta})) \ln(1 + \alpha) + y_i \ln(\alpha) \right\}$$

For more information about the negative binomial regression model, see *SAS/ETS User's Guide*.

Zero-Inflated Count Regression Overview

The main motivation for using zero-inflated count models is that real-life data frequently display overdispersion and excess zeros. Zero-inflated count models provide a way to both model the excess zeros and allow for overdispersion. In particular, there are two possible data generation processes for each observation. The result of a Bernoulli trial is used to determine which of the two processes to use. For observation i , Process 1 is chosen with probability φ_i and Process 2 with probability $1 - \varphi_i$. Process 1 generates only zero counts. Process 2 generates counts from either a Poisson or a negative binomial model. In general,

$$y_i \sim \begin{cases} 0 & \text{with probability } \varphi_i \\ g(y_i) & \text{with probability } 1 - \varphi_i \end{cases}$$

Therefore, the probability of $\{Y_i = y_i\}$ can be described as

$$\begin{aligned} P(y_i = 0|\mathbf{x}_i) &= \varphi_i + (1 - \varphi_i)g(0) \\ P(y_i|\mathbf{x}_i) &= (1 - \varphi_i)g(y_i), \quad y_i > 0 \end{aligned}$$

where $g(y_i)$ follows either the Poisson or the negative binomial distribution.

If the probability φ_i depends on the characteristics of observation i , then φ_i is written as a function of $\mathbf{z}_i' \boldsymbol{\gamma}$, where $\mathbf{z}_i'$ is the $1 \times (q + 1)$ vector of zero-inflated covariates and $\boldsymbol{\gamma}$ is the $(q + 1) \times 1$ vector of zero-inflated coefficients to be estimated. (The zero-inflated intercept is γ_0 ; the coefficients for the q zero-inflated covariates are $\gamma_1, \dots, \gamma_q$.) The function F that relates the product $\mathbf{z}_i' \boldsymbol{\gamma}$ (which is a scalar) to the probability φ_i is called the zero-inflated link function,

$$\varphi_i = F_i = F(\mathbf{z}_i' \boldsymbol{\gamma})$$

In the CNTSELECT procedure, the zero-inflated covariates are indicated in the `ZEROMODEL` statement. Furthermore, the zero-inflated link function F can be specified as either the logistic function,

$$F(\mathbf{z}_i' \boldsymbol{\gamma}) = \Lambda(\mathbf{z}_i' \boldsymbol{\gamma}) = \frac{\exp(\mathbf{z}_i' \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})}$$

or the standard normal cumulative distribution function (also called the probit function),

$$F(\mathbf{z}'_i \boldsymbol{\gamma}) = \Phi(\mathbf{z}'_i \boldsymbol{\gamma}) = \int_0^{\mathbf{z}'_i \boldsymbol{\gamma}} \frac{1}{\sqrt{2\pi}} \exp(-u^2/2) du$$

The zero-inflated link function is indicated by using the LINK= option in the ZEROMODEL statement. The default ZI link function is the logistic function.

Zero-Inflated Poisson Regression

In the zero-inflated Poisson (ZIP) regression model, the data generation process that is referred to earlier as Process 2 is

$$g(y_i) = \frac{\exp(-\mu_i) \mu_i^{y_i}}{y_i!}$$

where $\mu_i = e^{\mathbf{x}'_i \boldsymbol{\beta}}$. Thus the ZIP model is defined as

$$\begin{aligned} P(y_i = 0 | \mathbf{x}_i, \mathbf{z}_i) &= F_i + (1 - F_i) \exp(-\mu_i) \\ P(y_i | \mathbf{x}_i, \mathbf{z}_i) &= (1 - F_i) \frac{\exp(-\mu_i) \mu_i^{y_i}}{y_i!}, \quad y_i > 0 \end{aligned}$$

The conditional expectation and conditional variance of y_i are given by

$$\begin{aligned} E(y_i | \mathbf{x}_i, \mathbf{z}_i) &= \mu_i (1 - F_i) \\ V(y_i | \mathbf{x}_i, \mathbf{z}_i) &= E(y_i | \mathbf{x}_i, \mathbf{z}_i) (1 + \mu_i F_i) \end{aligned}$$

Note that the ZIP model (in addition to the ZINB model) exhibits overdispersion because $V(y_i | \mathbf{x}_i, \mathbf{z}_i) > E(y_i | \mathbf{x}_i, \mathbf{z}_i)$.

In general, the log-likelihood function of the ZIP model is

$$\mathcal{L} = \sum_{i=1}^N \ln [P(y_i | \mathbf{x}_i, \mathbf{z}_i)]$$

After a specific link function (either logistic or standard normal) for the probability φ_i is chosen, it is possible to write the exact expressions for the log-likelihood function and the gradient.

ZIP Model with Logistic Link Function

First, consider the ZIP model in which the probability φ_i is expressed by a logistic link function, namely

$$\varphi_i = \frac{\exp(\mathbf{z}'_i \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}'_i \boldsymbol{\gamma})}$$

The log-likelihood function is

$$\begin{aligned}\mathcal{L} = & \sum_{\{i:y_i=0\}} \ln [\exp(\mathbf{z}_i' \boldsymbol{\gamma}) + \exp(-\exp(\mathbf{x}_i' \boldsymbol{\beta}))] \\ & + \sum_{\{i:y_i>0\}} \left[y_i \mathbf{x}_i' \boldsymbol{\beta} - \exp(\mathbf{x}_i' \boldsymbol{\beta}) - \sum_{k=2}^{y_i} \ln(k) \right] \\ & - \sum_{i=1}^N \ln [1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})]\end{aligned}$$

ZIP Model with Standard Normal Link Function

Next, consider the ZIP model in which the probability φ_i is expressed by a standard normal link function: $\varphi_i = \Phi(\mathbf{z}_i' \boldsymbol{\gamma})$. The log-likelihood function is

$$\begin{aligned}\mathcal{L} = & \sum_{\{i:y_i=0\}} \ln \{ \Phi(\mathbf{z}_i' \boldsymbol{\gamma}) + [1 - \Phi(\mathbf{z}_i' \boldsymbol{\gamma})] \exp(-\exp(\mathbf{x}_i' \boldsymbol{\beta})) \} \\ & + \sum_{\{i:y_i>0\}} \left\{ \ln [(1 - \Phi(\mathbf{z}_i' \boldsymbol{\gamma}))] - \exp(\mathbf{x}_i' \boldsymbol{\beta}) + y_i \mathbf{x}_i' \boldsymbol{\beta} - \sum_{k=2}^{y_i} \ln(k) \right\}\end{aligned}$$

For more information about the zero-inflated Poisson regression model, see *SAS/ETS User's Guide*.

Zero-Inflated Conway-Maxwell-Poisson Regression

In the Conway-Maxwell-Poisson regression model, the data generation process is defined as

$$P(Y_i = y_i | \mathbf{x}_i, \mathbf{z}_i) = \frac{1}{Z(\lambda_i, v_i)} \frac{\lambda_i^{y_i}}{(y_i!)^{v_i}}, \quad y_i = 0, 1, 2, \dots$$

where the normalization factor is

$$Z(\lambda_i, v_i) = \sum_{n=0}^{\infty} \frac{\lambda_i^n}{(n!)^{v_i}}$$

and

$$\lambda_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

$$v_i = \exp(-\mathbf{g}_i' \boldsymbol{\delta})$$

The zero-inflated Conway-Maxwell-Poisson model can be written as

$$\begin{aligned}P(y_i | \mathbf{x}_i, \mathbf{z}_i) &= F_i + (1 - F_i) \frac{1}{Z(\lambda_i, v_i)}, \quad y_i = 0 \\ P(y_i | \mathbf{x}_i, \mathbf{z}_i) &= (1 - F_i) \frac{1}{Z(\lambda_i, v_i)} \frac{\lambda_i^{y_i}}{(y_i!)^{v_i}}, \quad y_i > 0\end{aligned}$$

The conditional expectation and conditional variance of y_i are given respectively by

$$E(y_i | \mathbf{x}_i, \mathbf{z}_i) = (1 - F_i) \frac{1}{Z(\lambda, \nu)} \sum_{j=0}^{\infty} \frac{j \lambda^j}{(j!)^\nu}$$

$$V(y_i | \mathbf{x}_i, \mathbf{z}_i) = (1 - F_i) \frac{1}{Z(\lambda, \nu)} \sum_{j=0}^{\infty} \frac{j^2 \lambda^j}{(j!)^\nu} - E(y_i | \mathbf{x}_i, \mathbf{z}_i)^2$$

The general form of the log-likelihood function for the Conway-Maxwell-Poisson zero-inflated model is

$$\mathcal{L} = \sum_{i=1}^N w_i \ln [P(y_i | \mathbf{x}_i, \mathbf{z}_i)]$$

Zero-Inflated Conway-Maxwell-Poisson Model with Logistic Link Function

For this model, the probability φ_i is expressed by using a logistic link function as

$$\varphi_i = \Lambda(\mathbf{z}_i' \boldsymbol{\gamma}) = \frac{\exp(\mathbf{z}_i' \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})}$$

The log-likelihood function is

$$\begin{aligned} \mathcal{L} = & \sum_{\{i: y_i=0\}} w_i \ln \left\{ \Lambda(\mathbf{z}_i' \boldsymbol{\gamma}) + [1 - \Lambda(\mathbf{z}_i' \boldsymbol{\gamma})] \frac{1}{Z(\lambda_i, \nu_i)} \right\} \\ & + \sum_{\{i: y_i>0\}} w_i \{ \ln [(1 - \Lambda(\mathbf{z}_i' \boldsymbol{\gamma}))] - \ln(Z(\lambda, \nu)) + (y_i \ln(\lambda) - \nu \ln(y_i!)) \} \end{aligned}$$

Zero-Inflated Conway-Maxwell-Poisson Model with Normal Link Function

For this model, the probability φ_i is specified by using the standard normal distribution function (probit function): $\varphi_i = \Phi(\mathbf{z}_i' \boldsymbol{\gamma})$.

The log-likelihood function is written as

$$\begin{aligned} \mathcal{L} = & \sum_{\{i: y_i=0\}} w_i \ln \left\{ \Phi(\mathbf{z}_i' \boldsymbol{\gamma}) + [1 - \Phi(\mathbf{z}_i' \boldsymbol{\gamma})] \frac{1}{Z(\lambda_i, \nu_i)} \right\} \\ & + \sum_{\{i: y_i>0\}} w_i \{ \ln [(1 - \Phi(\mathbf{z}_i' \boldsymbol{\gamma}))] - \ln(Z(\lambda, \nu)) + (y_i \ln(\lambda) - \nu \ln(y_i!)) \} \end{aligned}$$

Zero-Inflated Negative Binomial Regression

The zero-inflated negative binomial (ZINB) model in PROC CNTSELECT is based on the negative binomial model that has a quadratic variance function (when DIST=NEGBIN in the MODEL or PROC CNTSELECT statement). The ZINB model is obtained by specifying a negative binomial distribution for the data generation process referred to earlier as Process 2:

$$g(y_i) = \frac{\Gamma(y_i + \alpha^{-1})}{y_i! \Gamma(\alpha^{-1})} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \left(\frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}$$

Thus the ZINB model is defined to be

$$\begin{aligned} P(y_i = 0 | \mathbf{x}_i, \mathbf{z}_i) &= F_i + (1 - F_i) (1 + \alpha \mu_i)^{-\alpha^{-1}} \\ P(y_i | \mathbf{x}_i, \mathbf{z}_i) &= (1 - F_i) \frac{\Gamma(y_i + \alpha^{-1})}{y_i! \Gamma(\alpha^{-1})} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \\ &\quad \times \left(\frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}, \quad y_i > 0 \end{aligned}$$

In this case, the conditional expectation (E) and conditional variance (V) of y_i are

$$E(y_i | \mathbf{x}_i, \mathbf{z}_i) = \mu_i (1 - F_i)$$

$$V(y_i | \mathbf{x}_i, \mathbf{z}_i) = E(y_i | \mathbf{x}_i, \mathbf{z}_i) [1 + \mu_i (F_i + \alpha)]$$

Like the ZIP model, the ZINB model exhibits overdispersion because the conditional variance exceeds the conditional mean.

ZINB Model with Logistic Link Function

In this model, the probability φ_i is given by the logistic function, namely

$$\varphi_i = \frac{\exp(\mathbf{z}_i' \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})}$$

The log-likelihood function is

$$\begin{aligned} \mathcal{L} &= \sum_{\{i: y_i=0\}} \ln \left[\exp(\mathbf{z}_i' \boldsymbol{\gamma}) + (1 + \alpha \exp(\mathbf{x}_i' \boldsymbol{\beta}))^{-\alpha^{-1}} \right] \\ &+ \sum_{\{i: y_i>0\}} \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1}) \\ &+ \sum_{\{i: y_i>0\}} \{ -\ln(y_i!) - (y_i + \alpha^{-1}) \ln(1 + \alpha \exp(\mathbf{x}_i' \boldsymbol{\beta})) + y_i \ln(\alpha) + y_i \mathbf{x}_i' \boldsymbol{\beta} \} \\ &- \sum_{i=1}^N \ln [1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})] \end{aligned}$$

ZINB Model with Standard Normal Link Function

For this model, the probability φ_i is expressed by the standard normal distribution function (probit function): $\varphi_i = \Phi(\mathbf{z}'_i \boldsymbol{\gamma})$. The log-likelihood function is

$$\begin{aligned} \mathcal{L} = & \sum_{\{i: y_i=0\}} \ln \left\{ \Phi(\mathbf{z}'_i \boldsymbol{\gamma}) + [1 - \Phi(\mathbf{z}'_i \boldsymbol{\gamma})] (1 + \alpha \exp(\mathbf{x}'_i \boldsymbol{\beta}))^{-\alpha^{-1}} \right\} \\ & + \sum_{\{i: y_i>0\}} \ln [1 - \Phi(\mathbf{z}'_i \boldsymbol{\gamma})] \\ & + \sum_{\{i: y_i>0\}} \sum_{j=0}^{y_i-1} \{ \ln(j + \alpha^{-1}) \} \\ & - \sum_{\{i: y_i>0\}} \ln(y_i!) \\ & - \sum_{\{i: y_i>0\}} (y_i + \alpha^{-1}) \ln(1 + \alpha \exp(\mathbf{x}'_i \boldsymbol{\beta})) \\ & + \sum_{\{i: y_i>0\}} y_i \ln(\alpha) \\ & + \sum_{\{i: y_i>0\}} y_i \mathbf{x}'_i \boldsymbol{\beta} \end{aligned}$$

For more information about the zero-inflated negative binomial regression model, see *SAS/ETS User's Guide*.

Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, INIT, and PRIOR Statements

This section describes how you can refer to the parameters in the MODEL, ZEROMODEL, and DISPMODEL statements when you use the RESTRICT, TEST, BOUNDS, INIT, or PRIOR statement. The following examples use the RESTRICT statement, but the same remarks apply to naming parameters when you use the TEST, BOUNDS, INIT, or PRIOR statement. The names of the parameters can be seen in the “Parameter Estimates” table if you specify the option. If a parameter name contains a blank or some other special character (such as “*”, “-”, “(”, or “)”), then you must use the internal name of the parameter in order to refer to that parameter in the RESTRICT, TEST, BOUNDS, INIT, or PRIOR statement. You can specify the PRINTINTERNALNAMES option in the PROC CNTSELECT statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column is added to the “Parameter Estimates” table, showing the internal name of each parameter. Alternatively, the SHOWNAMES suboption of the PRIORSUMMARY option displays the internal names that are used for all parameters in the prior summary table.

To impose a restriction on a parameter that is related to a regressor in the MODEL statement, you simply use the name of the regressor itself to refer to its associated parameter. Suppose your model is

```
model y = x1 x2 x5;
```

where x1 through x5 are continuous variables. If you want to restrict the parameter associated with the regressor x5 to be greater than 1.7, then you should use the following statement:

```
RESTRICT x5 > 1.7;
```

To impose a restriction on a parameter associated with a regressor in the ZEROMODEL statement, you can form the name of the parameter by prefixing `Inf_` to the name of the regressor. Suppose your MODEL and ZEROMODEL statements are as follows:

```
model y = x1 x2 x5;
zeromodel y ~ x3 x5;
```

If you want to restrict the parameter related to the x5 regressor in the ZEROMODEL statement to be less than 1.0, then you refer to the parameter as `Inf_x5` and provide the following statement:

```
RESTRICT Inf_x5 < 1.0;
```

Even though the regressor x5 appears in both the MODEL and ZEROMODEL statements, the parameter associated with x5 in the MODEL statement is, of course, different from the parameter associated with x5 in the ZEROMODEL statement. Thus, when the name of a regressor is used in a RESTRICT statement without any prefix, it refers to the parameter associated with that regressor in the MODEL statement. Meanwhile, when the name of a regressor is used in a RESTRICT statement with the prefix `Inf_`, it refers to the parameter associated with that regressor in the ZEROMODEL statement. The parameter associated with the intercept in the ZEROMODEL is named `Inf_Intercept`.

In a similar way, you can form the name of a parameter associated with a regressor in the DISPMODEL statement by prefixing `Dsp_` to the name of the regressor. The parameter associated with the intercept in the DISPMODEL is named `Dsp_Intercept`.

Referring to Class-Level Parameters

When your MODEL includes a classification variable, you can impose restrictions on the parameters associated with each of the levels that are related to the classification variable as follows.

Suppose your classification variable is named `C` and it has three levels: 0, 1, 2. Suppose your model is the following:

```
class C;
model y = x1 x2 C;
```

Adding a classification variable as a regressor to your model introduces additional parameters into your model, each of which is associated with one of the levels of the classification variable. You can form the name of the parameter associated with a particular level of your class variable by inserting the underscore character between the name of the classification variable and the value of the level. Thus, to restrict the parameter associated with level 0 of the classification variable `C` to always be greater than 0.7, you refer to the parameter as `C_0` and provide the following statement:

```
RESTRICT C_0 > 0.7;
```

Referring to Parameters Associated with Interactions between Regressors

When a regressor in your model involves an interaction between other regressors, you can impose restrictions on the parameters associated with the interaction.

Suppose you have the following model:

```
model y = x1 x2 x3*x4;
```

You can form the name of the parameter that is associated with the interaction regressor $x3*x4$ by adding underscores between the variable names and the multiplication sign and by replacing the multiplication sign with an underscore. Thus, $x3_x4$ refers to the parameter that is associated with the interaction regressor $x3*x4$.

Referring to interactions between regressors and classification variables is handled in the same way. Suppose you have a classification variable that is named *C* and has three levels: 0, 1, 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 C*x3;
```

The interaction between the continuous variable *x3* and the classification variable *C* introduces three additional parameters, which are named $x3_C_0$, $x3_C_1$, and $x3_C_2$. Note how, although the order of the terms in the interaction is *C* followed by *x3*, the name of the parameter associated with the interaction is formed by placing the name of the continuous variable *x3* first, followed by three underscores, followed by the name of the classification variable *C*, then an underscore, and then the level value. Once again, depending on the parameterization that you specify in your CLASS statement, for each interaction in your model that involves a classification variable, one of the parameters associated with that interaction might be dropped from your model prior to optimization.

The name of a parameter that is associated with a nested interaction is formed in a slightly different way. Suppose you have a classification variable that is named *C* and has three levels: 0, 1, 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 x3(C);
```

The nested interaction between the continuous variable *x3* and the classification variable *C* introduces three additional parameters, which are named $x3_C_0$, $x3_C_1$, and $x3_C_2$. Note how the name in each case is formed from the name of the regressor by first inserting underscores between each variable name and the left and right parentheses, then by replacing the left and right parentheses with underscores.

Referring to Class Level Parameters with Negative Values

When the value of a level is a negative number, you must replace the minus sign with an 'N' when you form the name of the parameter that is associated with that particular level of the classification variable. For example, suppose that your classification variable is named *D* and has four levels: -1, 0, 1, 2. Suppose that your model is the following:

```
class D;
model y = x1 x2 D;
```

To restrict the parameter that is associated with level -1 of the classification variable *D* to always be less than 0.4, you refer to the parameter as D_N1 and provide the following statement:

```
RESTRICT D_N1 < 0.4;
```

Dropping a Class Level Parameter to Avoid Collinearity

Depending on the parameterization you impose on your classification variable, one of the parameters associated with its levels might be dropped from your model prior to optimization in order to avoid collinearity. For example, when the default parameterization GLM is imposed, the parameter that is associated with the last level of your classification variable is dropped prior to optimization. If you attempt to impose a restriction on a dropped parameter by using the RESTRICT statement, PROC CNTSELECT issues an error message in the log.

For example, suppose again that your classification variable is named C and that it has three levels: 0, 1, 2. Suppose your model is the following:

```
class C;
model y = x1 x2 C;
```

Because no additional options are specified in the CLASS statement, GLM parameterization is assumed. This means that the parameter named C_2 (which is the parameter associated with the last level of your classification variable) will be dropped from your model before the optimizer is invoked. Therefore, an error will be issued if you attempt to restrict the C_2 parameter in any way by referring to it in a RESTRICT statement. For example, the following RESTRICT statement will generate an error:

```
RESTRICT C_2 < 0.3;
```

Referring to Implicit Parameters

Unless you specified the NOINT option, you can impose a restriction on the parameter associated with the Intercept in your model simply by referring to it by name: Intercept.

For certain model types, one or more implicit parameters will be added to your model prior to optimization. You can impose restrictions on these implicit parameters.

For the Poisson model for which ERRORCOMP=RANDOM is specified, PROC CNTSELECT automatically adds the _Alpha parameter to your model.

If no ERRORCOMP= option is specified, for zero-inflated binomial and negative binomial models, PROC CNTSELECT adds the _Alpha parameter to the model. If ERRORCOMP=RANDOM is specified for the zero-inflated binomial and negative binomial models, then PROC CNTSELECT adds two implicit parameters to the model: _Alpha and _Beta.

For Conway-Maxwell Poisson models that do not include a DISPMODEL statement, the _lnNu parameter is added to the model.

Whenever your model type dictates the addition of one or more of these implicit parameters, you can impose restrictions on the implicit parameters by referring to them by name in a RESTRICT statement. For example, if your model type implies the existence of the _Alpha parameter, you can restrict _Alpha to be greater than 0.2 as follows:

```
RESTRICT _Alpha > 0.2;
```

Computational Resources

The time and memory that PROC CNTSELECT requires are proportional to the number of parameters in the model and the number of observations in the data table being analyzed. Less time and memory are required for smaller models and fewer observations. When PROC CNTSELECT is run in the high-performance distributed environment, the amount of time required is also affected by the number of nodes and the number of threads per node as specified in the PERFORMANCE statement.

The method that is chosen to calculate the variance-covariance matrix and the optimization method also affect the time and memory resources. All optimization methods available through the METHOD= option have similar memory use requirements. The processing time might differ for each method, depending on the number of iterations and functional calls needed. The data table is read into memory to save processing time. If not enough memory is available to hold the data, the CNTSELECT procedure stores the data in a utility file on disk and rereads the data as needed from this file, substantially increasing the execution time of the procedure. The gradient and the variance-covariance matrix must be held in memory. If the model has p parameters including the intercept, then at least $8 * (p + p * (p + 1)/2)$ bytes of memory are needed. The processing time is also a function of the number of iterations needed to converge to a solution for the model parameters. The number of iterations that are needed cannot be known in advance. You can use the MAXITER= option to limit the number of iterations that PROC CNTSELECT executes. You can alter the convergence criteria by using the nonlinear optimization options available in the PROC CNTSELECT statement. For a list of all the nonlinear optimization options, see “[Optimization Control Options](#)” on page 387.

Covariance Matrix Types

The COVEST= option in the PROC CNTSELECT statement enables you to specify the estimation method for the covariance matrix. COVEST=HESSIAN estimates the covariance matrix that is based on the inverse of the Hessian matrix; COVEST=OP uses the outer product of gradients; and COVEST=QML produces the covariance matrix that is based on both the Hessian and outer product matrices. Although all three methods produce asymptotically equivalent results, they differ in computational intensity and produce results that might differ in finite samples. The COVEST=OP option provides the covariance matrix that is typically the easiest to compute. In some cases, the OP approximation is considered more efficient than the Hessian or QML approximation because it contains fewer random elements. The QML approximation is computationally the most complex because it requires both the outer product of gradients and the Hessian matrix. In most cases, the OP or Hessian approximation is preferred to QML. The need for QML approximation arises in cases where the model is misspecified and the information matrix equality does not hold. The default is COVEST=HESSIAN.

Tests on Parameters

In general, you can write the tested hypothesis as

$$H_0 : \mathbf{h}(\theta) = 0$$

where $\mathbf{h}(\theta)$ is an $r \times 1$ vector-valued function of the parameters θ that is given by the r expressions that you specify in the TEST statement.

Let $\hat{V}$ be the estimate of the covariance matrix of $\hat{\theta}$. Let $\hat{\theta}$ be the unconstrained estimate of θ , and let $\tilde{\theta}$ be the constrained estimate of θ such that $\mathbf{h}(\tilde{\theta}) = 0$. Let

$$A(\theta) = \partial \mathbf{h}(\theta) / \partial \theta \big|_{\hat{\theta}}$$

Using this notation, you compute the test statistics for the three types of tests as follows:

- The Wald test statistic is defined as

$$W = \mathbf{h}'(\hat{\theta}) \left(A(\hat{\theta}) \hat{V} A'(\hat{\theta}) \right)^{-1} \mathbf{h}(\hat{\theta})$$

The Wald test is not invariant to reparameterization of the model (Gregory and Veall 1985; Gallant 1987, p. 219). For more information about the theoretical properties of the Wald test, see Phillips and Park (1988).

- The Lagrange multiplier test statistic is

$$LM = \lambda' A(\tilde{\theta}) \tilde{V} A'(\tilde{\theta}) \lambda$$

where λ is the vector of Lagrange multipliers from the computation of the restricted estimate $\tilde{\theta}$.

- The likelihood ratio test statistic is

$$LR = 2 \left(L(\hat{\theta}) - L(\tilde{\theta}) \right)$$

where $\tilde{\theta}$ represents the constrained estimate of θ and L is the concentrated log-likelihood value.

For each type of test, under the null hypothesis the test statistic is asymptotically distributed as a χ^2 random variable with r degrees of freedom, where r is the number of expressions in the TEST statement. The p -values that are reported for the tests are computed from the $\chi^2(r)$ distribution and are only asymptotically valid.

Monte Carlo simulations suggest that the asymptotic distribution of the Wald test is a poorer approximation of its small sample distribution than that of the other two tests. However, the Wald test has the lowest computational cost, because it does not require computation of the constrained estimate $\tilde{\theta}$.

The following statements use the TEST statement to perform a likelihood ratio test:


```
proc cntselect;
  model y = x1 x2 x3;
  test x1 = 0, x2 * .5 + 2 * x3 = 0 /lr;
run;
```

For more information, see *SAS/ETS User's Guide*.

Bayesian Analysis

To perform Bayesian analysis, you must specify a BAYES statement. Unless otherwise stated, all options in this section are options in the BAYES statement.

Bayes' theorem states that the posterior density $\pi(\theta|y)$ can be obtained via

$$\pi(\theta|y) \propto \pi(\theta)L(y|\theta)$$

where θ is a parameter or a vector of parameters and $\pi(\theta)$ is the product of the prior densities that you specify in the **PRIOR** statement. The term $L(y|\theta)$ is the likelihood associated with the model that is implied by the **MODEL** statement. For an introduction to Bayesian analysis, see the section “[Overview](#)” on page 5 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).”

To perform Bayesian analysis, you must draw a random sample from the posterior density $\pi(\theta|y)$ by using the Markov chain Monte Carlo (MCMC) method. For an introduction to MCMC, see the section “[Markov Chain Monte Carlo Method](#)” on page 16 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, PROC CNTSELECT uses the random walk Metropolis (RWM) algorithm to obtain the posterior sample. For an overview of Metropolis algorithms, see the section “[Metropolis and Metropolis-Hastings Algorithms](#)” on page 17 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” The RWM algorithm requires a proposal distribution to be tuned to match certain features of the posterior distribution. For more information about this tuning process, see the section “[Tuning Random Walk Metropolis Algorithms](#)” on page 19 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).”

Alternatively, PROC CNTSELECT can use the No-U-Turn Sampler (NUTS) algorithm, a variant of the Hamiltonian Monte Carlo (HMC) algorithm, to obtain the posterior sample. For an overview of HMC and NUTS, see the sections “[Hamiltonian Monte Carlo Sampler](#)” on page 21 and “[No-U-Turn Sampler](#)” on page 22 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” NUTS requires an integrator step size parameter and a mass matrix parameter to be tuned to match certain features of the posterior distribution. For more information about this tuning process, see the section “[Tuning for NUTS](#)” on page 23 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).”

Additional details specific to PROC CNTSELECT follow.

Tuning the Random Walk Metropolis Sampler

PROC CNTSELECT tunes the random walk Metropolis proposal distribution during a tuning phase that consists of n_{stage} tuning stages, each n_{tune} iterations long. For more information, see the section “[Tuning Random Walk Metropolis Algorithms](#)” on page 19 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” The random walk Metropolis proposal covariance matrix that is defined in that section, COV^{cur} , should approximate the posterior covariance matrix and must be initialized prior to tuning. PROC CNTSELECT always initializes it to the covariance matrix of the maximum likelihood estimates of the model parameters.

Tuning the No-U-Turn Sampler

PROC CNTSELECT tunes both the integrator step size and the mass matrix in a tuning phase that consists of n_{tune} iterations, divided into a sequence of tuning windows. For more information, see the section “[Tuning for NUTS](#)” on page 23 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” The step size is tuned throughout the entire tuning phase in order to achieve a desired Metropolis acceptance rate. The mass matrix is diagonal, and the ideal mass matrix is the diagonal matrix of inverse posterior variances. These posterior variances are learned in the tuning phase. PROC CNTSELECT always initializes the mass matrix to the diagonal matrix of the inverse square standard errors of the maximum likelihood parameter estimates.

Initial Values of the Markov Chain

PROC CNTSELECT starts each Markov chain at the maximum likelihood estimates (MLEs) of the parameters. If the MLE for a particular parameter is outside the support of its prior distribution, then PROC CNTSELECT instead initializes that parameter near the boundary of its prior support.

Bayesian Analysis with Weighted Likelihoods

In PROC CNTSELECT, you can account for sample weights in Bayesian analysis by using the **WEIGHT** statement, provided that the weights are handled properly. The weighted log-likelihood functions in PROC CNTSELECT have the form

$$\ell(\theta) = \sum_{i=1}^n w_i \ell_i(\theta)$$

where $\ell_i(\theta)$ is the log likelihood for observation i , w_i is the corresponding sample weight, and n is the sample size. When $w_i = 1$ for all i , this is equivalent to a nonweighted log likelihood. In this case the weights sum to the sample size.

In the general weighted case, the sum of the weights should still equal the sample size—that is, $\sum_{i=1}^n w_i = n$ —and w_i/n should represent the proportion of the sample size that corresponds to observation i . This ensures that the prior density and likelihood function are weighted correctly in the posterior density. For more information, see Savitsky and Toth (2016).

PROC CNTSELECT normalizes the weights to sum to the sample size by default, so weighted Bayesian analyses are valid. The **NONNORMALIZE** option in the **WEIGHT** statement tells PROC CNTSELECT not to normalize the weights, so it can result in misleading weighted Bayesian analyses and is generally not recommended.

The **FREQ** statement is similar to the **WEIGHT** statement, except that each frequency represents the sample size associated with the corresponding observation. By definition, the sum of the frequencies is equal to the sample size. So you can combine the **BAYES** statement with the **FREQ** statement without issue.

Distributed Data and MCMC

When you use a CAS server that is running in massively parallel processing (MPP) mode, the data are distributed across the workers of the server. Iterative methods such as MCMC can be significantly slower in this context because of communication costs. Additionally, when you request multiple Markov chains by using the **NCHAIN=** option, those chains are independent of each other and can be run in parallel instead of serially. When you perform Bayesian analysis by using the **BAYES** statement on a server in MPP mode, PROC CNTSELECT automatically tries to take advantage of the distributed environment to make MCMC

run as fast as possible. To do this, it automatically copies the data onto each worker machine and divides the number of requested Markov chains among the workers as evenly as possible. Then those workers run their chains in parallel and send their results back to the controller. MCMC is therefore always as slow as the slowest worker—typically one of the workers that have the most Markov chains if the chains cannot be evenly divided among the workers. The fastest possible CAS server configuration has at least as many workers as the number of Markov chains that you request, though there is no benefit to having more workers than chains.

Although copying the data to each of the workers and letting them run their own Markov chains in parallel is fast, it requires the data set to be small enough to fit on each worker machine. You can use the [DISTRIBUTED](#) option to instead keep the data distributed at the cost of a much slower analysis. The [DISTRIBUTED](#) option also enables you to obtain results that are numerically identical to those of PROC CNTSELECT in release 2022.11 and previous releases, which do not automatically copy the data set onto the workers for Bayesian analysis. If the CAS server is running in symmetric multiprocessing (SMP) mode, then the server has only a single machine and the [DISTRIBUTED](#) option does nothing.

Prior Distributions

The PRIOR statement is used to specify the prior distribution of the model parameters. You must specify a list of parameters, a tilde (~), and then a distribution with its parameters. You can specify multiple [PRIOR](#) statements to define independent prior distributions. Parameters that are associated with a regressor variable are referred to by the name of the corresponding regressor variable. For information about how parameters are named so that you can refer to them when specifying prior distributions, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, INIT, and PRIOR Statements](#)” on page 433. Alternatively, the SHOWNAMES suboption of the [PRIORSUMMARY](#) option displays the names that are used in all parameters in the prior summary table.

If a parameter is not associated with a PRIOR statement, then the default choices shown in [Table 10.4](#) are used.

Table 10.4 Default Values for Prior Distributions

Parameter Type	Default Prior Name	Default Prior Specification
Regression, location	Normal	NORMAL(MEAN=0, VAR=1E6)
Scale	Gamma	GAMMA(SHAPE=2, SCALE=100)

For more information about each prior density, see the section “[Standard Prior Distributions](#)” on page 12 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” For information about truncated densities, see the section “[Truncated Prior Distributions](#)” on page 442. For information about why the default prior on the standard deviation parameter is a truncated normal distribution, see the section “[Prior Distributions for Scale Parameters](#)” on page 442.

Prior Distributions for Scale Parameters

A common choice for a prior distribution on variance parameters and other scale parameters is the inverse gamma distribution. This prior can often be problematic in practice, and as a consequence it can even cause computational problems. This is partially because the density puts essentially no probability mass near zero. For more information, see Gelman (2006). The main advantage of the inverse gamma prior is that it is the conditionally conjugate prior for a variance parameter in a normal model; this makes certain sampling algorithms easier to use. For more information, see the section “[Conjugate Priors](#)” on page 9 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” However, the posterior sampler in PROC CNTSELECT does not take advantage of this conjugacy, so there is no computational benefit to using the inverse gamma prior.

Instead, PROC CNTSELECT uses the gamma for the default prior on scale parameters. This density allows for the scale parameter to be estimated arbitrarily close to zero and otherwise removes the problems associated with the inverse gamma prior. In general, if you want to specify prior information on a scale parameter, it is recommended that you encode that information into a gamma distribution, truncated normal distribution, or truncated t distribution.

Truncated Prior Distributions

All prior distributions except the uniform distribution can be turned into truncated prior distributions by using the LOWER and UPPER parameters. By default, LOWER is set to the maximum of the parameter’s lower bound and the specified prior distribution’s lower bound, and UPPER is set to the minimum of the parameter’s upper bound and the specified prior distribution’s upper bound. For example, for `_Alpha`, the dispersion parameter in a negative binomial model, the following statement specifies a truncated normal prior on `_Alpha` with LOWER=0 and UPPER=INFINITY:

```
prior _Alpha ~ normal(mean = 0, var = 1);
```

The density of any truncated prior can be computed from the corresponding untruncated density given in the section “[Standard Prior Distributions](#)” on page 12 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#),” as follows. Suppose $\pi(\theta)$ is the prior density for the parameter θ . Then let $\Pi(\theta)$ denote the corresponding CDF, defined by the integral

$$\Pi(\theta) = \int_{-\infty}^{\theta} \pi(x) dx$$

where $\pi(x)$ is taken to be 0 outside its range (for each prior density’s range, see the section “[Standard Prior Distributions](#)” on page 12 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#)”). Then the truncated density is constructed from the untruncated density by using the formula

$$\pi_{L,U}(\theta) = \frac{\pi(\theta)}{\Pi(U) - \Pi(L)}$$

where $L < U$ are the lower and upper truncation bounds, respectively. If the lower truncation bound is the lower bound of π ’s range, then $\Pi(L) = 0$, and similarly if the upper truncation bound is the upper bound of π ’s range, then $\Pi(U) = 1$.

Displayed Output

PROC CNTSELECT produces the following displayed output.

Model Fit Summary

The “Model Fit Summary” table contains the following information:

- dependent (count) variable name
- number of observations used
- number of missing values in data table, if any
- data table name
- type of model that was fit
- error components model type for panel data
- group id variable for panel data
- number of cross sections for panel data
- parameterization for the Conway-Maxwell-Poisson model
- frequency variable name, if any
- weight variable name, if any
- offset variable name, if any
- zero-inflated link function, if any
- zero-inflated offset variable name, if any
- log-likelihood value at solution
- maximum absolute gradient at solution
- number of iterations
- optimization technique used
- AIC value at solution (smaller value indicates better fit)
- SBC value at solution (smaller value indicates better fit)
- covariance matrix estimation method used

A line below the “Model Fit Summary” table indicates whether the algorithm successfully converged.

Parameter Estimates

The “Parameter Estimates” table gives the estimates of the model parameters. In zero-inflated (ZI) models, estimates are also given for the ZI intercept and ZI regressor parameters, which are labeled with the prefix “Inf_”. For example, the ZI intercept is labeled “Inf_intercept”. If you specify “Age” as a ZI regressor, then the “Parameter Estimates” table labels the corresponding parameter estimate “Inf_Age”. If you do not list any ZI regressors, then only the ZI intercept term is estimated.

If the DISPMODEL statement is specified for the Conway-Maxwell-Poisson model, the estimates are given for the dispersion intercept, and parameters are labeled with the prefix “Dsp_”. For example, the dispersion model intercept is labeled “Dsp_Intercept”. If you specify “Education” as a dispersion model regressor, then the “Parameter Estimates” table labels the corresponding parameter estimate “Dsp_Education”. If you do not list any dispersion regressors, then only the dispersion intercept is estimated.

“_Alpha” is the negative binomial dispersion parameter. The t statistic that is given for “_Alpha” is a test of overdispersion.

Covariance of Parameter Estimates

If you specify the COVB option in the PROC CNTSELECT or MODEL statement, the CNTSELECT procedure displays the estimated covariance matrix, which is defined as the inverse of the information matrix at the final iteration.

Correlation of Parameter Estimates

If you specify the CORRB option in the PROC CNTSELECT or MODEL statement, the CNTSELECT procedure displays the estimated correlation matrix, which is based on the Hessian matrix used at the final iteration.

OUTPUT OUT= Data Table

The OUTPUT statement creates a new data table that contains various estimates that you specify. You can request that the output data table contain the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the probability that the response variable will take the current value. In a zero-inflated model, you can also request that the output data table contain the estimates of $\mathbf{z}_i' \boldsymbol{\gamma}$, and the probability that the response is zero as a result of the zero-generating process. In a Conway-Maxwell-Poisson model, you can also request that the output data table contains estimates of $\mathbf{g}_i' \boldsymbol{\delta}$, λ , ν , μ , mode, variance and dispersion.

Except for the probability of the current value, these statistics can be computed for all observations in which the regressors are not missing, even if the response is missing. By adding observations with missing response values to the input data table, you can compute these statistics for new observations or for settings of the regressors that are not present in the data without affecting the model fit. Because of potential space limitations, the data table that is created by the OUTPUT statement does not contain the variables in the input data table.

ODS Table Names

PROC CNTSELECT assigns a name to each table that it creates. You can use these names to denote the table when you use the Output Delivery System (ODS) to select tables and create output data tables. These table names are listed in [Table 10.5](#).

Table 10.5 ODS Tables Produced by PROC CNTSELECT

Table Name	Description	Statement	Option
BSplineDetails	Basis details for B-spline effects	EFFECT	SPLINE with DETAILS
ClassInfo	Level information from the CLASS statement	CLASS	Default
CollectionLevelInfo	Level details for collection effects	EFFECT	COLLECTION with DETAILS
ConvergenceStatus	Convergence status	MODEL	Default
CorrB	Correlation of parameter estimates	MODEL	CORRB
CovB	Covariance of parameter estimates	MODEL	COVB
EntryCandidates	Details about candidates for entry into the model	SELECTION	DETAILS=ALL
FitModelSummary	Summary of nonlinear estimation	MODEL	Default
InitialParameterEstimates	Initial parameter estimates	PROC CNTSELECT or MODEL	PRINTALL
IterationHistory	Optimization iteration history	PROC CNTSELECT or MODEL	PRINTALL
MMLevelInfo	Level information for multimember effects	EFFECT	MULTIMEMBER with DETAILS
OptimizationResults	Optimization results	PROC CNTSELECT or MODEL	PRINTALL
ParameterEstimates	Parameter estimates	MODEL	Default
PolyDetails	Number of variables and columns, polynomial degree, and standardization method	EFFECT	POLY with DETAILS
PolyScaling	Centering and scaling details	EFFECT	POLY with DETAILS
RemovalCandidates	Details about candidates for removal from the model	SELECTION	DETAILS=ALL
SelectionSummary	Summary information about model selection steps	SELECTION	Default
SplineKnots	Knot and boundary knot values	EFFECT	SPLINE with DETAILS
StopReason	Reason for termination of model selection	SELECTION	Default
TestResults	Hypothesis test statistics	TEST	Default
TimingDetails	Detailed summary of time taken for all phases of execution	PROC CNTSELECT	PRINTTIMING=(DETAILS) or PRINTALL

Table 10.5 continued

Table Name	Description	Statement	Option
TimingSummary	Summary of time taken for main phases of execution	PROC CNTSELECT	PRINTTIMING or PRINTALL
TPFSplineDetails	Truncated power function (TPF) spline basis details	EFFECT	SPLINE with BASIS=TPF and DETAILS
ZNLOSettings	Nonlinear optimizer settings	PROC CNTSELECT or MODEL	PRINTALL
ODS Tables Created by the BAYES Statement			
AccRates	Metropolis acceptance rates	BAYES	SAMSUM
AutoCorr	Autocorrelation statistics for each parameter	BAYES	DIAG=AUTOCORR
AutoCorrByChain	Autocorrelation statistics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)= AUTOCORR
DiagSettings	Settings used for MCMC diagnostics	BAYES	DIAG(SETTINGS)=
DiagSummaries	ESS and MCSE diagnostics for each parameter	BAYES	Default
DiagSummariesByChain	ESS and MCSE diagnostics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)= SUMMARIES
Geweke	Geweke diagnostics for each parameter	BAYES	DIAG=GEWEKE
GewekeByChain	Geweke diagnostics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)= GEWEKE
Heidelberger	Heidelberger-Welch diagnostics for each parameter	BAYES	DIAG=HEIDEL
HeidelbergerByChain	Heidelberger-Welch diagnostics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)= HEIDEL
NUTSDiagnostics	Diagnostics for the No-U-Turn Sampler	BAYES	SAMSUM
NUTSMassMatrixDiag	Mass matrix for the No-U-Turn Sampler	BAYES	SAMSUM(VERBOSE)
PostCorr	Posterior correlation matrix	BAYES	STATS=CORR
PostCorrByChain	Posterior correlation matrix for each Markov chain	BAYES	STATS(BYCHAIN)= CORR
PostCov	Posterior covariance matrix	BAYES	STATS=COV
PostCovByChain	Posterior covariance matrix for each Markov chain	BAYES	STATS(BYCHAIN)=COV
PostIntervals	Equal-tail and highest posterior density intervals for each parameter	BAYES	STATS=INTERVALS

Table 10.5 *continued*

Table Name	Description	Statement	Option
PostIntervalsByChain	Equal-tail and highest posterior density intervals for each parameter in each Markov chain	BAYES	STATS(BYCHAIN)=INTERVALS
PostSummaries	Posterior mean, standard deviation, and percentiles for each parameter	BAYES	Default
PostSummariesByChain	Posterior mean, standard deviation, and percentiles for each parameter in each Markov chain	BAYES	STATS(BYCHAIN)=SUMMARIES
PriorSummary	Prior distribution for each parameter	BAYES	PRIORSUMMARY
Raftery	Raftery-Lewis diagnostics for each parameter	BAYES	DIAG=RAFTERY
RafteryByChain	Raftery-Lewis diagnostics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)=RAFTERY
SamplerSettings	Full MCMC sampler specification	BAYES	SAMSET

Examples: The CNTSELECT Procedure

Example 10.1: Zero-Inflated Poisson Model with CLASS Statement

This example shows how to use the CNTSELECT procedure to estimate a zero-inflated Poisson model with a classification variable `Group` that has two levels. The following DATA step generates 10,000 replicates from the zero-inflated Poisson (ZIP) model. The first 5,000 replicates belong to the first group, and the second 5,000 replicates belong to the second group. The model contains seven variables and three variables that correspond to the zero-inflated process.

```
data simulate;
  call streaminit(12345);
  array vars x1-x7;
  array zero_vars z1-z3;

  array parms{7} (.3 .4 .2 .4 -.3 -.5 -.3);
  array zero_parms{3} (-.6 .3 .2);

  intercept=0.5;
  group=1;
  z_intercept=-1;
  theta=0.5;

  do i=1 to 10000;
    sum_xb=0;
    sum_gz=0;
```

```

if i>5000 then do;
    intercept=2;
    group=2;
end;
do j=1 to 7;
    vars[j]=rand('NORMAL',0,1);
    sum_xb=sum_xb+parms[j]*vars[j];
end;
mu=exp(intercept+sum_xb);
y_p=rand('POISSON', mu);

do j=1 to 3;
    zero_vars[j]=rand('NORMAL',0,1);
    sum_gz = sum_gz+zero_parms[j]*zero_vars[j];
end;
z_gamma = z_intercept+sum_gz;
pzero = cdf('LOGISTIC',z_gamma);
cut=rand('UNIFORM');
if cut<pzero then y_p=0;
output;
end;
keep y_p group x1-x7 z1-z3;
run;

```

You can load the data set simulate into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```

data mylib.simulate;
    set simulate;
run;

```

The following statements estimate a zero-inflated Poisson model with the classification variable Group:

```

proc cntselect data=mylib.simulate dist=zip;
    class group;
    model y_p=group x1-x7;
    zeromodel y_p ~ z1-z3;
run;

```

Output 10.1.1 shows the results for the zero-inflated Poisson model.

Output 10.1.1 Zero-Inflated Poisson Model with CLASS Statement

The CNTSELECT Procedure

Class Level Information		
Class	Levels	Values
group	2	1 2

Output 10.1.1 *continued*

Model Fit Summary	
Dependent Variable	y_p
Number of Observations	10000
Data Set	SIMULATE
Model	ZIP
ZI Link Function	Logistic
Log Likelihood	-18176.3
Maximum Absolute Gradient	0.001931
Number of Iterations	7
Optimization Method	Newton-Raphson
AIC	36378.62
SBC	36472.35
Covariance Estimation	Hessian

Convergence criterion (FCONV=1E-8) satisfied.

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	2.001286	0.006746	296.67	<.0001
group 1	1	-1.491903	0.012383	-120.48	<.0001
group 2	0	0	.	.	.
x1	1	0.296633	0.004628	64.09	<.0001
x2	1	0.400643	0.004607	86.96	<.0001
x3	1	0.196278	0.004592	42.74	<.0001
x4	1	0.394971	0.004644	85.05	<.0001
x5	1	-0.299776	0.004508	-66.50	<.0001
x6	1	-0.497570	0.004809	-103.47	<.0001
x7	1	-0.295755	0.004510	-65.57	<.0001
Inf_Intercept	1	-0.969211	0.028573	-33.92	<.0001
Inf_z1	1	-0.609492	0.029128	-20.92	<.0001
Inf_z2	1	0.298572	0.027039	11.04	<.0001
Inf_z3	1	0.208062	0.026557	7.83	<.0001

The “Class Level Information” table shows that the classification variable Group has two levels. The “Model Fit Summary” table shows detailed information about the model and indicates that all 10,000 observations were used to fit the model. All parameter estimates in the “Parameter Estimates” table are highly significant and correspond to their theoretical values set during the data generating process.

Example 10.2: Zero-Inflated Poisson Model with BAYES and PRIOR Statements

This example shows how to use the CNTSELECT procedure to estimate a zero-inflated Poisson model by using Bayesian methods, with user-specified priors. The following DATA step generates 1,000 replicates from the zero-inflated Poisson (ZIP) model. The model contains seven variables and three variables that correspond to the zero-inflated process.

```
data bayes_ex;
  call streaminit(12345);
  array vars x1-x7;
  array zero_vars z1-z3;
  array parms{7} (.3 .4 .2 .4 -.3 -.5 -.3);
  array zero_parms{3} (-.6 .3 .2);
  intercept=0.5;
  group=1;
  z_intercept=-1;
  theta=0.5;
  do i=1 to 1000;
    sum_xb=0;
    sum_gz=0;
    if i>500 then do;
      intercept=2;
      group=2;
    end;
    do j=1 to 7;
      vars[j]=rand('NORMAL', 0, 1);
      sum_xb=sum_xb+parms[j]*vars[j];
    end;
    mu=exp(intercept+sum_xb);
    y_p=rand('POISSON', mu);
    do j=1 to 3;
      zero_vars[j]=rand('NORMAL', 0, 1);
      sum_gz = sum_gz+zero_parms[j]*zero_vars[j];
    end;
    z_gamma = z_intercept+sum_gz;
    pzero = cdf('LOGISTIC', z_gamma);
    cut=rand('UNIFORM');
    if cut<pzero then y_p=0;
    output;
  end;
  keep y_p group x1-x7 z1-z3;
run;
```

You can load the data set bayes_ex into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.bayes_ex;
  set bayes_ex;
run;
```

The following statements estimate a zero-inflated Poisson model via Bayesian analysis. Note that the BAYES statement controls the settings for the MCMC sampler and determines which Bayesian output to produce. The PRIOR statements set the prior distributions for the parameters.

```
proc cntselect data=mylib.bayes_ex dist=zip;
  class group;
  model y_p=group x1-x7;
  zeromodel y_p ~ z1-z3;
  bayes seed = 81239 nmc = 10000 sampler = rwm(ntu = 50) priorsummary(shownames);
  prior x1-x7 ~ normal(mean = 0, sd = 10);
  prior group_1 ~ normal(mean = 0, sd = 10, upper = 0);
  prior Intercept ~ normal(mean = 0, sd = 10);
  prior Inf_z1-Inf_z3 ~ normal(mean = 0, sd = 1);
  prior Inf_Intercept ~ normal(mean = 0, sd = 10);
run;
```

Output 10.2.1 shows the results for the zero-inflated Poisson model. The “Prior Summary” table shows detailed information about prior distributions for the parameters in the model. The “Posterior Summaries” table contains two point estimates of the parameters from the posterior samples: the posterior mean and the posterior median (50th percentile). The “MCMC Diagnostic Summaries” table contains basic convergence diagnostics to check whether the Markov chain has converged and whether the sample size is sufficient.

Output 10.2.1 Bayesian Estimation of Zero-Inflated Poisson Model

The CNTSELECT Procedure

Prior Summary								
Parameter	Name	Prior	Bounds		Hyperparameters			
			Lower	Upper	Name	Value	Name	Value
Intercept	Intercept	Normal	-Infy	Infy	Mean	0	Variance	100
group 1	group_1	Truncated Normal	-Infy	0	Mean	0	Variance	100
x1	x1	Normal	-Infy	Infy	Mean	0	Variance	100
x2	x2	Normal	-Infy	Infy	Mean	0	Variance	100
x3	x3	Normal	-Infy	Infy	Mean	0	Variance	100
x4	x4	Normal	-Infy	Infy	Mean	0	Variance	100
x5	x5	Normal	-Infy	Infy	Mean	0	Variance	100
x6	x6	Normal	-Infy	Infy	Mean	0	Variance	100
x7	x7	Normal	-Infy	Infy	Mean	0	Variance	100
Inf_Intercept	Inf_Intercept	Normal	-Infy	Infy	Mean	0	Variance	100
Inf_z1	Inf_z1	Normal	-Infy	Infy	Mean	0	Variance	1
Inf_z2	Inf_z2	Normal	-Infy	Infy	Mean	0	Variance	1
Inf_z3	Inf_z3	Normal	-Infy	Infy	Mean	0	Variance	1

Output 10.2.1 *continued*
The CNTSELECT Procedure

Posterior Summaries								
Parameter	N	Standard		Percentiles				
		Mean	Deviation	2.5%	25%	50%	75%	97.5%
Intercept	10000	1.9800	0.0131	1.9554	1.9707	1.9798	1.9885	2.0070
group 1	10000	-1.4828	0.0325	-1.5405	-1.5090	-1.4833	-1.4584	-1.4188
x1	10000	0.2853	0.0131	0.2605	0.2768	0.2851	0.2940	0.3121
x2	10000	0.4146	0.0107	0.3930	0.4081	0.4145	0.4219	0.4356
x3	10000	0.2129	0.0116	0.1901	0.2052	0.2134	0.2208	0.2362
x4	10000	0.4027	0.00998	0.3819	0.3971	0.4026	0.4088	0.4217
x5	10000	-0.3107	0.0129	-0.3358	-0.3196	-0.3101	-0.3022	-0.2849
x6	10000	-0.5188	0.0118	-0.5427	-0.5270	-0.5186	-0.5109	-0.4962
x7	10000	-0.2963	0.0123	-0.3186	-0.3053	-0.2967	-0.2882	-0.2720
Inf_Intercept	10000	-1.0055	0.0669	-1.1440	-1.0509	-1.0016	-0.9554	-0.8867
Inf_z1	10000	-0.5749	0.0791	-0.7243	-0.6287	-0.5779	-0.5268	-0.3997
Inf_z2	10000	0.4013	0.0914	0.2281	0.3346	0.3999	0.4617	0.5850
Inf_z3	10000	0.3761	0.0550	0.2736	0.3342	0.3737	0.4168	0.4807

The CNTSELECT Procedure

MCMC Diagnostic Summaries						
Parameter		MCSE		ESS	Autocorrelation	
		MCSE	MCSE/SD		Time	ESS/N
Intercept	*	0.00170	0.1300	59.1948	168.9	0.00592
group 1	*	0.00632	0.1943	26.4872	377.5	0.00265
x1	*	0.00171	0.1307	58.4973	170.9	0.00585
x2	*	0.000841	0.0787	161.6	61.8694	0.0162
x3		0.000726	0.0624	257.0	38.9091	0.0257
x4	*	0.00123	0.1228	66.2673	150.9	0.00663
x5	*	0.00160	0.1242	64.7892	154.3	0.00648
x6	*	0.00143	0.1214	67.9022	147.3	0.00679
x7	*	0.00111	0.0904	122.5	81.6416	0.0122
Inf_Intercept	*	0.00967	0.1445	47.8864	208.8	0.00479
Inf_z1	*	0.0111	0.1405	50.6374	197.5	0.00506
Inf_z2	*	0.0130	0.1418	49.7432	201.0	0.00497
Inf_z3	*	0.00961	0.1749	32.7080	305.7	0.00327

*Autocorrelation Remains

Example 10.3: Bayesian Analysis Using NUTS with Multiple Chains

This example shows how to use the CNTSELECT procedure to estimate a model in the Bayesian framework by using the No-U-Turn Sampler (NUTS) with multiple Markov chains.

The following DATA step creates the data set DocVisit, which contains information for approximately 5,000 Australian individuals about the number and possible determinants of doctor visits that were made during a two-week interval. This data set contains a subset of variables that are taken from the data set Racd3 that was used by Cameron and Trivedi (1998).

```
data docvisit;
  input sex age agesq income levyplus freepoor
        freerepa illness actdays hscore chcond1 chcond2
        doctorco nondocco hospadmi hospad medicine prescrib nonpresc;
  datalines;
1 0.19 0.0361 0.55 1 0 0 1 4 1 0 0 1 0 0 0 1 1 0
1 0.19 0.0361 0.45 1 0 0 1 2 1 0 0 1 0 0 0 2 1 1
0 0.19 0.0361 0.9 0 0 0 3 0 0 0 0 1 0 1 4 2 1 1
0 0.19 0.0361 0.15 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0.19 0.0361 0.45 0 0 0 2 5 1 1 0 1 0 0 0 3 1 2
1 0.19 0.0361 0.35 0 0 0 5 1 9 1 0 1 0 0 0 1 1 0
1 0.19 0.0361 0.55 0 0 0 4 0 2 0 0 1 0 0 0 0 0 0
1 0.19 0.0361 0.15 0 0 0 3 0 6 0 0 1 0 0 0 1 1 0
1 0.19 0.0361 0.65 1 0 0 2 0 5 0 0 1 0 0 0 1 0 1
0 0.19 0.0361 0.15 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0
0 0.19 0.0361 0.45 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0

... more lines ...
```

The variable Doctorco represents doctor visits. Additional variables in the data set that you want to evaluate as determinants of doctor visits include Sex (coded as 0=male, 1=female), Age (age in years divided by 100), Illness (number of illnesses during the two-week interval, with five or more coded as five), Income (annual income in Australian dollars divided by 1,000), and Hscore (a score on a general health questionnaire, in which a high score indicates bad health).

You can load the data set DocVisit into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.docvisit;
  set docvisit;
run;
```

The following statements estimate a Poisson regression model in the Bayesian framework by using the No-U-Turn Sampler (NUTS) with multiple Markov chains. NUTS is specified in the **SAMPLER=** option in the BAYES statement, and you can set several values that control the sampler as arguments to the **SAMPLER=NUTS** option. The **NCHAIN=** option controls the number of Markov chains, and the **NSAMPLE=** option controls the size of the posterior sample per chain. PROC CNTSELECT automatically aggregates the results of each chain to create the posterior statistics and MCMC diagnostics tables that you request. The BYCHAIN suboption of the **STATISTICS=** and **DIAGNOSTICS=** options additionally produces those tables separately for each chain. This example produces the “Posterior Summaries” table, the “MCMC Diagnostic Summaries” table, and the “Heidelberger-Welch Diagnostics” table for each chain as well as for all chains

combined.

```
proc cntselect data = mylib.docvisit;
  class sex illness;
  model doctorco=sex illness income hscore;
  bayes seed = 5317 nsample = 2000 nchain = 4
    sampler = nuts(ntune = 2000)
    statistics(bychain) = summary
    diagnostics(bychain) = (summary heidel);
run;
```

Output 10.3.1 shows the Bayesian estimation results. The “Posterior Summaries” table displays the estimates of the posterior mean and posterior quantiles that are computed using all four chains. The other posterior summary statistics tables that the **STATISTICS=** option produces would similarly be computed using all four chains. The “MCMC Diagnostic Summaries” table displays basic convergence diagnostics that are aggregated across all four chains. The “Autocorrelation Diagnostics” table would also be aggregated for multiple chains. For more information about how this aggregation is performed, see the section “Using Multiple Chains” on page 37 in Chapter 2, “Introduction to Bayesian Analysis Procedures.” The “Heidelberger-Welch Diagnostics” table displays a summary of all the Heidelberger-Welch tests that are performed for each chain. Specifically, for each parameter it shows the number of chains that passed each test, as well as the largest number of iterations that were discarded for the stationarity test. The “Geweke Diagnostics” table and the “Raftery-Lewis Diagnostics” table would both be summarized similarly.

Output 10.3.1 Posterior Summaries and MCMC Diagnostics

The CNTSELECT Procedure

Posterior Summaries								
				Percentiles				
Parameter	N	Mean	Standard Deviation	2.5%	25%	50%	75%	97.5%
Intercept	8000	-0.3880	0.0867	-0.5604	-0.4459	-0.3877	-0.3300	-0.2209
sex 0	8000	-0.2200	0.0537	-0.3267	-0.2553	-0.2202	-0.1850	-0.1145
illness 0	8000	-1.9385	0.1220	-2.1783	-2.0203	-1.9384	-1.8551	-1.7014
illness 1	8000	-0.6981	0.0895	-0.8773	-0.7587	-0.6976	-0.6386	-0.5215
illness 2	8000	-0.4704	0.0907	-0.6479	-0.5321	-0.4687	-0.4076	-0.2932
illness 3	8000	-0.4880	0.0988	-0.6832	-0.5553	-0.4863	-0.4191	-0.2978
illness 4	8000	-0.2736	0.1088	-0.4857	-0.3475	-0.2720	-0.1998	-0.0602
income	8000	-0.2534	0.0766	-0.4016	-0.3061	-0.2530	-0.2008	-0.1071
hscore	8000	0.0943	0.00899	0.0768	0.0882	0.0944	0.1005	0.1117

Output 10.3.1 *continued*
The CNTSELECT Procedure

MCMC Diagnostic Summaries					
Parameter	MCSE	MCSE/SD	ESS	Autocorrelation	
				Time	ESS/N
Intercept	0.00195	0.0225	1983.1	4.0340	0.2479
sex 0	0.000739	0.0137	5291.7	1.5118	0.6615
illness 0	0.00233	0.0191	2749.8	2.9093	0.3437
illness 1	0.00184	0.0205	2380.8	3.3602	0.2976
illness 2	0.00191	0.0210	2259.7	3.5403	0.2825
illness 3	0.00193	0.0195	2628.1	3.0441	0.3285
illness 4	0.00192	0.0177	3197.7	2.5018	0.3997
income	0.00115	0.0150	4474.0	1.7881	0.5593
hscore	0.000139	0.0154	4209.6	1.9004	0.5262

Heidelberger-Welch Diagnostics					
Parameter	Stationarity Test			Half-Width Test	
	Chains Tested	Tests Passed	Largest	Chains Tested	Tests Passed
			Iterations Discarded		
Intercept	4	4	0	4	4
sex 0	4	4	0	4	4
illness 0	4	4	0	4	4
illness 1	4	4	0	4	4
illness 2	4	4	0	4	4
illness 3	4	4	0	4	4
illness 4	4	4	0	4	4
income	4	4	0	4	4
hscore	4	4	0	4	4

Detailed results for each chain are available in [Output 10.3.2](#) as a result of the BYCHAIN options. The “Posterior Summaries by Chain” table displays the estimates of the posterior mean and posterior quantiles that are computed separately for each chain. Similarly, the “MCMC Diagnostic Summaries by Chain” table displays basic convergence diagnostics that are computed for each chain, and the “Heidelberger-Welch Diagnostics by Chain” table displays the Heidelberger-Welch diagnostics that are computed for each chain. For more information about the differences between the tables in [Output 10.3.1](#) and [Output 10.3.2](#), see the section “Using Multiple Chains” on page 37 in Chapter 2, “Introduction to Bayesian Analysis Procedures.”

Output 10.3.2 Posterior Summaries and MCMC Diagnostics by Chain
The CNTSELECT Procedure

Posterior Summaries by Chain									
Chain	Parameter	N	Standard		Percentiles				
			Mean	Deviation	2.5%	25%	50%	75%	97.5%
1	Intercept	2000	-0.3890	0.0889	-0.5713	-0.4462	-0.3878	-0.3280	-0.2251
	sex 0	2000	-0.2187	0.0529	-0.3273	-0.2531	-0.2177	-0.1839	-0.1174
	illness 0	2000	-1.9370	0.1222	-2.1747	-2.0219	-1.9346	-1.8559	-1.6948
	illness 1	2000	-0.6966	0.0915	-0.8787	-0.7594	-0.6977	-0.6374	-0.5124
	illness 2	2000	-0.4700	0.0934	-0.6487	-0.5323	-0.4731	-0.4078	-0.2739
	illness 3	2000	-0.4842	0.1011	-0.6870	-0.5518	-0.4815	-0.4145	-0.2853
	illness 4	2000	-0.2708	0.1131	-0.4917	-0.3475	-0.2700	-0.1955	-0.0506
	income	2000	-0.2534	0.0781	-0.4102	-0.3058	-0.2503	-0.2001	-0.1076
	hscore	2000	0.0942	0.00889	0.0766	0.0881	0.0943	0.1002	0.1112
2	Intercept	2000	-0.3952	0.0859	-0.5615	-0.4570	-0.3959	-0.3353	-0.2266
	sex 0	2000	-0.2193	0.0552	-0.3321	-0.2557	-0.2188	-0.1822	-0.1138
	illness 0	2000	-1.9295	0.1209	-2.1668	-2.0109	-1.9284	-1.8446	-1.6998
	illness 1	2000	-0.6906	0.0902	-0.8674	-0.7525	-0.6890	-0.6284	-0.5204
	illness 2	2000	-0.4633	0.0911	-0.6502	-0.5273	-0.4612	-0.3991	-0.2961
	illness 3	2000	-0.4828	0.1008	-0.6834	-0.5509	-0.4783	-0.4116	-0.2915
	illness 4	2000	-0.2672	0.1071	-0.4802	-0.3378	-0.2648	-0.1951	-0.0595
	income	2000	-0.2549	0.0783	-0.4036	-0.3112	-0.2565	-0.1998	-0.1034
	hscore	2000	0.0945	0.00915	0.0768	0.0883	0.0946	0.1007	0.1121
3	Intercept	2000	-0.3854	0.0876	-0.5567	-0.4421	-0.3858	-0.3284	-0.2149
	sex 0	2000	-0.2211	0.0532	-0.3244	-0.2561	-0.2209	-0.1859	-0.1177
	illness 0	2000	-1.9445	0.1223	-2.1891	-2.0227	-1.9458	-1.8619	-1.7014
	illness 1	2000	-0.6998	0.0887	-0.8772	-0.7594	-0.6989	-0.6417	-0.5174
	illness 2	2000	-0.4700	0.0898	-0.6480	-0.5313	-0.4654	-0.4091	-0.2953
	illness 3	2000	-0.4888	0.0974	-0.6759	-0.5570	-0.4884	-0.4236	-0.2999
	illness 4	2000	-0.2761	0.1079	-0.4858	-0.3522	-0.2756	-0.2011	-0.0680
	income	2000	-0.2544	0.0746	-0.4029	-0.3052	-0.2540	-0.2038	-0.1121
	hscore	2000	0.0941	0.00878	0.0767	0.0881	0.0943	0.1002	0.1108
4	Intercept	2000	-0.3826	0.0837	-0.5373	-0.4400	-0.3828	-0.3258	-0.2144
	sex 0	2000	-0.2208	0.0536	-0.3262	-0.2564	-0.2223	-0.1867	-0.1100
	illness 0	2000	-1.9431	0.1222	-2.1852	-2.0227	-1.9432	-1.8595	-1.7071
	illness 1	2000	-0.7052	0.0873	-0.8803	-0.7631	-0.7051	-0.6456	-0.5390
	illness 2	2000	-0.4781	0.0881	-0.6426	-0.5385	-0.4774	-0.4176	-0.3060
	illness 3	2000	-0.4959	0.0955	-0.6877	-0.5606	-0.4930	-0.4303	-0.3160
	illness 4	2000	-0.2801	0.1066	-0.4918	-0.3562	-0.2784	-0.2049	-0.0751
	income	2000	-0.2510	0.0753	-0.3888	-0.3037	-0.2517	-0.1998	-0.1045
	hscore	2000	0.0946	0.00913	0.0770	0.0884	0.0944	0.1008	0.1126

Output 10.3.2 *continued*
The CNTSELECT Procedure

MCMC Diagnostic Summaries by Chain						
Chain	Parameter	MCSE	MCSE/SD	ESS	Autocorrelation	
					Time	ESS/N
1	Intercept	0.00380	0.0427	548.7	3.6447	0.2744
	sex 0	0.00148	0.0280	1272.5	1.5717	0.6363
	illness 0	0.00433	0.0354	796.8	2.5099	0.3984
	illness 1	0.00313	0.0342	852.5	2.3459	0.4263
	illness 2	0.00364	0.0390	657.8	3.0405	0.3289
	illness 3	0.00375	0.0371	728.3	2.7459	0.3642
	illness 4	0.00327	0.0289	1198.6	1.6686	0.5993
	income	0.00217	0.0278	1293.4	1.5463	0.6467
	hscore	0.000253	0.0284	1238.4	1.6150	0.6192
2	Intercept	0.00395	0.0460	471.6	4.2407	0.2358
	sex 0	0.00155	0.0280	1276.5	1.5668	0.6382
	illness 0	0.00490	0.0405	608.2	3.2881	0.3041
	illness 1	0.00368	0.0408	601.8	3.3231	0.3009
	illness 2	0.00405	0.0445	504.4	3.9652	0.2522
	illness 3	0.00367	0.0364	755.3	2.6480	0.3776
	illness 4	0.00401	0.0374	713.4	2.8037	0.3567
	income	0.00243	0.0311	1035.3	1.9318	0.5176
	hscore	0.000274	0.0299	1116.4	1.7916	0.5582
3	Intercept	0.00425	0.0486	423.8	4.7187	0.2119
	sex 0	0.00148	0.0279	1284.7	1.5567	0.6424
	illness 0	0.00476	0.0389	660.8	3.0266	0.3304
	illness 1	0.00410	0.0462	468.4	4.2699	0.2342
	illness 2	0.00408	0.0455	483.2	4.1387	0.2416
	illness 3	0.00422	0.0433	533.3	3.7506	0.2666
	illness 4	0.00423	0.0392	650.3	3.0757	0.3251
	income	0.00211	0.0283	1251.4	1.5982	0.6257
	hscore	0.000298	0.0340	866.2	2.3090	0.4331
4	Intercept	0.00352	0.0420	566.3	3.5319	0.2831
	sex 0	0.00139	0.0260	1479.3	1.3520	0.7397
	illness 0	0.00458	0.0375	711.1	2.8125	0.3556
	illness 1	0.00365	0.0418	571.1	3.5017	0.2856
	illness 2	0.00342	0.0388	662.9	3.0168	0.3315
	illness 3	0.00372	0.0389	659.7	3.0318	0.3298
	illness 4	0.00374	0.0351	813.2	2.4593	0.4066
	income	0.00243	0.0322	963.4	2.0761	0.4817
	hscore	0.000281	0.0307	1060.3	1.8862	0.5302

Output 10.3.2 continued

Heidelberger-Welch Diagnostics by Chain								
Chain	Parameter	Stationarity Test			Half-Width Test			
		Cramer-von Mises Stat	Test p-Value	Test Outcome	Iterations Discarded	Half-Width	Mean Half-Width	Relative Test Outcome
1	Intercept	0.0933	0.6189	Passed	0	0.00679	-0.3890	-0.0174 Passed
	sex 0	0.1256	0.4735	Passed	0	0.00292	-0.2187	-0.0133 Passed
	illness 0	0.0868	0.6537	Passed	0	0.00793	-1.9370	-0.00409 Passed
	illness 1	0.0589	0.8208	Passed	0	0.00628	-0.6966	-0.00902 Passed
	illness 2	0.0562	0.8377	Passed	0	0.00604	-0.4700	-0.0128 Passed
	illness 3	0.0830	0.6749	Passed	0	0.00660	-0.4842	-0.0136 Passed
	illness 4	0.0420	0.9225	Passed	0	0.00675	-0.2708	-0.0249 Passed
	income	0.2201	0.2318	Passed	0	0.00446	-0.2534	-0.0176 Passed
	hscore	0.0851	0.6630	Passed	0	0.000476	0.0942	0.00506 Passed
2	Intercept	0.1632	0.3516	Passed	0	0.00785	-0.3952	-0.0199 Passed
	sex 0	0.1064	0.5545	Passed	0	0.00308	-0.2193	-0.0140 Passed
	illness 0	0.0845	0.6664	Passed	0	0.0103	-1.9295	-0.00535 Passed
	illness 1	0.1439	0.4088	Passed	0	0.00793	-0.6906	-0.0115 Passed
	illness 2	0.1492	0.3920	Passed	0	0.00805	-0.4633	-0.0174 Passed
	illness 3	0.1601	0.3601	Passed	0	0.00771	-0.4828	-0.0160 Passed
	illness 4	0.2057	0.2567	Passed	0	0.00861	-0.2672	-0.0322 Passed
	income	0.1084	0.5454	Passed	0	0.00460	-0.2549	-0.0180 Passed
	hscore	0.1427	0.4127	Passed	0	0.000506	0.0945	0.00535 Passed
3	Intercept	0.0849	0.6639	Passed	0	0.00870	-0.3854	-0.0226 Passed
	sex 0	0.0451	0.9054	Passed	0	0.00283	-0.2211	-0.0128 Passed
	illness 0	0.0642	0.7875	Passed	0	0.0101	-1.9445	-0.00520 Passed
	illness 1	0.0761	0.7149	Passed	0	0.00810	-0.6998	-0.0116 Passed
	illness 2	0.1187	0.5006	Passed	0	0.00837	-0.4700	-0.0178 Passed
	illness 3	0.0755	0.7180	Passed	0	0.00860	-0.4888	-0.0176 Passed
	illness 4	0.0379	0.9437	Passed	0	0.00818	-0.2761	-0.0296 Passed
	income	0.1017	0.5767	Passed	0	0.00457	-0.2544	-0.0180 Passed
	hscore	0.0783	0.7016	Passed	0	0.000606	0.0941	0.00644 Passed
4	Intercept	0.1059	0.5565	Passed	0	0.00614	-0.3826	-0.0160 Passed
	sex 0	0.0966	0.6016	Passed	0	0.00259	-0.2208	-0.0117 Passed
	illness 0	0.1086	0.5443	Passed	0	0.00834	-1.9431	-0.00429 Passed
	illness 1	0.0311	0.9724	Passed	0	0.00703	-0.7052	-0.00996 Passed
	illness 2	0.0334	0.9637	Passed	0	0.00639	-0.4781	-0.0134 Passed
	illness 3	0.0314	0.9712	Passed	0	0.00709	-0.4959	-0.0143 Passed
	illness 4	0.0354	0.9551	Passed	0	0.00720	-0.2801	-0.0257 Passed
	income	0.0622	0.8003	Passed	0	0.00464	-0.2510	-0.0185 Passed
	hscore	0.2856	0.1484	Passed	0	0.000571	0.0946	0.00604 Passed

References

Cameron, A. C., and Trivedi, P. K. (1986). "Econometric Models Based on Count Data: Comparisons and Applications of Some Estimators and Tests." *Journal of Applied Econometrics* 1:29–53.

- Cameron, A. C., and Trivedi, P. K. (1998). *Regression Analysis of Count Data*. Cambridge: Cambridge University Press.
- Conway, R. W., and Maxwell, W. L. (1962). "A Queuing Model with State Dependent Service Rates." *Journal of Industrial Engineering* 12:132–136.
- Gallant, A. R. (1987). *Nonlinear Statistical Models*. New York: John Wiley & Sons.
- Gelman, A. (2006). "Prior Distributions for Variance Parameters in Hierarchical Models." *Bayesian Analysis* 1:515–533.
- Gregory, A. W., and Veall, M. R. (1985). "On Formulating Wald Tests for Nonlinear Restrictions." *Econometrica* 53:1465–1468.
- Guikema, S. D., and Coffelt, J. P. (2008). "A Flexible Count Data Regression Model for Risk Analysis." *Risk Analysis* 28:213–223.
- Lambert, D. (1992). "Zero-Inflated Poisson Regression with an Application to Defects in Manufacturing." *Technometrics* 34:1–14.
- Long, J. S. (1997). *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, CA: Sage Publications.
- Lord, D., Guikema, S. D., and Geedipally, S. R. (2008). "Application of the Conway-Maxwell-Poisson Generalized Linear Model for Analyzing Motor Vehicle Crashes." *Accident Analysis and Prevention* 40:1123–1134.
- Phillips, C. B., and Park, J. Y. (1988). "On Formulating Wald Tests of Nonlinear Restrictions." *Econometrica* 56:1065–1083.
- Savitsky, T. D., and Toth, D. (2016). "Bayesian Estimation under Informative Sampling." *Electronic Journal of Statistics* 10:1677–1708.
- Shmueli, G., Minka, T. P., Kadane, J. B., Borle, S., and Boatwright, P. (2005). "A Useful Distribution for Fitting Discrete Data: Revival of the Conway-Maxwell-Poisson Distribution." *Journal of the Royal Statistical Society, Series C* 54:127–142.

Chapter 11

CPANEL Procedure

Contents

Overview: CPANEL Procedure	462
PROC CPANEL Features	463
PROC CPANEL Compared with Other SAS Procedures	464
Using CAS Sessions and CAS Engine Librefs	465
Getting Started: CPANEL Procedure	466
Syntax: CPANEL Procedure	468
Functional Summary	468
PROC CPANEL Statement	470
BY Statement	471
CLASS Statement	471
COMPARE Statement	472
CORRELATED Statement	474
DISPLAY Statement	474
DISPLAYOUT Statement	475
ENDOGENOUS Statement	476
ID Statement	477
INSTRUMENTS Statement	478
MODEL Statement	479
OUTPUT Statement	488
PREDETERMINED Statement	489
RESTRICT Statement	489
TEST Statement	490
Details: CPANEL Procedure	491
Specifying the Input Data	491
Specifying the Regression Model	492
Missing Values	493
Unbalanced Data	493
Common Notation	493
Pooled Regression (POOLED Option)	494
Between-Groups Regression (BTWNG and BTWNT Options)	494
One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)	494
One-Way Fixed-Effects Model, First Differencing (FDONE Option)	495
Two-Way Fixed-Effects Model (FIXTWO Option)	496
One-Way Random-Effects Model (RANONE Option)	497
Two-Way Random-Effects Model (RANTWO Option)	500
Hausman-Taylor Estimation (HTAYLOR Option)	503

Amemiya-MaCurdy Estimation (AMACURDY Option)	505
Instrumental Variables Regression	505
Dynamic Panel Estimation (DYNDIFF and DYN SYS Options)	511
Poolability Test for Fixed Effects	521
Hausman Specification Tests	521
Restricted Estimation	521
Linear Hypothesis Testing	522
Heteroscedasticity-Corrected Covariance Matrices	523
Heteroscedasticity- and Autocorrelation-Consistent Covariance Matrices	526
R-Square	529
Displayed Output	529
ODS Table Names	532
Examples: CPANEL Procedure	534
Example 11.1: Airline Cost Data: Fixed Effects	534
Example 11.2: Analyzing Demand for Liquid Assets: Random Effects	538
Example 11.3: Panel Study of Income Dynamics (PSID): Hausman-Taylor Models	541
Example 11.4: Crime Rates in North Carolina: Instrumental Variables Regression	546
Example 11.5: Cigarette Sales Data: Dynamic Panel Estimation	550
References	556

Overview: CPANEL Procedure

The CPANEL procedure analyzes a class of linear econometric models that arise when time series and cross-sectional data are combined. This type of data is called *panel data*. Typical examples of panel data include observations over time on people, households, countries, or firms. For example, in the case of survey data on household income, the panel is created by repeatedly surveying the same households over many years. The individual entities that are followed over time are called *cross sections*.

Regression models for panel data are characterized by an error structure that can be divided into a cross-sectional component, a time component, and an observation-level component. Panel data models can be categorized by the exact structure of the error term and how it interacts with the regression equation. The CPANEL procedure supports the following models:

- one-way and two-way models
- fixed-effects, random-effects, and hybrid models
- instrumental variables (IV) models
- dynamic panel models

A one-way model depends only on the cross section to which the observation belongs. A two-way model depends on both the cross section and the time period to which the observation belongs. The models are called fixed-effects models if the effects are nonrandom and are called random-effects models otherwise.

If the effects are fixed, the models are essentially regression models with dummy variables that correspond to the effects. For fixed-effects models, ordinary least squares (OLS) estimation, with some facility for absorbing the dummy variables, is the best linear unbiased estimator. Random-effects models use a two-stage approach. In the first stage, variance components are estimated by your choice of the methods described by Swamy and Arora (1972); Fuller and Battese (1974); Wansbeek and Kapteyn (1989); Wallace and Hussain (1969); Nerlove (1971). In the second stage, variance components are used to standardize the data, and then OLS regression is performed.

Random-effects models are more efficient than fixed-effects models, and they can estimate effects for variables that do not vary within cross sections. The cost of these added features is that random-effects models carry much more stringent assumptions than their fixed-effects counterparts. The CPANEL procedure also supports models that blend the desirable features of random and fixed effects. These hybrid models are described by Hausman and Taylor (1981) and Amemiya and MaCurdy (1986).

Instrumental variables (IV) models are models that introduce auxiliary regression variables to deal with endogenous regressors—regressors that are correlated with the residual errors. The CPANEL procedure supports IV generalizations of the pooled regression model, the between-groups regression model, the one-way fixed-effects model, the one-way random-effects model, the two-way fixed-effects model, and the two-way random-effects model. For the one-way random-effects model, PROC CPANEL uses methods that were developed by Baltagi (1981). PROC CPANEL can also estimate these models by using the efficient generalized method of moments (GMM).

Dynamic panel models are autoregression models that include lagged versions of the dependent variable as regressors. Examples of such models can be found in Arellano and Bond (1991) and Blundell and Bond (1998). The CPANEL procedure fits these models by using the generalized method of moments (GMM). You can perform GMM on either the difference equations or the full system of difference and level equations.

PROC CPANEL Features

The CPANEL procedure fits panel-data regression models by using various techniques, and does the following:

- provides an **ID** statement for specifying the cross section and time variables
- provides flexible model-building syntax that supports continuous regressors, classification (**CLASS**) variables, interactions, and nested effects
- fits one-way and two-way fixed-effects models
- fits one-way and two-way random-effects models
- fits Hausman and Taylor (1981) models
- fits Amemiya and MaCurdy (1986) models
- fits instrumental variables (IV) regression models
- fits dynamic panel models
- enables you to fit multiple models by using multiple estimators, all within one procedure call

- provides a **COMPARE** statement for producing customizable comparisons of various models and estimators
- provides a **TEST** statement for performing linear tests of hypotheses
- provides a **RESTRICT** statement for imposing linear parameter constraints
- provides an **OUTPUT** statement for producing a data table that contains predictions and residuals

Because the CPANEL procedure runs on SAS Cloud Analytic Services (CAS), it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on CAS
- exploits all the available cores and concurrent threads. For information about how PROC CPANEL uses threads, see the section “[Multithreading](#)” on page 108 in Chapter 4, “[Shared Concepts](#).”

PROC CPANEL Compared with Other SAS Procedures

The CPANEL procedure models panel data in a way that is comparable to how the HPPANEL and PANEL procedures in SAS/ETS software do this.

PROC CPANEL Compared with the HPPANEL Procedure

The functionality of the CPANEL procedure closely resembles that of the HPPANEL procedure, which is a high-performance procedure. The CPANEL procedure is the next generation of the HPPANEL procedure, and it was developed specifically for SAS Viya. Both procedures are designed to run on a cluster of machines that distribute the data and the computations. Both procedures perform computations in multiple threads.

The CPANEL procedure provides all the functionality of the HPPANEL procedure, and also provides the following:

- instrumental variable (IV) regression estimators
- dynamic panel estimators, as described by Arellano and Bond (1991) and Blundell and Bond (1998)
- Hausman and Taylor (1981) and Amemiya and MaCurdy (1986) estimators, which are hybrids of fixed-effects and random-effects estimators
- estimation of the fixed-effects model by first differencing (FD)
- Swamy and Arora (1972) estimates of variance components
- tables that compare parameters from multiple models and multiple estimation methods
- the option to specify a character variable as the cross section ID variable
- the option to not specify a time series ID variable when time is not relevant to the estimation

PROC CPANEL Compared with the PANEL Procedure

The CPANEL procedure provides the following functionality not found in PROC PANEL:

- instrumental variables (IV) regression estimators
- Swamy and Arora (1972) estimates of variance components
- support for estimation when the data contain cross sections that consist of only one observation
- the option to specify a character variable as the cross section ID variable
- the option to not specify a time-series ID variable when time is not relevant to the estimation

The PANEL procedure parameterizes the fixed-effects model by using the last ordered cross section and the latest time point as reference categories. The CPANEL procedure uses a mean-centered parameterization.

The PANEL procedure supports the Da Silva (1975) method for serial correlation, and the Parks (1967) method for first-order autocorrelation. These two methods are not provided in this release of the CPANEL procedure.

The PANEL procedure offers a wide variety of postfitting analyses, such as model specification tests and tests for unit root. The PANEL procedure is also able to construct utility data sets that contain lagged variables and data reformatted from wide form to long form. This release of the CPANEL procedure is limited in postfitting and data-generating functionality, because the focus for large data sets is primarily on model fitting and comparison.

The CPANEL procedure is specifically designed to operate in SAS Viya and performs computations in multiple threads. The PANEL procedure executes in a single thread on a single machine.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: CPANEL Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “[Using CAS Sessions and CAS Engine Librefs](#)” on page 54 and “[Loading a SAS Data Set onto a CAS Server](#)” on page 55 in Chapter 4, “[Shared Concepts](#).”

The following DATA step creates the data table Electricity from the cost function data in Greene (1990). This DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref. The variable Production is the log of output in millions of kilowatt-hours, and the variable Cost is the log of cost in millions of dollars.

```
data mylib.Electricity;
  input firm year production cost @@;
datalines;
1 1955 5.36598 1.14867 1 1960 6.03787 1.45185
1 1965 6.37673 1.52257 1 1970 6.93245 1.76627
2 1955 6.54535 1.35041 2 1960 6.69827 1.71109
2 1965 7.40245 2.09519 2 1970 7.82644 2.39480
3 1955 8.07153 2.94628 3 1960 8.47679 3.25967
3 1965 8.66923 3.47952 3 1970 9.13508 3.71795
4 1955 8.64259 3.56187 4 1960 8.93748 3.93400
4 1965 9.23073 4.11161 4 1970 9.52530 4.35523
5 1955 8.69951 3.50116 5 1960 9.01457 3.68998
5 1965 9.04594 3.76410 5 1970 9.21074 4.05573
6 1955 9.37552 4.29114 6 1960 9.65188 4.59356
6 1965 10.21163 4.93361 6 1970 10.34039 5.25520
;
```

Consider the model

$$C_{it} = \beta_0 + \beta_1 P_{it} + v_i + e_{it} \text{ for } i = 1, \dots, N \text{ and } t = 1, \dots, T$$

where C_{it} represents cost, P_{it} represents production, v_i is the cross-sectional error component, and e_{it} is the error variance component.

If you assume that the cross-sectional effects are random, five possible estimators are available for the variance components. The VCOMP=FB option in the following statements uses the Fuller and Battese (1974) estimator to fit the model:

```
proc cpanel data = mylib.Electricity;
  id firm year;
  model cost = production / ranone vcomp = fb;
run;
```

The output of these statements is shown in [Output 11.1](#).

Figure 11.1 One-Way Random-Effects Estimation Results

The CPANEL Procedure
One-Way Random Effects (RanOne)
Fuller and Battese Variance Components
Dependent Variable: cost

Model Description			
Estimation Method		RanOne	
Variance Estimation		Model Based	
Data Set		ELECTRICITY	
Number of Observations		24	
Number of Cross Sections		6	
Time Series Length		4	

Fit Statistics			
SSE	0.4143	DFE	22
MSE	0.0188	Root MSE	0.1372
R-Square	0.9164		

Variance Component Estimates		
Source	Variance	Standard Deviation
Cross Sections	0.04109	0.202707
Error	0.015533	0.124632

Hausman Test For Random Effects			
Coefficients	DF	m Value	Pr > m
1	1	9.08	0.0026

Parameter Estimates					
Variable	DF	Estimate	Standard Error	t Value	Pr > t
Intercept	1	-3.27307	0.4277	-7.65	<.0001
production	1	0.779469	0.0502	15.53	<.0001

Printed first is a report that provides the estimation method and various data counts. Fit statistics and variance components estimates are printed next. A Hausman specification test compares this model to its fixed-effects counterpart. Finally, the table of regression parameter estimates shows the estimates, standard errors, and t tests.

Syntax: CPANEL Procedure

The following statements are available in the CPANEL procedure:

```

PROC CPANEL < options > ;
  BY variables ;
  CLASS variable < (options) > ... < variable < (options) > > < / global-options > ;
  COMPARE < model-list > < / options > ;
  CORRELATED effects ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  ENDOGENOUS effects < / option > ;
  ID cross-section-id < time-series-id > ;
  INSTRUMENTS effects < / option > ;
  MODEL response = < effects > < / options > ;
  OUTPUT OUT=libref.data-table < options > ;
  PREDETERMINED effects < / option > ;
  RESTRICT equation1 < , equation2... > ;
  TEST equation1 < , equation2... > ;

```

Functional Summary

Table 11.1 summarizes the statements and options that the CPANEL procedure supports.

Table 11.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CPANEL	DATA=
Prevents partitioning of data by cross sections	PROC CPANEL	NOPART
Specifies the name of an output SAS data table	OUTPUT	OUT=
Variable Role Statements		
Specifies BY-group processing	BY	
Specifies the classification variables	CLASS	
Specifies the cross section and time ID variables	ID	
Declares correlated variables or effects	CORRELATED	
Declares endogenous variables or effects	ENDOGENOUS	
Declares instrumental variables or effects	INSTRUMENTS	
Declares predetermined variables or effects	PREDETERMINED	
Printing Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	

Table 11.1 *continued*

Description	Statement	Option
Prints correlations of the estimates	MODEL	CORRB
Prints covariances of the estimates	MODEL	COVB
Suppress parameter labels from output tables	MODEL	NOLABEL
Suppresses printed output	PROC CPANEL	NOPRINT
Prints all available output	PROC CPANEL	PRINTALL
Prints fixed effects	MODEL	PRINTFIXED
Model Estimation Technique Options		
Specifies the Amemiya-MaCurdy model	MODEL	AMACURDY
Specifies the between-groups model	MODEL	BTWNG
Specifies the between-time-periods model	MODEL	BTWNT
Specifies the dynamic panel model via first-differencing GMM	MODEL	DYNDIFF
Specifies the dynamic panel model via system GMM	MODEL	DYNSYS
Specifies the one-way fixed-effects model via first-differencing	MODEL	FDONE
Specifies the one-way fixed-effects model	MODEL	FIXONE
Specifies the one-way fixed-effects model with respect to time	MODEL	FIXONETIME
Specifies the two-way fixed-effects model	MODEL	FIXTWO
Specifies the Hausman-Taylor model	MODEL	HTAYLOR
Specifies the between-groups model with instrumental variables	MODEL	IVBTWNG
Specifies the one-way fixed-effects model with instrumental variables	MODEL	IVFIXONE
Specifies the one-way fixed-effects model for time effects with instrumental variables	MODEL	IVFIXONETIME
Specifies the two-way fixed-effects model with instrumental variables	MODEL	IVFIXTWO
Specifies the generalized two-stage least squares model with instrumental variables	MODEL	IVG2SLS
Specifies the pooled regression model with instrumental variables	MODEL	IVPOOLED
Specifies the one-way random-effects model with instrumental variables	MODEL	IVRANONE
Specifies the two-way random-effects model with instrumental variables	MODEL	IVRANTWO
Specifies the pooled regression model	MODEL	POOLED
Specifies the one-way random-effects model	MODEL	RANONE
Specifies the two-way random-effects model	MODEL	RANTWO
Model Estimation Control Options		
Sets the number of AR(m) tests	MODEL	ARTESTS=

Table 11.1 *continued*

Description	Statement	Option
Specifies bias-corrected variances for two-step GMM	MODEL	BIASCORRECTED
Requests the clustered HCCME estimator for the covariance matrix	MODEL	CLUSTER
Sets the number of dependent variable lags	MODEL	DLAGS=
Specifies the generalized inverse type	MODEL	GINV=
Specifies one-step or two-step GMM	MODEL	GMM=
Requests the HAC estimator for the variance-covariance matrix	MODEL	HAC
Requests the HCCME estimator for the covariance matrix	MODEL	HCCME=
Limits the number of GMM instruments	MODEL	MAXBAND=
Suppresses the intercept term	MODEL	NOINT
Specifies robust variances estimator	MODEL	ROBUST
Specifies the method for the variance components estimator	MODEL	VCOMP=
Tests and Restrictions		
Specifies linear equality restrictions on the parameters	RESTRICT	
Performs tests of linear hypotheses	TEST	WALD, LM, LR
Model Comparison Statements		
Create tables that display side-by-side model comparisons	COMPARE	

PROC CPANEL Statement

PROC CPANEL <options> ;

The PROC CPANEL statement invokes the procedure. You can specify the following *options*:

DATA=*libref.data-table*

names the input data table for PROC CPANEL to use. The default is the most recently created data table. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 465.

data-table specifies the name of the input data table.

NOPART

prevents repartitioning of the data. Normally, the CPANEL procedure redistributes the data table so that observations for each cross section are not divided among multiple machines. Repartitioning the data makes model computations more efficient but has a cost in memory and computing cycles that are associated with redistribution. Specifying the NOPART option leaves the data distributed as they are, at the expense of less efficient model computations.

NOPRINT

suppresses all printed output.

PRINTALL

prints all available output.

In addition, any of the following MODEL statement options can be specified in the PROC CPANEL statement: **CORRB**, **COVB**, **NOLABEL**, and **PRINTFIXED**. When specified in the PROC CPANEL statement, these options apply globally to every MODEL statement. For a complete description of each of these options, see the section “MODEL Statement” on page 479.

BY Statement

BY *variables* ;

You can specify a BY statement in PROC CPANEL to obtain separate analyses of observations in groups that are defined by the values of the BY variables. If you specify more than one BY statement, only the last one specified is used. For more information, see the discussion of BY-group processing in the “Grouping Data” section of *SAS Programmers Guide: Essentials*.

CLASS Statement

CLASS *variable* < (*options*) > . . . < *variable* < (*options*) > > < / *global-options* > ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. Table 11.2 summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “CLASS Statement” on page 56 in Chapter 4, “Shared Concepts.”

Table 11.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable

COMPARE Statement

COMPARE <model-list> </options> ;

The COMPARE statement creates tables of side-by-side comparisons of parameter estimates and other model statistics. You can fit multiple models simultaneously by specifying multiple MODEL statements, and you can specify a COMPARE statement to create tables that compare the models.

The COMPARE statement creates two tables: the first table compares model fit statistics such as R-square and mean square error; the second table compares regression coefficients, their standard errors, and (optionally) *t* tests.

By default, comparison tables are created for all fitted models, but you can use the optional *model-list* to limit the comparison to a subset of the fitted models. The *model-list* consists of a set of model labels, as specified in the MODEL statement; for more information, see the section “MODEL Statement” on page 479. If a model does not have a label, you refer to it generically as “Model *i*,” where the corresponding model is the *i*th MODEL statement specified.

You can specify one or more COMPARE statements. The following code illustrates the use of the COMPARE statement:

```
proc cpanel data = mylib.a;
  id csid tsid;
  mod_one: model y = x1 x2 x3 / fixone;
  second: y = x1 x2 / fixone;
  model y = x1 x2 x3 x4 / ranone;
  compare;
  compare "second" "Model 3";
run;
```

The first COMPARE statement compares all three fitted models. The second COMPARE statement uses the generic “Model 3” to identify the third model and compares it to the second model.

You can specify the following *options* after a slash (/):

MSTAT (*mstat-list*)

specifies a list of model fit statistics to be displayed. A set of statistics is displayed by default, but you can use this option to specify a custom set of model statistics.

The *mstat-list* can contain one or more of the following keywords:

ALL	displays all model fit statistics. Not all statistics are appropriate for all models, and thus not always calculated. A blank cell in the table indicates that the statistic is not appropriate for that model.
DFE	displays the error degrees of freedom. This statistic is displayed by default.
F	displays the <i>F</i> statistic of the overall test for no fixed effects.
FNUMDF	displays the numerator degrees of freedom of the overall test for no fixed effects.
FDENDF	displays the denominator degrees of freedom of the overall test for no fixed effects.
M	displays the Hausman test <i>m</i> statistic.
MDF	displays the Hausman test degrees of freedom.

MSE	displays the model mean square error. This statistic is displayed by default.
NCS	displays the number of cross sections. This statistic is displayed by default.
NOBS	displays the number of observations. This statistic is displayed by default.
NONE	suppresses the table of model fit statistics when specified alone, and is ignored when specified with other options.
NTS	displays the maximum length of the time series. This statistic is displayed by default.
PROBF	displays the significance level of the overall test for no fixed effects.
PROBM	displays the significance level of the Hausman test.
RMSE	displays the model root mean square error.
RSQUARE	displays the model R-square fit statistic. This statistic is displayed by default.
SSE	displays the model sum of squares.
VARCS	displays the variance component that corresponds to cross sections in random-effects models.
VARERR	displays the variance component that corresponds to error in random-effects models.
VARTS	displays the variance component that corresponds to time series in random-effects models.

OUTPARM=libref.data-table

names an output data table to contain the data from the comparison table for parameter estimates, standard errors, and *t* tests. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 465.

OUTSTAT=libref.data-table

names an output data table to contain the data from the comparison table for model fit statistics, such as R-square and mean square error. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 465.

PSTAT (pstat-list)

specifies a list of parameter statistics to be displayed. By default, estimated regression coefficients and their standard errors are displayed. Use this option to specify a custom set of parameter statistics.

pstat-list can contain one or more of the following keywords:

ALL	displays all parameter statistics.
ESTIMATE	displays the estimated regression coefficient. This statistic is displayed by default.
NONE	suppresses the table of parameter statistics when specified alone, and is ignored when specified with other options.
STDERR	displays the standard error. This statistic is displayed by default.
PROBT	displays the significance level of the <i>t</i> test.

T displays the t statistic.

For a demonstration of the COMPARE statement, see [Example 11.2](#).

CORRELATED Statement

CORRELATED *effects* ;

The CORRELATED statement selects the subset of effects from the **MODEL** statement that are correlated with the individual (or cross-sectional) error term. This statement is valid only when you specify the **AMACURDY** or **HTAYLOR** model estimation method in the MODEL statement.

If you specify an *effect* in the CORRELATED statement but not in the MODEL statement, then the *effect* is added to the list of model effects and treated as correlated with the cross-sectional error term. If you have multiple MODEL statements, then each CORRELATED statement is associated with the MODEL statement that immediately precedes it.

The following statements fit a Hausman and Taylor (1981) model with model effects X1, X2, Z1, and Z2. The effects X2 and Z2 are correlated with the cross-sectional (firm-level) error term.

```
proc cpanel data = mylib.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / htaylor;
  correlated X2 Z2;
run;
```

You can also specify classification variables, interactions, and nested effects. For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 91 in Chapter 4, “[Shared Concepts](#).”

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 532. A path is a table name that is prefixed with dot-separated grouping information. For example, a

SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying */tions/* selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying *!/tions/* selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 532. You cannot specify the ODS table named *OutputCasTables* in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

ENDOGENOUS Statement

ENDOGENOUS *effects* </ option> ;

The ENDOGENOUS statement declares a set of *effects* that are treated as endogenous during estimation, and is valid only when you perform either instrumental variables regression or dynamic panel estimation.

Instrumental Variables (IV) Regression

You perform instrumental variables (IV) regression by specifying one or more of the **IVBTWNG**, **IVFIXONE**, **IVFIXONETIME**, **IVFIXTWO**, **IVG2SLS**, **IVPOOLED**, **IVRANONE**, and **IVRANTWO** options in the **MODEL** statement.

When you use the ENDOGENOUS statement for IV regression, it selects the subset of effects from the **MODEL** statement that are endogenous—that is, the effects that are correlated with the observation-level error term.

If you specify an *effect* in the ENDOGENOUS statement but not in the **MODEL** statement, then the *effect* is added to the list of model effects and treated as endogenous. If you have multiple **MODEL** statements, then each ENDOGENOUS statement is associated with the **MODEL** statement that immediately precedes it.

The ENDOGENOUS statement is used in conjunction with the **INSTRUMENTS** statement to perform IV regression. The following statements perform IV fixed-effects regression with the model effects X1, X2, Z1, and Z2. The effects X2 and Z2 are endogenous, and G1 and G2 are external instruments.

```
proc cpanel data = mylib.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone;
  endogenous X2 Z2;
  instruments G1 G2;
run;
```

Dynamic Panel Estimation

You perform dynamic panel estimation by specifying the **DYNDIFF** or **DYNSYS** option in the **MODEL** statement.

When you use the ENDOGENOUS statement in dynamic panel estimation, it specifies a set of variables whose instruments follow a prescribed form that is appropriate for endogenous regressors.

If you have multiple MODEL statements, then each ENDOGENOUS statement is associated with the MODEL statement that immediately precedes it.

You can specify the following *option* after a slash (/):

EQ=BOTH | DIFF | LEVEL

specifies which equations include the endogenous instruments. You can specify the following values:

- BOTH** includes the endogenous instruments in both the difference and level equations.
- DIFF** includes the endogenous instruments only in the difference equations.
- LEVEL** includes the endogenous instruments only in the level equations.

By default, EQ=BOTH.

The following statements perform dynamic panel estimation by using GMM-style instruments for the endogenous variable *Sales* and standard instruments for the exogenous variable *Price*. Both sets of instruments are included only in the difference equations.

```
proc cpanel data = mylib.a;
  id firm year;
  model Sales = Price / dyndiff;
  endogenous Sales / eq = diff;
  instruments Price / eq = diff;
run;
```

For both IV regression and dynamic panel estimation, you can also specify classification variables, interactions, and nested effects in the ENDOGENOUS statement. For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 91 in Chapter 4, “[Shared Concepts](#).”

ID Statement

ID *cross-section-id* < *time-series-id* > ;

The ID statement specifies variables in the input data table that identify the cross section and time period for each observation. Sometimes estimation does not require a time variable, in which case specifying the *time-series-id* variable is optional.

INSTRUMENTS Statement

INSTRUMENTS *effects* </ *option* > ;

The INSTRUMENTS statement specifies instrumental variables and effects. It is valid only when you perform either instrumental variables regression or dynamic panel estimation.

Instrumental Variables (IV) Regression

You perform instrumental variables (IV) regression by specifying one or more of the [IVBTWNG](#), [IVFIXONE](#), [IVFIXONETIME](#), [IVFIXTWO](#), [IVG2SLS](#), [IVPOOLED](#), [IVRANONE](#), and [IVRANTWO](#) options in the MODEL statement.

The effects that you specify in the INSTRUMENTS statement are auxiliary effects external to those that you specify in the MODEL statement. If you have multiple MODEL statements, then each INSTRUMENTS statement is associated with the MODEL statement that immediately precedes it.

The INSTRUMENTS statement is used in conjunction with the ENDOGENOUS statement to perform IV regression. The following statements perform IV fixed-effects regression with the model effects X1, X2, Z1, and Z2. The effects X2 and Z2 are endogenous, and G1 and G2 are external instruments.

```
proc cpanel data = mylib.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone;
  endogenous X2 Z2;
  instruments G1 G2;
run;
```

Dynamic Panel Estimation

You perform dynamic panel estimation by specifying the [DYNDIFF](#) or [DYNSSYS](#) options in the MODEL statement.

When you use the INSTRUMENTS statement in dynamic panel estimation, it specifies a set of variables to be included as standard instruments.

If you have multiple MODEL statements, then each INSTRUMENTS statement is associated with the MODEL statement that immediately precedes it.

You can specify the following *option* after a slash (/):

EQ=BOTH | DIFF | LEVEL

specifies which equations include the standard instruments. You can specify the following values:

- BOTH** includes the standard instruments in both the difference and level equations.
- DIFF** includes the standard instruments only in the difference equations.
- LEVEL** includes the standard instruments only in the level equations.

By default, EQ=BOTH.

The following statements perform dynamic panel estimation by using GMM-style instruments for the endogenous variable Sales and standard instruments for the exogenous variable Price. Both sets of instruments are included only in the difference equations.


```

proc cpanel data = mylib.a;
  id firm year;
  model Sales = Price / dyndiff;
  endogenous Sales / eq = diff;
  instruments Price / eq = diff;
run;

```

For both IV regression and dynamic panel estimation, you can also specify classification variables, interactions, and nested effects in the INSTRUMENTS statement. For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 91 in Chapter 4, “[Shared Concepts](#).”

MODEL Statement

MODEL *response* = < *effects* > < / *options* > ;

The MODEL statement specifies the regression model, the error structure that is assumed for the regression residuals, and the estimation technique to be used. The response variable (*response*) on the left side of the equal sign is regressed on the independent variables (*effects*), which are listed after the equal sign. You can specify any number of MODEL statements. For each MODEL statement, you can specify only one *response*.

For information about constructing the model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 91 in Chapter 4, “[Shared Concepts](#).”

You can label models. Model labels are used in the printed output to identify the results for different models. If you do not specify a label, the model is referred to by numerical order wherever necessary. You can label models by prefixing the MODEL statement by a label followed by a colon as follows:

label: **MODEL** ...;

The MODEL statement supports many options, some more specific than others. [Table 11.3](#) summarizes the *options* available in the MODEL statement. These are subsequently discussed in detail in the order in which they are presented in the table.

Table 11.3 Summary of MODEL Statement Options

Option	Description
Estimation Technique Options	
AMACURDY	Fits a one-way model by using the Amemiya-MaCurdy estimator
BTWNG	Fits the between-groups model
BTWNT	Fits the between-time-periods model
DYNDIFF	Fits a dynamic panel model by using first-differences GMM
DYNSYS	Fits a dynamic panel model by using system GMM
FDONE	Fits a one-way model by using first differences
FIXONE	Fits a one-way fixed-effects model
FIXONETIME	Fits a one-way fixed-effects model for time effects
FIXTWO	Fits a two-way fixed-effects model

Table 11.3 *continued*

Option	Description
HTAYLOR	Fits a one-way model by using the Hausman-Taylor estimator
IVBTWNG	Fits a between-groups regression model by using instrumental variables
IVFIXONE	Fits a one-way fixed-effects model by using instrumental variables
IVFIXONETIME	Fits a one-way time fixed-effects model for time effects by using instrumental variables
IVFIXTWO	Fits a two-way fixed-effects model by using instrumental variables
IVG2SLS	Fits a generalized two-stage least squares model by using instrumental variables
IVPOOLED	Fits the pooled regression model by using instrumental variables
IVRANONE	Fits a one-way random-effects model by using instrumental variables
IVRANTWO	Fits a two-way random-effects model by using instrumental variables
POOLED	Fits the pooled regression model
RANONE	Fits a one-way random-effects model
RANTWO	Fits a two-way random-effects model
Estimation Control Options	
BIASCORRECTED	Computes bias-corrected two-step GMM variances for instrumental variables models and dynamic panel models
GMM=	Specifies one-step or two-step GMM for instrumental variables models and dynamic panel models
NOINT	Suppresses the intercept
VCOMP=	Specifies the type of variance component estimation for random-effects estimation
Dynamic Panel Estimation Options	
ARTESTS=	Specifies the maximum lag of $AR(m)$ tests
DLAGS=	Specifies the number of dependent variable lags to include
GINV=	Specifies the generalized inverse method
MAXBAND=	Limits the number of instruments
Alternative Variances Options	
CLUSTER	Corrects covariance for intracluster correlation
HAC	Specifies a heteroscedasticity- and autocorrelation-consistent (HAC) covariance

Table 11.3 *continued*

Option	Description
HCCME=	Specifies a heteroscedasticity-corrected covariance matrix estimator (HCCME)
MAXBAND=	Limits the number of instruments
ROBUST	Computes robust variances
Printed Output Options	
CORRB	Prints the parameter correlation matrix
COVB	Prints the parameter covariance matrix
NOLABEL	Suppresses variable labels
PRINTFIXED	Estimates and prints the fixed effects

You can specify the following *options* after a slash (/).

Estimation Technique Options

These options specify the assumed error structure and estimation method. You can specify more than one option, in which case the analysis is repeated for each. The default is FIXONE (one-way fixed effects).

All estimation methods are described in detail in the section “[Details: CPANEL Procedure](#)” on page 491.

AMACURDY

requests Amemiya-MaCurdy estimation for a model that has correlated individual (cross-sectional) effects. You specify the correlated effects by using the CORRELATED statement.

BTWNG

estimates a between-groups model.

BTWNT

estimates a between-time-periods model.

DYNDIFF

estimates a dynamic panel by using the generalized method of moments (GMM) on equations that are formed by first differencing.

DYNSYS

estimates a dynamic panel by using GMM on the system that combines first-differenced equations and level equations.

FDONE

estimates a one-way model by using first-differenced methods.

FIXONE

estimates a one-way fixed-effects model that corresponds to cross-sectional effects only.

FIXONETIME

estimates a one-way fixed-effects model that corresponds to time effects only.

FIXTWO

estimates a two-way fixed-effects model.

HTAYLOR

requests Hausman-Taylor estimation for a model that has correlated individual (cross-sectional) effects. You specify the correlated effects by using the `CORRELATED` statement.

IVBTWNG

requests instrumental variables regression for between-groups estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVFIXONE

requests instrumental variables regression for one-way fixed-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVFIXONETIME

requests instrumental variables regression for one-way time fixed-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVFIXTWO

requests instrumental variables regression for two-way fixed-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVG2SLS

requests instrumental variables regression for generalized two-stage least squares estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVPOOLED

requests instrumental variables regression for pooled regression. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVRANONE

requests instrumental variables regression for one-way random-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the **ENDOGENOUS** statement, and you specify external instruments by using the **INSTRUMENTS** statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVRANTWO

requests instrumental variables regression for two-way random-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the **ENDOGENOUS** statement, and you specify external instruments by using the **INSTRUMENTS** statement. You can choose to use two-stage least squares or GMM to estimate the model.

POOLED

estimates a pooled (OLS) model.

RANONE

estimates a one-way random-effects model.

RANTWO

estimates a two-way random-effects model.

Estimation Control Options

These options define parameters that control the estimation and can be specific to the chosen technique (for example, how to estimate variance components in a random-effects model).

BIASCORRECTED

computes the bias-corrected covariance matrix of the two-step generalized method of moments (GMM) estimator. This statement is valid only when you perform either dynamic panel estimation or instrumental variables regression.

CLUSTER

specifies the cluster correction for the covariance matrix. You can specify this option when you specify **HCCME=0, 1, 2, or 3**.

GMM=ONESTEP | TWOSTEP

specifies the number of GMM stages to use in the estimation. This statement is valid only when you perform either dynamic panel estimation or instrumental variables regression.

You can specify the following values:

ONESTEP	uses one-step GMM, which is computationally simple but dependent on model assumptions.
TWOSTEP	uses two-step GMM, which is computationally intensive but more robust to violations of model assumptions.

By default, **GMM=ONESTEP**.

The following statements perform two-step GMM in a dynamic panel model, which is requested by the **DYNDIFF** option in the **MODEL** statement. You can use similar statements for a dynamic panel estimation that uses a full system of difference and level equations by specifying the **DYNSYS** option

instead of the DYNDIFF option in the MODEL statement. The endogenous variable is *Sales*, and the GMM-style instruments is the predetermined variable *Price*. Both sets of instruments are included only in the difference equations.

```
proc cpanel data = mylib.a;
  id firm year;
  model Sales = Price / dyndiff GMM = TWOSTEP;
  endogenous Sales / eq = diff;
  predetermined Price / eq = diff;
run;
```

The following statements perform two-step GMM in an instrumental variables model. The endogenous variables are *X2* and *Z2*, and the external instruments are *G1* and *G2*.

```
proc cpanel data = mylib.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone GMM = TWOSTEP;
  endogenous X2 Z2;
  instruments G1 G2;
run;
```

The IVFIXONE option in the MODEL statement requests IV one-way fixed-effects estimation. You can request other types of IV estimation by specifying IVBTWNG, IVFIXONETIME, IVFIXTWO, IVG2SLS, IVPOOLED, IVRANONE, or IVRANTWO instead of IVFIXONE in the MODEL statement.

HAC <(options)>

specifies the heteroscedasticity- and autocorrelation-consistent (HAC) covariance matrix estimator. This option is not available for between-groups or between-time-periods models, and cannot be combined with the HCCME= option.

You can specify the following *options* within parentheses and separated by spaces:

ADJUSTDF

makes a small-sample adjustment to the degrees of freedom in the covariance calculation.

BANDWIDTH=*number* | *method*

specifies the fixed bandwidth value or bandwidth selection method to be used in the kernel function. You can specify either a fixed value (*number*) or one of the *methods* listed after *number*.

number

specifies a fixed value of the bandwidth parameter.

ANDREWS | **AN**

specifies the Andrews (1991) bandwidth selection method.

NEWWEYWEST<(C=number)> | NW <(C=number)>

specifies the bandwidth selection method of Newey and West (1994) You can also specify *C=number* to calculate the lag selection parameter; by default, *C=12*.

SAMPLESIZE<(options)> | SS<(options)>

calculates the bandwidth according to the following equation based on the sample size,

$$b = \gamma T^r + c$$

where *b* is the bandwidth parameter; *T* is the sample size; and γ , *r*, and *c* are values that you specify using the following *options* within parentheses and separated by spaces:

CONSTANT=number

specifies the constant *c* in the equation. By default, *CONSTANT=0.5*.

GAMMA=number

specifies the coefficient γ in the equation. By default, *GAMMA=0.75*.

INTSS

specifies that the bandwidth parameter be an integer; that is, $b = \lfloor \gamma T^r + c \rfloor$, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to *x*.

RATE=number

specifies the growth rate *r* in the equation. By default, *RATE=0.3333*.

By default, *BANDWIDTH=ANDREWS*.

KERNEL=BARTLETT | PARZEN | QS | TH | TRUNCATED

specifies the type of kernel function. You can specify the following values:

BARTLETT	specifies the Bartlett kernel function.
PARZEN	specifies the Parzen kernel function.
QS	specifies the quadratic spectral kernel function.
TH	specifies the Tukey-Hanning kernel function.
TRUNCATED	specifies the truncated kernel function.

By default, *KERNEL=QS*.

KERNELLB=number

specifies the lower bound of the kernel weight value. Any kernel weight less than *number* is regarded as 0, which accelerates the calculation in large samples, especially for the quadratic spectral kernel function. By default, *KERNELLB=0*.

PREWHITENING

requests prewhitening in the covariance calculation.

The following statements perform one-way fixed-effects estimation by using the Newey-West bandwidth and a Bartlett kernel based on a prewhitened gradient vector:

```
proc cpanel data = mylib.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone gmm=twostep
  hac(bandwidth=neweywest(c=12) kernel=bartlett prewhitening);
run;
```

HCCME=0 | 1 | 2 | 3

specifies the type of adjustment to the HCCME covariance matrix. You can specify the values 0 through 3, which indicate the type of covariance adjustment.

NOINT

suppresses the intercept parameter from the model.

ROBUST

computes the robust covariance matrix. This option can be applied to dynamic panel models, instrumental variables regression models, and static panel models. For dynamic panel models and instrumental variables regression models, the option provides heteroscedasticity-corrected standard errors according to the GMM estimation method. The following statements perform IV fixed-effects regression with robust standard errors:

```
proc cpanel data = mylib.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone gmm=twostep robust;
  endogenous X2 Z2;
  instruments G1 G2;
run;
```

For static panel models, this option produces cross-sectional clustered standard errors when HCCME=0. The following statements perform one-way fixed-effects regression with clustered standard errors:

```
proc cpanel data = mylib.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / fixone robust;
run;
```

VCOMP=FB | NL | SA | WH | WK

specifies the type of variance component estimate to use. You can specify the following values:

- FB** uses the Fuller-Battese method.
- NL** uses the Nerlove method.
- SA** uses the Swamy-Arora method.
- WH** uses the Wallace-Hussain method.
- WK** uses the Wansbeek-Kapteyn method.

By default, VCOMP=SA.

Dynamic Panel Estimation Options

These options are specific to dynamic panel estimation, which you obtain by specifying the DYNDIFF or DYNSSYS option in the MODEL statement.

ARTESTS=integer

specifies the maximum order of the test for the presence of autoregression (AR) effects. By default, ARTESTS=2.

DLAGS=integer

specifies the number of dependent-variable lags to use as regressors. By default, DLAGS=1.

GINV=G2 | G4

specifies what type of generalized inverse to use. You can specify the following values:

G2 uses the G2 generalized inverse.

G4 uses the G4 generalized inverse.

The difference between G2 and G4 becomes evident when you invert singular matrices. The G2 generalized inverse drops rows and columns from singular matrices to produce a viable inverse. The G4 inverse, on the other hand, is the Moore-Penrose generalized inverse, which averages the variance effects between collinear rows. The G4 inverse is usually more stable, but it is computationally intensive. By default, GINV=G2. If you have trouble reproducing published results, often the solution is to switch to GINV=G4.

MAXBAND=integer

if specified, sets the maximum number of GMM-style instruments per observation, for each variable. Because the number of GMM-style instruments grows quadratically with the number of time periods, this option makes estimation more feasible when you have many time periods.

Printed Output Options

These options alter how results are presented.

CORRB

prints the matrix of estimated correlations between the parameter estimates.

COVB

prints the matrix of estimated covariances between the parameter estimates.

NOLABEL

suppresses variable labels from the printed output.

PRINTFIXED

estimates and prints the fixed effects in models where they would normally be absorbed within the estimation.

OUTPUT Statement

OUTPUT OUT=*libref.data-table* < options > ;

The OUTPUT statement creates a data table that contains observationwise statistics that PROC CPANEL computes after fitting the model. In order to avoid data duplication for large data tables, only the identification and dependent variables are included in the output data table unless you specify additional variables in the **COPYVAR=** option. To avoid further data duplication, the OUTPUT statement is allowed only with MODEL statements that request a single estimation. If you have multiple MODEL statements, then each OUTPUT statement is associated with the MODEL statement that immediately precedes it.

You must specify the following option:

OUT=*libref.data-table*

names the output data table for PROC CPANEL to use. You must specify this option before any other options. *libref.data-table* is a two-level name, where

<i>libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>libref</i> , see the section “Using CAS Sessions and CAS Engine Librefs” on page 465.
---------------	--

<i>data-table</i>	specifies the name of the output data table.
-------------------	--

You can also specify the following options:

COPYVAR=*variable*

COPYVARS=(*variables*)

transfers one or more *variables* from the input data table to the output data table.

PREDICTED=*name*

P=*name*

writes the predicted values to the output data table.

RESIDUAL=*name*

R=*name*

writes the residuals to the output data table.

PREDETERMINED Statement

PREDETERMINED *effects* < / *option* > ;

The PREDETERMINED statement specifies a set of variables whose instruments follow a prescribed form that is appropriate for predetermined regressors and is valid only when you perform dynamic panel estimation by specifying the **DYNDIFF** or **DYNSYS** option in the MODEL statement.

If you have multiple MODEL statements, then each PREDETERMINED statement is associated with the MODEL statement that immediately precedes it.

You can specify the following *option* after a slash (/):

EQ=BOTH | DIFF | LEVEL

specifies which equations include the predetermined instruments. You can specify the following values:

BOTH includes the predetermined instruments in both the difference and level equations.

DIFF includes the predetermined instruments only in the difference equations.

LEVEL includes the predetermined instruments only in the level equations.

By default, EQ=BOTH.

The following statements perform dynamic panel estimation by using GMM-style instruments for the endogenous variable Sales and GMM-style instruments for the predetermined variable Price. Both sets of instruments are included only in the difference equations.

```
proc cpanel data = mylib.a;
  id firm year;
  model Sales = Price / dyndiff;
  endogenous Sales / eq = diff;
  predetermined Price / eq = diff;
run;
```

You can also specify classification variables, interactions, and nested effects in the PREDETERMINED statement. For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 91 in Chapter 4, “[Shared Concepts](#).”

RESTRICT Statement

RESTRICT *equation* < , *equation2* . . . > ;

The RESTRICT statement specifies linear equality restrictions on the parameters in the previous MODEL statement. There can be as many unique restrictions as the number of parameters in the preceding MODEL statement. Multiple RESTRICT statements are understood as joint restrictions on a model’s parameters. Restrictions on the intercept are obtained by the use of the INTERCEPT keyword. RESTRICT statements before the first MODEL statement are automatically associated with the first MODEL statement, in addition to any RESTRICT statements that follow it but precede subsequent MODEL statements.

Currently, only linear equality restrictions are permitted in PROC CPANEL. Tests and restriction expressions can be composed of only algebraic operations that involve the addition symbol (+), subtraction symbol (–), and multiplication symbol (*).

The following statements illustrate the use of the RESTRICT statement:

```
proc cpanel data = mylib.a;
  id csid tsid;
  model y = x1 x2 x3;
  restrict x1 = 0, x2 * .5 + 2 * x3 = 0;
  restrict intercept = 1;
run;
```

If you are fitting a dynamic panel model, you can place restrictions on lags of the dependent variable by referencing the name of the dependent variable, followed by an underscore and the lag order. For example:

```
proc cpanel data = mylib.a;
  model sales = price / dyndiff;
  restrict sales_1 = 0.5;
run;
```

Note that a RESTRICT statement cannot include a division sign in its formulation.

TEST Statement

TEST *equation <, equation2... > < /options> ;*

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the regression parameters in the preceding MODEL statement. Like RESTRICT statements, TEST statements before the first MODEL statement are automatically associated with the first MODEL statement, in addition to any TEST statements that follow it but precede subsequent MODEL statements. Each equation specifies a linear hypothesis to be tested. All hypotheses in one TEST statement are tested jointly. Variable names in the equations must correspond to regressors in the preceding MODEL statement, and each name represents the coefficient of the corresponding regressor. The INTERCEPT keyword refers to the coefficient of the intercept.

You can specify the following *options* after a slash (/):

ALL

specifies Lagrange multiplier, likelihood ratio, and Wald tests.

LM

specifies the Lagrange multiplier test.

LR

specifies the likelihood ratio test.

WALD

specifies the Wald test.

The Wald test is performed by default.

The following statements illustrate the use of the TEST statement:

```
proc cpanel data = mylib.a;
  id csid tsid;
  model y = x1 x2 x3;
  test x1 = 0, x2 * .5 + 2 * x3 = 0 / all;
  test intercept = 0, x3 = 0;
run;
```

The first test investigates the joint hypothesis that $\beta_1 = 0$ and $0.5\beta_2 + 2\beta_3 = 0$.

Currently, only linear equality restrictions and tests are permitted in PROC CPANEL. Tests and restriction expressions can be composed only of algebraic operations that involve the addition symbol (+), subtraction symbol (−), and multiplication symbol (*).

If you are fitting a dynamic panel model, you can perform tests on lags of the dependent variable by referencing the name of the dependent variable, followed by an underscore and the lag order. For example:

```
proc cpanel data = mylib.a;
  model sales = price / dyndiff;
  test sales_1 = 0.5 / wald;
run;
```

When you use the TEST statement along with the MODEL statement options [AMACURDY](#), [DYNDIFF](#), [DYNSSYS](#), [HTAYLOR](#), [IVBTWNG](#), [IVFIXONE](#), [IVFIXONETIME](#), [IVFIXTWO](#), [IVG2SLS](#), [IVPOOLED](#), [IVRANONE](#), and [IVRANTWO](#), only the Wald test is available.

Details: CPANEL Procedure

Specifying the Input Data

Panel data are identified by both a cross section identification (ID) variable and a time variable. Suppose that cross sections are identified by the variable *State* and time periods are identified by the variable *Date*. You specify the cross section and time series variables in an ID statement. The following statements show the appropriate syntax:

```
proc panel data = mylib.a;
  id State Date;
  model ...;
run;
```

For some estimation strategies (such as one-way fixed effects), time is not relevant to the estimation. In these cases, specifying the time variable is optional. The CPANEL procedure alerts you if do not specify a time variable when one is required.

When taking lags or first differences, the CPANEL procedure determines that observations are adjacent in time only if their time values are exactly one unit apart. For this reason, you should index time by using a variable that contains consecutive integers.

PROC CPANEL performs its estimations regardless of how the data are partitioned logically across machines, but it might temporarily repartition the data to perform computations more efficiently. You can specify the

NOPART option in the PROC CPANEL statement to bypass this repartitioning, at the expense of slower execution.

Specifying the Regression Model

The CPANEL procedure is similar to other regression procedures in SAS software. Suppose you want to regress the variable *Y* on regressors *X1* and *X2*. You specify the dependent variable first, followed by an equal sign, followed by the list of regression variables, as shown in the following statements:

```
proc cpanel data mylib.a;
  id state date;
  model y = x1 x2;
run;
```

One advantage of using PROC CPANEL is that you can incorporate a model for the structure of the error terms. It is important to consider what type of model is appropriate for your data and to specify the corresponding option in the MODEL statement. The following model estimation options are supported: POOLED, BTWNG, BTWNT, FIXONE, FDONE, FIXTWO, RANONE, RANTWO, HTAYLOR, AMACURDY, IVPOOLED, IVBTWNG, IVFIXONE, IVFIXONETIME, IVFIXTWO, IVG2SLS, IVRANONE, IVRANTWO, DYNDIFF, and DYN SYS. The methods that underlie these estimation options are described, in order, beginning with the section “[Pooled Regression \(POOLED Option\)](#)” on page 494.

The following statements fit a one-way random-effects model with variance components estimated by the Fuller-Battese (FB) method:

```
proc cpanel data mylib.a;
  id State Date;
  model Y = X1 X2 / ranone vcomp = fb;
run;
```

You can specify more than one estimation option in the MODEL statement, and the analysis is repeated for each specified method. You can use multiple MODEL statements to estimate different regression models or to estimate the same model by different methods.

You can also specify classification variables, interactions, and nested effects, as demonstrated by the following statements:

```
proc cpanel data mylib.a;
  id State Date;
  class C1 C2 C3;
  model Y = X1 X2 C1 C2(C1) C3 C1*C3 / ranone vcomp = fb;
run;
```

For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 91 in Chapter 4, “[Shared Concepts](#).”

Missing Values

Any observation in the input data table that has a missing value for the cross-section ID, time series ID, dependent variable, or any model effect is ignored by PROC CPANEL when it fits the model.

If there are observations in your data where only the dependent variable is missing, you can still compute predicted values for these observations and store them in an output data table by using the OUTPUT statement.

Unbalanced Data

Unbalanced data occur when not all time values are observed for all cross sections or, when time is not part of the estimation, the cross sections are not all the same size.

Whether the data are unbalanced by design or because of missing values, almost all the methods supported by the CPANEL procedure take proper account of the unbalanced data. The one exception is the AMACURDY method, which is suitable only for balanced data.

Common Notation

This section presents notation that is common to all subsequent sections. Consider the panel regression:

$$y_{it} = \alpha + \sum_{k=1}^K x_{itk} \beta_k + u_{it} \quad i = 1, \dots, N; t = 1, \dots, T_i$$

The total number of observations is $M = \sum_{i=1}^N T_i$. For balanced data, $T_i = T$ for all i . For unbalanced data, define T to be the number of unique time periods.

The exact representation of u_{it} and the underlying assumptions depend on the estimation method.

In matrix notation the model is

$$y_{it} = \alpha + \mathbf{x}_{it} \boldsymbol{\beta} + u_{it}$$

where $\mathbf{x}_{it}$ is a $1 \times K$ row vector of independent variables and $\boldsymbol{\beta}$ is the $K \times 1$ vector of coefficients. Let $\mathbf{y}$ and $\mathbf{X}$ be matrices that are formed by arranging the dependent and independent variables by cross section, and by time within each cross section. Let $\mathbf{X}_\alpha$ be the $\mathbf{X}$ matrix augmented by a first column of ones, which corresponds to the intercept term α .

Define the following utility matrices:

$\mathbf{I}_p$ is an identity matrix of dimension p .

$\mathbf{j}_p$ is a $p \times 1$ column vector of ones.

$\mathbf{J}_p = \mathbf{j}_p \mathbf{j}_p'$ is a matrix of ones of dimension p .

$\bar{\mathbf{J}}_p = p^{-1} \mathbf{J}_p$.

$$\mathbf{E}_p = \mathbf{I}_p - \bar{\mathbf{J}}_p.$$

In the following sections, the panel data are assumed to be unbalanced unless otherwise indicated. If the data are balanced, the formulas reduce appropriately.

Pooled Regression (POOLED Option)

You perform pooled regression by specifying the POOLED option in the MODEL statement. Pooled regression is standard ordinary least squares (OLS) regression without any cross-sectional or time effects. The error structure is simply $u_{it} = e_{it}$, where the e_{it} are independently and identically distributed (iid) with zero mean and variance σ_e^2 .

Between-Groups Regression (BTWNG and BTWNT Options)

You perform between-groups regression by specifying the BTWNG option in the MODEL statement. When the data are balanced, between-groups regression is ordinary least squares (OLS) regression performed on data that have been collapsed into cross-sectional means. When the data are unbalanced, the regression is weighted to assign more importance to larger cross sections. The cross-sectional means are multiplied by the weighting factor $w_i = \sqrt{T_i/\bar{T}}$, where $\bar{T} = M/N$ is the average cross section size.

The BTWNT option works similarly, except that the data are collapsed by time period instead of by cross section.

One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)

You perform one-way fixed-effects estimation by specifying the FIXONE option in the MODEL statement. The error structure for the one-way fixed-effects model is

$$u_{it} = v_i + e_{it}$$

where the v_i are nonrandom parameters that are restricted to sum to 0, and the e_{it} are iid with zero mean and variance σ_e^2 .

The fixed-effects model can be estimated by ordinary least squares (OLS), treating the v_i as coefficients on dummy variables that identify the cross sections. However, when N is large, you might want to estimate only β and not v_i .

Let $\mathbf{Q}_0 = \text{diag}(\mathbf{E}_{T_i})$. The matrix $\mathbf{Q}_0$ represents the *within transformation*, the conversion of the raw data to deviations from a cross section's mean. Let $\mathbf{X}_w = \mathbf{Q}_0\mathbf{X}$ and $\mathbf{y}_w = \mathbf{Q}_0\mathbf{y}$. The within estimator of β is

$$\hat{\beta}_w = (\mathbf{X}_w' \mathbf{X}_w)^{-1} \mathbf{X}_w' \mathbf{y}_w$$

The previous estimation does not involve the intercept term because $\hat{\beta}_w$ is the same whether or not the intercept α is included in the model.

Standard errors, t statistics, and fit statistics such as mean square error (MSE) are all equivalent to those obtained from OLS regression of $\mathbf{y}_w$ on $\mathbf{X}_w$. The only exception is the error degrees of freedom, which equals $M - N - K$ to account for the tacit estimation of the N fixed effects.

The estimate of the intercept is

$$\hat{\alpha} = \bar{y}_{..} - \bar{x}_{..}\hat{\beta}_w$$

where $\bar{y}_{..}$ is the overall mean of y_{it} and $\bar{x}_{..}$ is the overall mean of x_{it} .

Fixed effects are not estimated as part of the regression, but they can be obtained by specifying the PRINTFIXED option in the MODEL statement. Each fixed effect is estimated as

$$\hat{v}_i = \bar{y}_{i.} - \bar{x}_{i.}\hat{\beta}_w$$

where $\bar{y}_{i.}$ and $\bar{x}_{i.}$ are cross-sectional means.

Variance estimates of $\hat{\alpha}$ and $\hat{v}_i$ are obtained by the delta method. If you specify the NOINT option, then the $\hat{v}_i$ are shifted to absorb $\hat{\alpha}$.

The FIXONETIME option works similarly, except that the data are grouped by time period instead of by cross section.

One-Way Fixed-Effects Model, First Differencing (FDONE Option)

You perform one-way fixed-effects estimation via first differencing by specifying the FDONE option in the MODEL statement. The method of first differencing offers an alternative to the within estimator $\hat{\beta}_w$. Consider the following one-way fixed-effects model:

$$y_{it} = \alpha + x_{it}\beta + v_i + e_{it}$$

For this model, the fixed effects are removed by subtracting first-order lags from both sides of the equation:

$$y_{it} - y_{i,t-1} = (x_{it} - x_{i,t-1})\beta + (e_{it} - e_{i,t-1})$$

Define $\Delta y_{it} = y_{it} - y_{i,t-1}$ and $\Delta x_{it} = x_{it} - x_{i,t-1}$, for $i = 1, \dots, N$ and $t = 2, \dots, T_i$. You obtain the first-differenced estimator, $\hat{\beta}_d$, and its variance by performing OLS regression of Δy_{it} on Δx_{it} .

You cannot obtain the error variance directly from the OLS regression because that regression is based on the differenced residual $e_{it} - e_{i,t-1}$. Instead, you can estimate σ_e^2 by $\hat{\sigma}_e^2 = \text{SSE}/(M - N - K)$, where

$$\text{SSE} = \sum_{i=1}^N \sum_{t=1}^{T_i} (y_{it} - x_{it}\hat{\beta}_d - \hat{v}_i)^2$$

and $\hat{v}_i = \bar{y}_{i.} - \bar{x}_{i.}\hat{\beta}_d$.

Estimation and inference for α and v_i are identical to that described in the section “One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)” on page 494, with $\hat{\beta}_w$ replaced by $\hat{\beta}_d$.

Two-Way Fixed-Effects Model (FIXTWO Option)

You perform two-way fixed-effects estimation by specifying the `FIXTWO` option in the `MODEL` statement. The error specification for the two-way fixed-effects model is

$$u_{it} = v_i + \lambda_t + e_{it}$$

where the v_i and λ_t are nonrandom parameters to be estimated. Assume the identifiability restriction $\sum_t \lambda_t = 0$.

Estimation is similar to that for one-way fixed effects, for which a within transformation is used to convert the problem to OLS regression. For two-way models under the general case of unbalanced data, the within transformation is more complex.

Following Wansbeek and Kapteyn (1989) and Baltagi (2013, sec. 9.4), let $\mathbf{X}^*$ and $\mathbf{y}^*$ be versions of $\mathbf{X}$ and $\mathbf{y}$ whose rows are sorted by time period, and by cross section within each time period. With the data sorted in this manner, define $\mathbf{D}_N$ to be the $M \times N$ design matrix for cross sections. Each row of $\mathbf{D}_N$ contains a 1 in the column that corresponds to that observation's cross section, and 0s in the remaining columns. Similarly, define $\mathbf{D}_T$ to be the $M \times T$ design matrix for time periods. In balanced data, $\mathbf{D}_N = \mathbf{j}_T \otimes \mathbf{I}_N$ and $\mathbf{D}_T = \mathbf{I}_T \otimes \mathbf{j}_N$.

Define the following:

$$\begin{aligned} \Delta_N &= \mathbf{D}_N' \mathbf{D}_N & (N \times N) \\ \Delta_T &= \mathbf{D}_T' \mathbf{D}_T & (T \times T) \\ \mathbf{A} &= \mathbf{D}_T' \mathbf{D}_N & (T \times N) \\ \bar{\mathbf{D}} &= \mathbf{D}_T - \mathbf{D}_N \Delta_N^{-1} \mathbf{A}' & (M \times T) \\ \mathbf{Q} &= \Delta_T - \mathbf{A} \Delta_N^{-1} \mathbf{A}' & (T \times T) \\ \mathbf{P} &= \mathbf{I}_M - \mathbf{D}_N \Delta_N^{-1} \mathbf{D}_N' - \bar{\mathbf{D}} \mathbf{Q}^{-1} \bar{\mathbf{D}}' & (M \times M) \end{aligned}$$

The matrix $\mathbf{P}$ provides the two-way within transformation. If the data are balanced, this amounts to transforming any data value z_{it} to $z_{it} - \bar{z}_{i.} - \bar{z}_{.t} + \bar{z}_{..}$.

Applying the two-way within transformation means that you can use OLS regression of $\mathbf{P}\mathbf{y}^*$ on $\mathbf{P}\mathbf{X}^*$ to obtain $\hat{\boldsymbol{\beta}}_f$, $\text{Var}(\hat{\boldsymbol{\beta}}_f)$, and fit statistics such as mean-square error (MSE), provided that you adjust the error degrees of freedom to equal $M - N - T - K + 1$.

Define the residual vector $\mathbf{r}^* = \mathbf{y}^* - \mathbf{X}^* \hat{\boldsymbol{\beta}}_f$. Estimates of the time effects are $\hat{\boldsymbol{\lambda}} = \mathbf{Q}^{-1} \bar{\mathbf{D}}' \mathbf{r}^*$, and estimates of the cross-sectional effects are $\hat{\mathbf{v}} = (\Theta_1 - \Theta_2 + \Theta_3) \mathbf{r}^*$, where

$$\begin{aligned} \Theta_1 &= \Delta_N^{-1} \mathbf{D}_N' \\ \Theta_2 &= \Delta_N^{-1} \mathbf{A}' \mathbf{Q}^{-1} \mathbf{D}_T' \\ \Theta_3 &= \Delta_N^{-1} \mathbf{A}' \mathbf{Q}^{-1} \mathbf{A} \Delta_N^{-1} \mathbf{D}_N' \end{aligned}$$

If the model includes an intercept, then $\hat{\alpha} = \bar{y}_{..} - \bar{x}_{..} \hat{\boldsymbol{\beta}}_f$, and the $\hat{v}_i$ are shifted to exclude $\hat{\alpha}$.

Variance and covariance estimates for $\hat{\alpha}$, $\hat{\mathbf{v}}$, and $\hat{\boldsymbol{\lambda}}$ are obtained by the delta method, because each of these quantities is a linear transformation of $\mathbf{y}^*$ and $\hat{\boldsymbol{\beta}}_f$.

One-Way Random-Effects Model (RANONE Option)

You perform one-way random-effects estimation by specifying the RANONE option in the MODEL statement. The specification for the one-way random-effects model is

$$u_{it} = v_i + e_{it}$$

where the v_i are iid with zero mean and variance σ_v^2 , and the e_{it} are iid with zero mean and variance σ_e^2 . Furthermore, a random-effects specification assumes that the error terms are mutually uncorrelated and that each error term is uncorrelated with $\mathbf{X}$.

Estimation proceeds in two steps. First, you obtain estimates of the variance components σ_v^2 and σ_e^2 . The CPANEL procedure provides five methods for estimating variance components; these methods are described in the following subsections.

Second, with the variance components in hand, you form a weight for each cross section,

$$\hat{\theta}_i = 1 - \hat{\sigma}_e / \hat{w}_i$$

where $\hat{w}_i^2 = T_i \hat{\sigma}_v^2 + \hat{\sigma}_e^2$. Taking $\hat{\theta}_i$, you form the partial deviations:

$$\tilde{y}_{it} = y_{it} - \hat{\theta}_i \bar{y}_i.$$

$$\tilde{\mathbf{x}}_{\alpha,it} = \mathbf{x}_{\alpha,it} - \hat{\theta}_i \bar{\mathbf{x}}_{\alpha,i}.$$

The random-effects estimation is then the result of OLS regression on the transformed data.

The CPANEL procedure provides five options for estimating variance components, as described in the following subsections.

Swamy-Arora Method

The Swamy and Arora (1972) method is the default method of estimating variance components; it can also be specified as the VCOMP=SA option in the MODEL statement. The Swamy-Arora method is part of a class of methods known as analysis of variance (ANOVA) estimators.

ANOVA estimators obtain variance components by solving a system of equations that is based on expected sums of squares. The following quadratic forms correspond to the within and between sums of squares, respectively:

$$q_e = \mathbf{u}' \mathbf{Q}_0 \mathbf{u}$$

$$q_v = \mathbf{u}' \mathbf{P}_0 \mathbf{u}$$

In these equations, $\mathbf{Q}_0 = \text{diag}(\mathbf{E}_{T_i})$, $\mathbf{P}_0 = \text{diag}(\bar{\mathbf{J}}_{T_i})$, and $\mathbf{u}$ is the vector of true residuals.

The ANOVA methods differ only in how they estimate $\mathbf{u}$. The Swamy-Arora method substitutes into q_e the within residuals from one-way fixed-effects estimation and substitutes into q_v the residuals from between-groups regression.

The expected values of the quadratic forms are

$$E(\hat{\mathbf{u}}_w' \mathbf{Q}_0 \hat{\mathbf{u}}_w) = (M - N - K) \sigma_e^2$$

$$E(\hat{\mathbf{u}}_b' \mathbf{P}_0 \hat{\mathbf{u}}_b) = \left[M - \text{tr} \left\{ (\mathbf{X}'_{\alpha} \mathbf{P}_0 \mathbf{X}_{\alpha})^{-1} \mathbf{X}'_{\alpha} \mathbf{Z}_0 \mathbf{Z}'_0 \mathbf{X}_{\alpha} \right\} \right] \sigma_v^2 + (N - K - 1) \sigma_e^2$$

where $\mathbf{Z}_0 = \text{diag}(\mathbf{j}_{T_i})$.

Estimates of σ_v^2 and σ_e^2 are obtained by first setting the expected sums of squares to their observed values, and then solving.

Wallace-Hussain Method

You can use the Wallace and Hussain (1969) method of estimating variance components by specifying the VCOMP=WH option in the MODEL statement. The Wallace-Hussain method is an ANOVA method that uses the residuals from pooled (OLS) regression, $\hat{\mathbf{u}}_p$, in both quadratic forms.

The expected values of the quadratic forms are

$$E\left(\hat{\mathbf{u}}_p' \mathbf{Q}_0 \hat{\mathbf{u}}_p\right) = (d_1 - d_3)\sigma_v^2 + (M - N - K - 1 + d_2)\sigma_e^2$$

$$E\left(\hat{\mathbf{u}}_p' \mathbf{P}_0 \hat{\mathbf{u}}_p\right) = (M - 2d_1 + d_3)\sigma_v^2 + (N - d_2)\sigma_e^2$$

where

$$d_1 = \text{tr} \left\{ \left(\mathbf{X}'_\alpha \mathbf{X}_\alpha \right)^{-1} \mathbf{X}'_\alpha \mathbf{Z}_0 \mathbf{Z}'_0 \mathbf{X}_\alpha \right\}$$

$$d_2 = \text{tr} \left\{ \left(\mathbf{X}'_\alpha \mathbf{X}_\alpha \right)^{-1} \mathbf{X}'_\alpha \mathbf{P}_0 \mathbf{X}_\alpha \right\}$$

$$d_3 = \text{tr} \left\{ \left(\mathbf{X}'_\alpha \mathbf{X}_\alpha \right)^{-1} \mathbf{X}'_\alpha \mathbf{P}_0 \mathbf{X}_\alpha \left(\mathbf{X}'_\alpha \mathbf{X}_\alpha \right)^{-1} \mathbf{X}'_\alpha \mathbf{Z}_0 \mathbf{Z}'_0 \mathbf{X}_\alpha \right\}$$

Wansbeek-Kapteyn Method

You can use the Wansbeek-Kapteyn method of estimating variance components by specifying the VCOMP=WK option in the MODEL statement. The method is a specialization (Baltagi and Chang 1994) of the approach used by Wansbeek and Kapteyn (1989) for unbalanced two-way models. The method was also suggested by Amemiya (1971) for balanced data.

The Wansbeek and Kapteyn method is an ANOVA method that uses the within residuals from one-way fixed effects, $\hat{\mathbf{u}}_w$, in both quadratic forms.

The expected values of the quadratic forms are

$$E\left(\hat{\mathbf{u}}_w' \mathbf{Q}_0 \hat{\mathbf{u}}_w\right) = (M - N - K)\sigma_e^2$$

$$E\left(\hat{\mathbf{u}}_w' \mathbf{P}_0 \hat{\mathbf{u}}_w\right) = (N - 1 + d)\sigma_e^2 + \left(M - M^{-1} \sum_{i=1}^N T_i^2 \right) \sigma_v^2$$

where

$$d = \text{tr} \left\{ (\mathbf{X}' \mathbf{Q}_0 \mathbf{X})^{-1} \mathbf{X}' \mathbf{P}_0 \mathbf{X} \right\} - \text{tr} \left\{ (\mathbf{X}' \mathbf{Q}_0 \mathbf{X})^{-1} \mathbf{X}' \bar{\mathbf{J}}_M \mathbf{X} \right\}$$

Fuller-Battese Method

You can use the Fuller and Battese (1974) method of estimating variance components by specifying the VCOMP=FB option in the MODEL statement. Following Baltagi (2013, sec. 9.2), obtain $\hat{\sigma}_e^2$ as the mean square error (MSE) from one-way fixed effects. The cross-sectional variance is

$$\hat{\sigma}_v^2 = \frac{R(\mathbf{v}|\boldsymbol{\beta}) - (N-1)\hat{\sigma}_e^2}{M - \text{tr}\{\mathbf{Z}'_0\mathbf{X}_\alpha(\mathbf{X}'_\alpha\mathbf{X}_\alpha)^{-1}\mathbf{X}'_\alpha\mathbf{Z}_0\}}$$

where

$$R(\mathbf{v}|\boldsymbol{\beta}) = R(\boldsymbol{\beta}|\mathbf{v}) + R(\mathbf{v}) - R(\boldsymbol{\beta})$$

for

$$\begin{aligned} R(\mathbf{v}) &= \mathbf{y}'\mathbf{Z}_0(\mathbf{Z}'_0\mathbf{Z}_0)^{-1}\mathbf{Z}'_0\mathbf{y} \\ R(\boldsymbol{\beta}|\mathbf{v}) &= \mathbf{y}'_w\mathbf{X}'_w(\mathbf{X}'_w\mathbf{X}_w)^{-1}\mathbf{X}'_w\mathbf{y}_w \\ R(\boldsymbol{\beta}) &= \mathbf{y}'\mathbf{X}'_\alpha(\mathbf{X}'_\alpha\mathbf{X}_\alpha)^{-1}\mathbf{X}'_\alpha\mathbf{y} \end{aligned}$$

Nerlove Method

You can use the Nerlove (1971) method of estimating variance components by specifying the VCOMP=NL option in the MODEL statement. Nerlove's method provides a simple alternative to previous estimation strategies. You estimate σ_v^2 as the sample variance of the cross-sectional effects, estimated from a one-way fixed-effects regression. Specifically, $\hat{\sigma}_v^2 = (N-1)^{-1} \sum_{i=1}^N (\hat{v}_i - \bar{v})^2$, where $\bar{v}$ is the mean of the estimated fixed effects. You estimate σ_e^2 by taking the error sum of squares from one-way fixed-effects regression and then dividing by M .

Selecting the Appropriate Variance Component Method

Baltagi and Chang (1994) conducted an extensive simulation study regarding the finite-sample properties of the variance estimators that are supported by the CPANEL procedure. The choice of method has little bearing on estimates of regression coefficients, their standard errors, and estimation of the error variance σ_e^2 . If your goal is inference on $\boldsymbol{\beta}$, then the variance-component method will matter little.

The methods have varying performance in how they estimate σ_v^2 , the cross-sectional variance. All five methods tend to perform poorly if either the data are severely unbalanced or the ratio σ_v^2/σ_e^2 is much greater than 1.

The Nerlove method is the only method that guarantees a nonnegative estimate of σ_v^2 ; the other four methods reset a negative estimate to 0. However, the Nerlove method is particularly unsuitable for unbalanced data because the sample variance that it computes is not weighted by T_i .

Two-Way Random-Effects Model (RANTWO Option)

You perform two-way random-effects estimation by specifying the RANTWO option in the MODEL statement. The specification for the two-way random-effects model is

$$u_{it} = v_i + \lambda_t + e_{it}$$

where the v_i are iid with zero mean and variance σ_v^2 , the λ_t are iid with zero mean and variance σ_λ^2 , and the e_{it} are iid with zero mean and variance σ_e^2 . Furthermore, a random-effects specification assumes that the error terms are mutually uncorrelated and that each error term is uncorrelated with $\mathbf{X}$.

Estimation proceeds in two steps. First, you obtain estimates of the variance components σ_v^2 , σ_λ^2 , and σ_e^2 . The CPANEL procedure provides five methods for estimating variance components; these methods are described in the following subsections.

Second, with the variance-component estimates in hand, you transform the data in such a way that estimation can take place using ordinary least squares (OLS). In two-way models with unbalanced data, the transformation is quite complex. Throughout this section, $\mathbf{y}$ and $\mathbf{X}$ are treated as being sorted first by time, and then by cross-section within time. For the definitions of the design matrices $\mathbf{D}_N$ and $\mathbf{D}_T$, see the section “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 496. The variance of $\mathbf{y}$ is

$$\mathbf{\Omega} = \sigma_e^2 \mathbf{I}_M + \sigma_v^2 \mathbf{D}_N \mathbf{D}_N' + \sigma_\lambda^2 \mathbf{D}_T \mathbf{D}_T'$$

and estimation proceeds as OLS regression of $\hat{\sigma}_e \hat{\mathbf{\Omega}}^{-1/2} \mathbf{y}$ on $\hat{\sigma}_e \hat{\mathbf{\Omega}}^{-1/2} \mathbf{X}_\alpha$.

Rather than invert the $M \times M$ matrix $\hat{\mathbf{\Omega}}$ directly, Wansbeek and Kapteyn (1989) provide the more convenient form

$$\hat{\sigma}_e^2 \hat{\mathbf{\Omega}}^{-1} = \mathbf{V} - \mathbf{V} \mathbf{D}_T \tilde{\mathbf{P}}^{-1} \mathbf{D}_T' \mathbf{V}$$

where

$$\begin{aligned} \mathbf{V} &= \mathbf{I}_M - \mathbf{D}_N \tilde{\Delta}_N^{-1} \mathbf{D}_N' \\ \tilde{\mathbf{P}} &= \tilde{\Delta}_T - \mathbf{D}_T' \mathbf{D}_N \tilde{\Delta}_N^{-1} \mathbf{D}_N' \mathbf{D}_T \end{aligned}$$

with $\tilde{\Delta}_N = \mathbf{D}_N' \mathbf{D}_N + (\hat{\sigma}_e^2 / \hat{\sigma}_v^2) \mathbf{I}_N$ and $\tilde{\Delta}_T = \mathbf{D}_T' \mathbf{D}_T + (\hat{\sigma}_e^2 / \hat{\sigma}_\lambda^2) \mathbf{I}_T$.

If the data are balanced, then the calculations are simplified considerably—the data are transformed from z_{it} to $z_{it} - \hat{\theta}_1 \bar{z}_{i.} - \hat{\theta}_2 \bar{z}_{.t} + \hat{\theta}_3 \bar{z}_{..}$, where

$$\begin{aligned} \hat{\theta}_1 &= 1 - \hat{\sigma}_e (T \hat{\sigma}_v^2 + \hat{\sigma}_e^2)^{-1/2} \\ \hat{\theta}_2 &= 1 - \hat{\sigma}_e (N \hat{\sigma}_\lambda^2 + \hat{\sigma}_e^2)^{-1/2} \\ \hat{\theta}_3 &= \hat{\theta}_1 + \hat{\theta}_2 + \hat{\sigma}_e (T \hat{\sigma}_v^2 + N \hat{\sigma}_\lambda^2 + \hat{\sigma}_e^2)^{-1/2} - 1 \end{aligned}$$

The CPANEL procedure provides five options for estimating variance components, as described in the following subsections.

Swamy-Arora Method

The Swamy and Arora (1972) method is the default method of estimating variance components; it can also be specified as the VCOMP=SA option in the MODEL statement. The Swamy-Arora method is part of a class of methods known as analysis of variance (ANOVA) estimators.

ANOVA estimators obtain variance components by solving a system of equations that is based on expected sums of squares. The following quadratic forms correspond to the two-way within sum of squares, the sum of squares between time periods, and the sum of squares between cross sections, respectively:

$$\begin{aligned} q_e &= \mathbf{u}' \mathbf{P} \mathbf{u} \\ q_\lambda &= \mathbf{u}' \mathbf{D}_T \Delta_T^{-1} \mathbf{D}_T' \mathbf{u} \\ q_v &= \mathbf{u}' \mathbf{D}_N \Delta_N^{-1} \mathbf{D}_N' \mathbf{u} \end{aligned}$$

The matrix $\mathbf{P}$ is the two-way within transformation defined in the section “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 496, $\Delta_T = \mathbf{D}_T' \mathbf{D}_T$, $\Delta_N = \mathbf{D}_N' \mathbf{D}_N$, and $\mathbf{u}$ is the vector of true residuals.

The ANOVA methods differ only in how they estimate $\mathbf{u}$. The Swamy-Arora method substitutes into q_e the within residuals from two-way fixed-effects estimation, substitutes into q_λ the residuals from between-time-periods regression, and substitutes into q_v the residuals from between-cross-sections regression.

The expected values of the quadratic forms are

$$\begin{aligned} E(\hat{\mathbf{u}}_f' \mathbf{P} \hat{\mathbf{u}}_f) &= (M - N - T - K + 1)\sigma_e^2 \\ E(\hat{\mathbf{u}}_\lambda' \mathbf{P}_\lambda \hat{\mathbf{u}}_\lambda) &= \left[M - \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{P}_\lambda \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{D}_T \mathbf{D}_T' \mathbf{X}_\alpha \right\} \right] \sigma_\lambda^2 + (T - K - 1)\sigma_e^2 \\ E(\hat{\mathbf{u}}_v' \mathbf{P}_v \hat{\mathbf{u}}_v) &= \left[M - \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{P}_v \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{D}_N \mathbf{D}_N' \mathbf{X}_\alpha \right\} \right] \sigma_v^2 + (N - K - 1)\sigma_e^2 \end{aligned}$$

where $\mathbf{P}_\lambda = \mathbf{D}_T \Delta_T^{-1} \mathbf{D}_T'$ and $\mathbf{P}_v = \mathbf{D}_N \Delta_N^{-1} \mathbf{D}_N'$.

Estimates of σ_v^2 , σ_λ^2 , and σ_e^2 are obtained by setting the expected sums of squares to their observed values and solving.

Wallace-Hussain Method

You can use the Wallace and Hussain (1969) method of estimating variance components by specifying the VCOMP=WH option in the MODEL statement. The Wallace-Hussain method is an ANOVA method that uses the residuals from pooled (OLS) regression, $\hat{\mathbf{u}}_p$, in all three quadratic forms.

The expected values of the quadratic forms are

$$\begin{aligned} E(\hat{\mathbf{u}}_p' \mathbf{P} \hat{\mathbf{u}}_p) &= d_{11}\sigma_e^2 + d_{12}\sigma_v^2 + d_{13}\sigma_\lambda^2 \\ E(\hat{\mathbf{u}}_p' \mathbf{P}_\lambda \hat{\mathbf{u}}_p) &= d_{21}\sigma_e^2 + d_{22}\sigma_v^2 + d_{23}\sigma_\lambda^2 \\ E(\hat{\mathbf{u}}_p' \mathbf{P}_v \hat{\mathbf{u}}_p) &= d_{31}\sigma_e^2 + d_{32}\sigma_v^2 + d_{33}\sigma_\lambda^2 \end{aligned}$$

Define $\Sigma = (\mathbf{X}'_\alpha \mathbf{X}_\alpha)^{-1}$, which is the inverse crossproducts matrix from pooled regression. Also define $\mathbf{S}_v = \mathbf{X}'_\alpha \mathbf{D}_N \mathbf{D}_N' \mathbf{X}_\alpha$ and $\mathbf{S}_\lambda = \mathbf{X}'_\alpha \mathbf{D}_T \mathbf{D}_T' \mathbf{X}_\alpha$, which are the individual-level sum of squares and the time-

level sum of squares, respectively. The coefficients are

$$\begin{aligned}
 d_{11} &= M - N - T + 1 - \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P} \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{12} &= \text{tr}(\mathbf{S}_v \boldsymbol{\Sigma} \mathbf{X}'_{\alpha} \mathbf{P} \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{13} &= \text{tr}(\mathbf{S}_{\lambda} \boldsymbol{\Sigma} \mathbf{X}'_{\alpha} \mathbf{P} \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{21} &= T - \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_{\lambda} \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{22} &= T - 2\text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_{\lambda} \mathbf{D}_N \mathbf{D}'_N \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) + \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_{\lambda} \mathbf{X}_{\alpha} \boldsymbol{\Sigma} \mathbf{S}_v \boldsymbol{\Sigma}) \\
 d_{23} &= M - 2\text{tr}(\mathbf{S}_{\lambda} \boldsymbol{\Sigma}) + \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_{\lambda} \mathbf{X}_{\alpha} \boldsymbol{\Sigma} \mathbf{S}_{\lambda} \boldsymbol{\Sigma}) \\
 d_{31} &= N - \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_v \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{32} &= M - 2\text{tr}(\mathbf{S}_v \boldsymbol{\Sigma}) + \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_v \mathbf{X}_{\alpha} \boldsymbol{\Sigma} \mathbf{S}_v \boldsymbol{\Sigma}) \\
 d_{33} &= N - 2\text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_v \mathbf{D}_T \mathbf{D}'_T \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) + \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_v \mathbf{X}_{\alpha} \boldsymbol{\Sigma} \mathbf{S}_{\lambda} \boldsymbol{\Sigma})
 \end{aligned}$$

Wansbeek-Kapteyn Method

You can use the Wansbeek-Kapteyn method of estimating variance components by specifying the VCOMP=WK option in the MODEL statement. The method is a specialization (Baltagi and Chang 1994) of the approach used by Wansbeek and Kapteyn (1989) for unbalanced two-way models.

The Wansbeek and Kapteyn method is an ANOVA method that uses the within residuals from two-way fixed effects, $\hat{\mathbf{u}}_f$, in all three quadratic forms.

The expected values of the quadratic forms are

$$\begin{aligned}
 E(\hat{\mathbf{u}}'_f \mathbf{P} \hat{\mathbf{u}}_f) &= (M - N - T - K + 1) \sigma_e^2 \\
 E(\hat{\mathbf{u}}'_f \mathbf{P}_{\lambda} \hat{\mathbf{u}}_f) &= (T + k_N - k_0) \sigma_e^2 + (T - \delta_N) \sigma_v^2 + (M - \delta_T) \sigma_{\lambda}^2 \\
 E(\hat{\mathbf{u}}'_f \mathbf{P}_v \hat{\mathbf{u}}_f) &= (N + k_T - k_0) \sigma_e^2 + (M - \delta_N) \sigma_v^2 + (N - \delta_T) \sigma_{\lambda}^2
 \end{aligned}$$

where $\delta_N = M^{-1} \sum_{i=1}^N T_i^2$ and $\delta_T = M^{-1} \sum_{t=1}^T N_t^2$. The other constants are defined by

$$\begin{aligned}
 k_0 &= 1 + M^{-1} \mathbf{j}'_M \mathbf{X} (\mathbf{X}' \mathbf{P} \mathbf{X})^{-1} \mathbf{X}' \mathbf{j}_M \\
 k_N &= \text{tr}\{(\mathbf{X}' \mathbf{P} \mathbf{X})^{-1} \mathbf{X}' \mathbf{P}_{\lambda} \mathbf{X}\} \\
 k_T &= \text{tr}\{(\mathbf{X}' \mathbf{P} \mathbf{X})^{-1} \mathbf{X}' \mathbf{P}_v \mathbf{X}\}
 \end{aligned}$$

When the NOINT option is specified, the variance-component equations change slightly: k_0 , δ_N , and δ_T are all replaced by 0.

Fuller-Battese Method

You can use the Fuller and Battese (1974) method of estimating variance components by specifying the VCOMP=FB option in the MODEL statement. Following the discussion in Baltagi, Song, and Jung (2002), the Fuller-Battese method is a variation of the three ANOVA methods discussed previously.

The quadratic form, q_e , is the same as in the previous methods, and $\mathbf{u}$ is estimated by the two-way within residuals $\hat{\mathbf{u}}_f$. The other two quadratic forms, q_λ and q_v , are replaced by the error sums of squares from one-way fixed-effects estimations.

The resulting system of equations is

$$\begin{aligned} E(\hat{\mathbf{u}}_f' \mathbf{P} \hat{\mathbf{u}}_f) &= (M - N - T - K + 1)\sigma_e^2 \\ E(\tilde{\mathbf{u}}_\lambda' \tilde{\mathbf{u}}_\lambda) &= (M - T - K)\sigma_e^2 + \left[M - T - \text{tr} \left\{ \mathbf{X}' \mathbf{W}_\lambda \mathbf{D}_N \mathbf{D}_N' \mathbf{W}_\lambda \mathbf{X} (\mathbf{X}' \mathbf{W}_\lambda \mathbf{X})^{-1} \right\} \right] \sigma_v^2 \\ E(\tilde{\mathbf{u}}_v' \tilde{\mathbf{u}}_v) &= (M - N - K)\sigma_e^2 + \left[M - N - \text{tr} \left\{ \mathbf{X}' \mathbf{W}_v \mathbf{D}_T \mathbf{D}_T' \mathbf{W}_v \mathbf{X} (\mathbf{X}' \mathbf{W}_v \mathbf{X})^{-1} \right\} \right] \sigma_\lambda^2 \end{aligned}$$

where $\mathbf{W}_\lambda = \mathbf{I}_M - \mathbf{P}_\lambda$, $\mathbf{W}_v = \mathbf{I}_M - \mathbf{P}_v$, $\tilde{\mathbf{u}}_\lambda$ are the residuals from a one-way model with time fixed effects, and $\tilde{\mathbf{u}}_v$ are the residuals from a one-way model with individual fixed effects.

Nerlove Method

You can use the Nerlove (1971) method of estimating variance components by specifying the VCOMP=NL option in the MODEL statement.

Begin by fitting a two-way fixed-effects model. The estimator of the error variance is

$$\hat{\sigma}_e^2 = M^{-1} \hat{\mathbf{u}}_f' \mathbf{P} \hat{\mathbf{u}}_f$$

You obtain $\hat{\sigma}_v^2$ as the sample variance of the N estimated individual effects, and $\hat{\sigma}_\lambda^2$ as the sample variance of the T estimated time effects.

Hausman-Taylor Estimation (HTAYLOR Option)

You perform Hausman-Taylor estimation by specifying the HTAYLOR option in the MODEL statement. The Hausman and Taylor (1981) model is a hybrid that combines the consistency of a fixed-effects model with the efficiency and applicability of a random-effects model. One-way random-effects models assume exogeneity of the regressors, namely that they are independent of both the cross-sectional and observation-level errors. When some regressors are correlated with the cross-sectional errors, the random effects model can be adjusted to deal with this form of endogeneity.

Consider the one-way model:

$$y_{it} = \mathbf{x}_{1it} \boldsymbol{\beta}_1 + \mathbf{x}_{2it} \boldsymbol{\beta}_2 + \mathbf{z}_{1i} \boldsymbol{\gamma}_1 + \mathbf{z}_{2i} \boldsymbol{\gamma}_2 + v_i + e_{it}$$

The regressors are subdivided so that $\mathbf{x}_{1it}$ and $\mathbf{x}_{2it}$ vary within cross sections whereas $\mathbf{z}_{1i}$ and $\mathbf{z}_{2i}$ do not and would otherwise be dropped from a fixed-effects model. The subscript 1 denotes variables that are independent of both error terms (exogenous variables), and the subscript 2 denotes variables that are

independent of the observation-level errors e_{it} but correlated with cross-sectional errors v_i . The intercept term (if your model has one) is included as part of $\mathbf{z}_{1i}$.

The Hausman-Taylor estimator is a two-stage least squares (2SLS) regression on data that are weighted similarly to data for random-effects estimation. The weights are functions of the estimated variance components.

The observation-level variance is estimated from a one-way fixed-effects model fit. Obtain $\mathbf{y}_w$, $\mathbf{X}_w$, and $\hat{\boldsymbol{\beta}}_w$ from the section “[One-Way Fixed-Effects Model \(FIXONE and FIXONETIME Options\)](#)” on page 494. Then $\hat{\sigma}_e^2 = \text{SSE}/(M - N)$, where

$$\text{SSE} = (\mathbf{y}_w - \mathbf{X}_w \hat{\boldsymbol{\beta}}_w)' (\mathbf{y}_w - \mathbf{X}_w \hat{\boldsymbol{\beta}}_w)$$

To estimate the cross-sectional error variance, form the mean-residual vector $\mathbf{r} = \mathbf{P}'_0(\mathbf{y} - \mathbf{X}_w \hat{\boldsymbol{\beta}}_w)$, where $\mathbf{P}_0 = \text{diag}(\bar{\mathbf{J}}_{T_i})$. You can use the mean residuals to obtain intermediate estimates of the coefficients for $\mathbf{z}_1$ and $\mathbf{z}_2$ via two-stage least squares (2SLS) estimation. At the first stage, use $\mathbf{x}_1$ and $\mathbf{z}_1$ as instrumental variables to predict $\mathbf{z}_2$. At the second stage, regress $\mathbf{r}$ on both $\mathbf{z}_1$ and the predicted $\mathbf{z}_2$ to obtain $\hat{\boldsymbol{\gamma}}_1^m$ and $\hat{\boldsymbol{\gamma}}_2^m$.

To estimate the cross-sectional variance, compute $\hat{\sigma}_v^2 = \{R(v)/N - \hat{\sigma}_e^2\}/\bar{T}$, where $\bar{T} = N/(\sum_{i=1}^N T_i^{-1})$ and

$$R(v) = (\mathbf{r} - \mathbf{Z}_1 \hat{\boldsymbol{\gamma}}_1^m - \mathbf{Z}_2 \hat{\boldsymbol{\gamma}}_2^m)' (\mathbf{r} - \mathbf{Z}_1 \hat{\boldsymbol{\gamma}}_1^m - \mathbf{Z}_2 \hat{\boldsymbol{\gamma}}_2^m)$$

The design matrices $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are formed by stacking the data observations of $\mathbf{z}_{1i}$ and $\mathbf{z}_{2i}$, respectively.

After variance-component estimation, transform the dependent variable into partial deviations: $y_{it}^* = y_{it} - \hat{\theta}_i \bar{y}_{i.}$. Likewise, transform the regressors to form $\mathbf{x}_{1it}^*$, $\mathbf{x}_{2it}^*$, $\mathbf{z}_{1i}^*$, and $\mathbf{z}_{2i}^*$. The partial weights $\hat{\theta}_i$ are determined by $\hat{\theta}_i = 1 - \hat{\sigma}_e/\hat{w}_i$, with $\hat{w}_i^2 = T_i \hat{\sigma}_v^2 + \hat{\sigma}_e^2$.

Finally, you obtain the Hausman-Taylor estimates by performing 2SLS regression of y_{it}^* on $\mathbf{x}_{1it}^*$, $\mathbf{x}_{2it}^*$, $\mathbf{z}_{1i}^*$, and $\mathbf{z}_{2i}^*$. For the first-stage regression, use the following instruments:

- $\bar{x}_{it}$, the deviations from cross-sectional means for all time-varying variables (correlated and uncorrelated) for the i th cross section during time period t
- $(1 - \hat{\theta}_i) \bar{x}_{1i.}$, where $\bar{x}_{1i.}$ are the means of the time-varying exogenous variables for the i th cross section
- $(1 - \hat{\theta}_i) \mathbf{z}_{1i}$

Multiplication by the factor $(1 - \hat{\theta}_i)$ is redundant in balanced data, but necessary in the unbalanced case to produce accurate instrumentation; see Gardner (1998).

Let k_1 equal the number of regressors in $\mathbf{x}_1$, and let g_2 equal the number of regressors in $\mathbf{z}_2$. Then the Hausman-Taylor model is identified only if $k_1 \geq g_2$; otherwise, no estimation takes place.

Hausman and Taylor (1981) describe a specification test that compares their model to fixed effects. For a null hypothesis of fixed effects, Hausman's m statistic is calculated by comparing the parameter estimates and variance matrices for both models identically to how it is calculated for one-way random effects models; for more information, see the section “[Hausman Specification Tests](#)” on page 521. However, the degrees of freedom of the test are not based on matrix rank but instead are equal to $k_1 - g_2$.

Amemiya-MaCurdy Estimation (AMACURDY Option)

You perform Amemiya-MaCurdy estimation by specifying the AMACURDY option in the MODEL statement. The Amemiya and MaCurdy (1986) model is similar to the Hausman-Taylor model. Following the development in the section “Hausman-Taylor Estimation (HTAYLOR Option)” on page 503, estimation is identical up to the final 2SLS instrumental variables regression. In addition to the set of instruments used by the Hausman-Taylor estimator, use the following:

- $\mathbf{x}_{1i1}, \mathbf{x}_{1i2}, \dots, \mathbf{x}_{1iT}$

For each observation in the i th cross section, you use the data on the time-varying exogenous regressors for the entire cross section. Because of the structure of the added instruments, the Amemiya-MaCurdy estimator can be applied only to balanced data.

The Amemiya-MaCurdy model attempts to gain efficiency over Hausman-Taylor by adding instruments. This comes at a price of a more stringent assumption on the exogeneity of the $\mathbf{x}_1$ variables. Although the Hausman-Taylor model requires only that the cross-sectional means of $\mathbf{x}_1$ be orthogonal to v_i , the Amemiya-MaCurdy estimation requires orthogonality at every point in time; see Baltagi (2013, sec. 7.4).

A Hausman specification test is provided to test the validity of the added assumption. Define $\alpha' = (\beta_1', \beta_2', \gamma_1', \gamma_2')$, its Hausman-Taylor estimate as $\hat{\alpha}_{HT}$, and its Amemiya-MaCurdy estimate as $\hat{\alpha}_{AM}$. Under the null hypothesis, both estimators are consistent and $\hat{\alpha}_{AM}$ is efficient. The Hausman test statistic is

$$m = (\hat{\alpha}_{HT} - \hat{\alpha}_{AM})' (\hat{\Sigma}_{HT} - \hat{\Sigma}_{AM})^{-1} (\hat{\alpha}_{HT} - \hat{\alpha}_{AM})$$

where $\hat{\Sigma}_{HT}$ and $\hat{\Sigma}_{AM}$ are variance-covariance estimates of $\hat{\alpha}_{HT}$ and $\hat{\alpha}_{AM}$, respectively. Under the null hypothesis, m follows a χ^2 distributed with degrees of freedom equal to the rank of $(\hat{\Sigma}_{HT} - \hat{\Sigma}_{AM})^{-1}$.

Instrumental Variables Regression

Recall the original specification of the regression model:

$$y_{it} = \alpha + \sum_{k=1}^K x_{itk} \beta_k + u_{it} \quad i = 1, \dots, N; t = 1, \dots, T_i$$

Consider that a subset of the regressors are *endogenous*—that is, these regressors are correlated with the observation-level errors u_{it} .

Formally, divide $\mathbf{x}$ and $\boldsymbol{\beta}$ so that

$$y_{it} = \mathbf{x}_{1it} \boldsymbol{\beta}_1 + \mathbf{x}_{2it} \boldsymbol{\beta}_2 + u_{it}$$

where $\mathbf{x}_{1it}$ are k_1 uncorrelated (exogenous) regressors, $\mathbf{x}_{2it}$ are k_2 endogenous regressors, and the intercept (if your model has one) is contained in $\mathbf{x}_{1it}$.

In matrix notation, the model is

$$\mathbf{y} = \mathbf{X}_1 \boldsymbol{\beta}_1 + \mathbf{X}_2 \boldsymbol{\beta}_2 + \mathbf{u}$$

where the matrix rows are arranged by cross section, and by time within cross section.

Instrumental variables (IV) regression is the process of using external regressors, called *instruments*, to deal with the endogeneity. Let $\mathbf{Z}$ represent a set of p instruments such that $p \geq k_2$. The set of endogenous regressors, $\mathbf{X}_2$, are specified as effects in the ENDOGENOUS statement. The set of external instruments, $\mathbf{Z}$, are specified as effects in the INSTRUMENTS statement. Define $\mathbf{X} = [\mathbf{X}_1 \mathbf{X}_2]$.

The CPANEL procedure performs instrumental variables regression by the method of two-stage least squares (2SLS) or the efficient GMM. Two-stage least squares is a special case of the GMM and is equivalent to the one-step GMM method. The efficient GMM method, which is also called two-step GMM, is a GMM that uses the optimal weighting matrix. The optimal weighting matrix is based on the residuals from the one-step GMM. If you define the estimated residual vector as $\hat{\mathbf{u}}$ and the degrees of freedom as df , then you can obtain the estimated variance of the error term as $\sigma_u^2 = \mathbf{u}'\mathbf{u}/df$ and use it to form the optimal weighting matrix.

The 2SLS estimator can be obtained as

$$\hat{\boldsymbol{\beta}}_{2SLS} = [\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}]^{-1}\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{y}$$

The estimated variance-covariance matrix of the 2SLS estimator is

$$\text{Var}(\hat{\boldsymbol{\beta}}_{2SLS}) = \hat{\sigma}^2[\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}]^{-1}$$

You can also obtain the heteroscedasticity robust standard error for 2SLS by specifying the ROBUST option. Define $\hat{\Omega}$ as the diagonal matrix of squared residuals from the 2SLS estimation:

$$\hat{\Omega} = \begin{pmatrix} \hat{u}_1^2 & & 0 \\ & \ddots & \\ 0 & & \hat{u}_{NT}^2 \end{pmatrix}$$

Define $\mathbf{P}_Z = \mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'$ and $\mathbf{W}_2 = \mathbf{Z}'\hat{\Omega}\mathbf{Z}$. Then the estimated robust variance-covariance matrix of the 2SLS estimator is

$$\text{Var}_r(\hat{\boldsymbol{\beta}}_{2SLS}) = [\mathbf{X}'\mathbf{P}_Z\mathbf{X}]^{-1}[\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{W}_2(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}][\mathbf{X}'\mathbf{P}_Z\mathbf{X}]^{-1}$$

The two-step GMM estimator can be obtained as

$$\hat{\boldsymbol{\beta}}_{GMM} = [\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\hat{\Omega}\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}]^{-1}\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\hat{\Omega}\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{y}$$

with asymptotic variance

$$\text{Var}(\hat{\boldsymbol{\beta}}_{GMM}) = [\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\hat{\Omega}\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}]^{-1}$$

You can also obtain the heteroscedasticity robust standard error for two-step GMM. Define $\hat{\Omega}_2$ as the diagonal matrix of squared residuals from the two-step GMM estimation and $\mathbf{W}_3 = \mathbf{Z}'\hat{\Omega}_2\mathbf{Z}$. Then the estimated robust variance-covariance matrix of the two-step GMM estimator is

$$\text{Var}_r(\hat{\boldsymbol{\beta}}_{GMM}) = [\mathbf{X}'\mathbf{Z}(\mathbf{W}_2)^{-1}\mathbf{Z}'\mathbf{X}]^{-1}[\mathbf{X}'\mathbf{Z}(\mathbf{W}_2)^{-1}\mathbf{W}_3(\mathbf{W}_2)^{-1}\mathbf{Z}'\mathbf{X}][\mathbf{X}'\mathbf{Z}(\mathbf{W}_2)^{-1}\mathbf{Z}'\mathbf{X}]^{-1}$$

Similar to the two-step GMM estimation for a dynamic panel in the section “[Estimation](#)” on page 514, you can obtain a bias-corrected variance of $\hat{\boldsymbol{\beta}}_{GMM}$ for IV models by using the BIASCORRECTED option. The

formulas are as the same as those from the section “[Estimation](#)” on page 514. Define the one-step and two-step residuals as $\hat{u}_{1i} = y_i - \mathbf{X}_i \hat{\beta}_{2SLS}$ and $\hat{u}_{2i} = y_i - \mathbf{X}_i \hat{\beta}_{GMM}$, respectively. Also define

$$\mathbf{P}_e = \sum_{i=1}^N \mathbf{Z}_i' \hat{u}_{2i}; \quad \mathbf{P}_x = \sum_{i=1}^N \mathbf{Z}_i' \mathbf{X}_i^s$$

Formulate the matrix $\mathbf{D}$ such that its k th column is $\mathbf{D}_k = \mathbf{V}_2 \mathbf{P}_x' (W_2)^{-1} \mathbf{F}_k (W_2)^{-1} \mathbf{P}_e$, where $\mathbf{V}_2 = \text{Var}(\hat{\beta}_{GMM})$. The matrix $\mathbf{F}_k$ is the quadratic form

$$\mathbf{F}_k = \sum_{i=1}^{NT} \mathbf{Z}_i' \left(\mathbf{x}_{ik} \hat{u}_{1i} + \hat{u}_{1i} \mathbf{x}_{ik}' \right) \mathbf{Z}_i$$

where $\mathbf{x}_{ik}$ is the k th column of $\mathbf{X}_i$. Note that the $\mathbf{F}_k$ here is different from the $\mathbf{F}_k$ in the dynamic panel section. By using this $\mathbf{F}_k$, the bias-corrected standard errors only correct for the bias from the two-step GMM, but not robust to serial correlation.

The Windmeijer (2005) bias-corrected variance is

$$\text{Var}^w(\hat{\beta}_{GMM}) = \mathbf{V}_2 + \mathbf{D} \mathbf{V}_2 + \mathbf{V}_2 \mathbf{D}' + \mathbf{D} \mathbf{V}_1^r \mathbf{D}'$$

where $\mathbf{V}_1^r$ is the robust variance estimate of $\hat{\beta}_{2SLS}$.

Depending on the error structure of u_{it} , there are eight IV models available, as described in the following subsections. In each model different $\mathbf{X}$, $\mathbf{Z}$, and $\mathbf{y}$ are substituted in the preceding formulas to obtain the 2SLS and the GMM estimators.

Instrumental Variables Pooled Regression (IVPOOLED Option)

You perform IV pooled regression by specifying the IVPOOLED option in the MODEL statement. Pooled model uses the standard regression model without any cross-sectional or time effects. The error structure is simply $u_{it} = e_{it}$, where the e_{it} are iid with zero mean and variance σ_e^2 . The errors e_{it} are uncorrelated with $\mathbf{x}_{1it}$, but correlated with $\mathbf{x}_{2it}$. The degree of freedom is $M - K - 1$.

Instrumental Variables Between-Groups Regression (IVBTWNG Option)

You perform IV between-groups regression by specifying the IVBTWNG option in the MODEL statement. When the data are balanced, between-groups IV model estimation is two-stage least squares (2SLS) regression or two-step GMM estimation performed on data that have been collapsed into cross-sectional means. When the data are unbalanced, the regression is weighted to assign more importance to larger cross sections. The cross-sectional means are multiplied by the weighting factor $w_i = \sqrt{T_i / \bar{T}}$, where $\bar{T} = M/N$ is the average cross section size.

Instrumental Variables One-Way Fixed Effects (IVFIXONE Option)

You perform IV one-way fixed-effects estimation by specifying the IVFIXONE option in the MODEL statement. The error structure for the one-way fixed-effects model is

$$u_{it} = v_i + e_{it}$$

where the v_i are nonrandom parameters that are restricted to sum to 0, and the e_{it} are iid with zero mean and variance σ_e^2 . The errors e_{it} are uncorrelated with $\mathbf{x}_{1it}$, but correlated with $\mathbf{x}_{2it}$.

The IV fixed-effects model can be estimated as an IV pooled regression, treating the v_i as coefficients on dummy variables that identify the cross sections. However, when N is large, you might want to estimate only β and not v_i .

Let $\mathbf{Q}_0 = \text{diag}(\mathbf{E}_{T_i})$. The matrix $\mathbf{Q}_0$ is the within transformation that removes means (and v_i) from the estimation. Define $\mathbf{y}_w = \mathbf{Q}_0\mathbf{y}$, and use the same transformation to form $\mathbf{X}_{1w}$, $\mathbf{X}_{2w}$ and $\mathbf{Z}_w$. Because the within transformation also removes the intercept term, the intercept is not included as part of $\mathbf{X}_1$.

The within-group estimates $\hat{\beta}_{\text{WIV}}$, $\text{Var}(\hat{\beta}_{\text{WIV}})$ can be obtained by substituting $\mathbf{y}_w$, $\mathbf{X}_w$, and $\mathbf{Z}_w$ into the general estimation equation in section “[Instrumental Variables Regression](#)” on page 505. You can obtain these estimates by using 2SLS method or two-step GMM method. When forming the error variance $\hat{\sigma}_e^2$, the error degrees of freedom are adjusted to equal $M - N - K$ to account for the N fixed effects.

The estimate of the intercept of the IVFIXONE model is

$$\hat{\alpha} = \bar{y}_{..} - \bar{x}_{..}\hat{\beta}_{\text{WIV}}$$

where $\bar{y}_{..}$ is the overall mean of y_{it} and $\bar{x}_{..}$ is the overall mean of x_{it} .

Fixed effects are not estimated as part of the regression but can be obtained by specifying the PRINTFIXED option in the MODEL statement. Each fixed effect in 2SLS is estimated as

$$\hat{v}_i = \bar{y}_{i.} - \bar{x}_{i.}\hat{\beta}_{\text{WIV}}$$

where $\bar{y}_{i.}$ and $\bar{x}_{i.}$ are cross-sectional means.

Variance estimates of $\hat{\alpha}$ and $\hat{v}_i$ are obtained by the delta method. If you specify the NOINT option, then the $\hat{v}_i$ are shifted to absorb $\hat{\alpha}$.

When the estimation method is 2SLS, PROC CPANEL reports a poolability test F . The F test assumes the null hypothesis that all fixed effects are jointly 0. Following the methodology of Wooldridge (2002, pp. 98–99) for restricted 2SLS, the test statistic is

$$F = \frac{(\text{SSE}_{2r} - \text{SSE}_{2u}) / (N - 1)}{\text{SSE}_u / (M - N - K)} \sim F(N - 1, M - N - K)$$

SSE_u is the final unrestricted error sum of squares

$$\text{SSE}_u = \left(\mathbf{y}_w - \mathbf{X}_{1w}\hat{\beta}_{1,\text{WIV}} - \mathbf{X}_{2w}\hat{\beta}_{2,\text{WIV}} \right)' \left(\mathbf{y}_w - \mathbf{X}_{1w}\hat{\beta}_{1,\text{WIV}} - \mathbf{X}_{2w}\hat{\beta}_{2,\text{WIV}} \right)$$

SSE_{2u} is the second-stage unrestricted error sum of squares

$$\text{SSE}_{2u} = \left(\mathbf{y}_w - \mathbf{X}_{1w}\hat{\beta}_{1,\text{WIV}} - \hat{\mathbf{X}}_{2w}\hat{\beta}_{2,\text{WIV}} \right)' \left(\mathbf{y}_w - \mathbf{X}_{1w}\hat{\beta}_{1,\text{WIV}} - \hat{\mathbf{X}}_{2w}\hat{\beta}_{2,\text{WIV}} \right)$$

and SSE_{2r} is the second-stage restricted error sum of squares, which is equal to the residual sum of squares from a pooled OLS regression of $\mathbf{y}$ on $\mathbf{X}_1$ and $\mathbf{X}_2^* = \hat{\mathbf{X}}_{2w} + \mathbf{P}_0\mathbf{X}_2$.

Instrumental Variables One-Way Time Fixed Effects (IVFIXONETIME Option)

You perform IV one-way time fixed-effects estimation by specifying the IVFIXONETIME option in the MODEL statement. The error structure for the one-way time fixed-effects model is

$$u_{it} = \lambda_t + e_{it}$$

where the λ_t are nonrandom parameters that are restricted to sum to 0, and the e_{it} are iid with zero mean and variance σ_e^2 . The errors e_{it} are uncorrelated with $\mathbf{x}_{1it}$, but correlated with $\mathbf{x}_{2it}$.

The steps of the estimation process of the one-way time fixed-effects model are the same as they are for the one-way fixed-effects model in the section “[Instrumental Variables One-Way Fixed Effects \(IVFIXONE Option\)](#)” on page 507, except IVFIXONETIME uses a different within-group transformation matrix $\mathbf{Q}_0 = \mathbf{I}_M - \bar{\mathbf{J}}_p \otimes \mathbf{I}_T$. By using $\mathbf{Q}_0$, the within-group transformation removes the time mean rather than the cross section means.

Instrumental Variables One-Way Time Fixed Effects (IVFIXTWO Option)

You perform IV two-way fixed-effects estimation by specifying the IVFIXTWO option in the MODEL statement. The error specification for the two-way fixed-effects model is

$$u_{it} = v_i + \lambda_t + e_{it}$$

where the v_i and λ_t are nonrandom parameters to be estimated. Assume the identifiability restriction $\sum_t \lambda_t = 0$.

To estimate the IV two-way fixed-effects model, first, a within transformation is needed to remove the cross-section means and time means. This step is the same as it is in section “[Two-Way Fixed-Effects Model \(FIXTWO Option\)](#)” on page 496 for both balanced and unbalanced panel. After the transformation is done, the within-group estimates $\hat{\beta}_{WIV}$, $\text{Var}(\hat{\beta}_{WIV})$ can be obtained by substituting $\mathbf{y}_w$, $\mathbf{X}_w$, and $\mathbf{Z}_w$ into the general estimation equation in section “[Instrumental Variables Regression](#)” on page 505.

Instrumental Variables One-Way Random Effects (IVRANONE Option)

You perform IV one-way random-effects estimation by specifying the IVRANONE option in the MODEL statement. The specification for the one-way random-effects model is

$$u_{it} = v_i + e_{it}$$

where the v_i are iid with zero mean and variance σ_v^2 , and the e_{it} are iid with zero mean and variance σ_e^2 . The regressors in $\mathbf{x}_{1it}$ are orthogonal to e_{it} , but the regressors in $\mathbf{x}_{2it}$ are correlated with e_{it} . This form of endogeneity is different from that assumed by Hausman and Taylor (1981), where $\mathbf{x}_{2it}$ is correlated with v_i but not with e_{it} .

The estimation performed by the CPANEL procedure is based on the error-component two-stage least squares (EC2SLS) method of Baltagi (1981). Estimation proceeds in two steps.

First, estimates of the variance components are obtained by using a variant of the Swamy-Arora method. Consider the system of equations

$$\begin{aligned} E(\hat{\mathbf{u}}_F' \mathbf{Q}_0 \hat{\mathbf{u}}_F) &= (M - N - K)\sigma_e^2 \\ E(\hat{\mathbf{u}}_B' \mathbf{P}_0 \hat{\mathbf{u}}_B) &= \left[M - \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{P}_0 \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{Z}_0 \mathbf{Z}'_0 \mathbf{X}_\alpha \right\} \right] \sigma_v^2 + (N - K - 1)\sigma_e^2 \end{aligned}$$

where $\hat{u}_F$ are the residuals from IV one-way fixed effects, $\hat{u}_B$ are the residuals from IV between-groups regression, and the other quantities are as defined in the section “[Swamy-Arora Method](#)” on page 497. Estimates of σ_v^2 and σ_e^2 are obtained by setting the expected sums of squares to their observed values and solving.

Second, with the variance components in hand, you form a weight for each cross section:

$$\hat{\theta}_i = 1 - \hat{\sigma}_e / \hat{w}_i$$

where $\hat{w}_i^2 = T_i \hat{\sigma}_v^2 + \hat{\sigma}_e^2$. Taking $\hat{\theta}_i$, you form the partial deviations $y_{it}^* = y_{it} - \hat{\theta}_i \bar{y}_{i.}$, and similarly to form x_{1it}^* , x_{2it}^* , and z_{it}^* . Estimation then proceeds as a 2SLS or two-step GMM estimation of y_{it}^* on x_{1it}^* and x_{2it}^* . The following instruments are used in the estimation: x_{1it}^* , $\bar{x}_{1i.}$, z_{it}^* , and $\bar{z}_{i.}$. When the GMM=TWOSTEP option is specified, the estimation of the variance by the Swamy-Arora method is based on 2SLS estimation of within-group estimation and between-group estimation.

A Hausman specification test that compares IV random effects to IV fixed effects is formed by using the method described in the section “[Hausman Specification Tests](#)” on page 521. Failure to reject the null hypothesis favors the random-effects specification.

Instrumental Variables Generalized Two-Stage Least Squares (IVG2SLS Option)

You perform IV generalized two-stage least squares estimation by specifying the IVG2SLS option in the MODEL statement. The model specification and the model estimation steps are as the same as they are for the IVRANONE option in the section “[Instrumental Variables One-Way Random Effects \(IVRANONE Option\)](#)” on page 509, except for the choice of instrumental variables.

According to Baltagi (1981), in the generalized two-stage least squares model, the following instruments are used in the estimation: x_{1it}^* and z_{it}^* . The instrument z_{it}^* always includes an intercept whether an intercept is specified or not. Asymptotically, the estimates from the IVG2SLS and IVRANONE options yield the same variance-covariance matrix. In a small sample, the IVRANONE option can be more efficient by including the redundant instruments $\bar{x}_{1i.}$ and $\bar{z}_{i.}$.

Instrumental Variables Two-Way Random Effects (IVRANTWO Option)

You perform IV two-way random-effects estimation by specifying the IVRANTWO option in the MODEL statement. The specification for the error term of the IV two-way random-effects model is

$$u_{it} = v_i + \lambda_t + e_{it}$$

where the v_i are iid with zero mean and variance σ_v^2 , the λ_t are iid with zero mean and variance σ_λ^2 , and the e_{it} are iid with zero mean and variance σ_e^2 . Furthermore, a random-effects specification assumes that the error terms are mutually uncorrelated and that each error term is uncorrelated with $\mathbf{X}$.

IVRANTWO deals only with balanced panel data. The estimation proceeds in three steps, and the details are the same as they are for the RANTWO model in Baltagi (2013, sec. 3.3). First, estimate the variance components σ_v^2 , σ_λ^2 , and σ_e^2 . Define $Q_1 = I_N \otimes I_T$, $Q_2 = I_N \otimes \bar{J}_T$, $Q_3 = \bar{J}_N \otimes I_T$, $Q_4 = \bar{J}_N \otimes \bar{J}_T$, and $Q = Q_1 - Q_2 - Q_3 + k_0 Q_4$. If an intercept is specified, $k_0 = 1$; if an intercept is not specified, $k_0 = 0$. Run 2SLS regression on Qy , QX , and QZ to obtain the sum of squared errors for IVFIXTWO and name it sse_1 . Also define the degrees of freedom for IVFIXTWO to be df_1 . Run 2SLS regression on Q_2y , Q_2X , and Q_2Z to obtain the sum of squared errors for IVBTWNG and name it sse_2 . Also define the degrees of freedom for IVBTWNG to be df_2 . Run 2SLS regression on Q_3y , Q_3X , and Q_3Z to obtain the

sum of squared errors for IV between times and name it sse_3 . Also define the degrees of freedom for IV between times to be df_3 . Define

$$\hat{\lambda}_1 = \hat{\sigma}_e^2 = sse_1/df_1$$

$$\hat{\lambda}_2 = sse_2/df_2$$

$$\hat{\lambda}_3 = sse_3/df_3$$

The estimated variances are

$$\hat{\sigma}_v^2 = (\hat{\lambda}_2 - \hat{\lambda}_1)/T$$

$$\hat{\sigma}_\lambda^2 = (\hat{\lambda}_3 - \hat{\lambda}_1)/N$$

Define $\lambda_4 = T\sigma_v^2 + N\sigma_\lambda^2 + \sigma_e^2$, and obtain

$$\theta_1 = 1 - \sigma_e/\lambda_2^{1/2}$$

$$\theta_2 = 1 - \sigma_e/\lambda_3^{1/2}$$

$$\theta_3 = \theta_1 + \theta_2 + (\sigma_e/\lambda_4^{1/2}) - 1$$

Second, transform the data using θ_1 , θ_2 , and θ_3 . Define $Q_r = Q_1 - \theta_1 Q_2 - \theta_2 Q_3 + \theta_3 Q_4$. The transformed variables are obtained as $Q_r y$, $Q_r X$, and $Q_r Z$. Lastly, substitute the transformed variables into the general estimation equation that is shown in the section “[Instrumental Variables Regression](#)” on page 505 to obtain the estimates for the IVRANTWO model. Notice that if $\sigma_e^2 = 0$, this model collapses to the IVPOOLED model. If $\sigma_\lambda^2 = 0$ and $\sigma_v^2 = 0$, this model also collapses to the IVPOOLED model. If $\sigma_\lambda^2 = 0$ and $\sigma_v^2 > 0$, this model collapses to the IVRANONE model. If $\sigma_\lambda^2 > 0$ and $\sigma_v^2 = 0$, this model collapses to the IVRANONE model with respect to time.

Similar to the IVRANONE model, when the estimation option is two-step GMM, the variance components are obtained using 2SLS. When $\sigma_\lambda^2 > 0$, $\sigma_v^2 > 0$, and $\sigma_e^2 > 0$, the instrumental variables include only the transformed exogenous variable $\mathbf{X}_1^*$ and the transformed instrumental variable $\mathbf{Z}^*$ without the redundant instruments such as cross-section means and time means.

Dynamic Panel Estimation (DYNDIFF and DYNSSYS Options)

You perform dynamic panel estimation that uses first differences by specifying the DYNDIFF option in the MODEL statement. For dynamic panel estimation that uses a full system of difference and level equations, specify the DYNSSYS option. For an example of dynamic panel estimation, see [Example 11.5](#).

Dynamic panel models are regression models that include lagged versions of the dependent variable as covariates. Consider the following panel regression, which includes L lags of the dependent variable:

$$y_{it} = \sum_{j=1}^L \phi_j y_{i, t-j} + \sum_{k=1}^K x_{itk} \beta_k + v_i + \epsilon_{it}$$

The effect v_i is common to all observations for that individual, so it is correlated with any lagged y because it played a role in its realization. For this reason, lagged dependent variables are endogenous regressors and require special consideration.

First Differencing

For ease of notation, consider the special case $L = K = 1$. A first attempt to remove the source of the correlation would be to take first differences, which removes v_i . That is,

$$\Delta y_{it} = \phi \Delta y_{i,t-1} + \Delta x_{it} \beta + \eta_{it}$$

where $\Delta y_{it} = y_{i,t} - y_{i,t-1}$, $\Delta x_{it} = x_{i,t} - x_{i,t-1}$, and $\eta_{it} = \epsilon_{i,t} - \epsilon_{i,t-1}$. Even though the individual effects are removed, the problem of endogeneity persists because $\Delta y_{i,t-1}$ is correlated with the differenced error term η_{it} . That is because $\epsilon_{i,t-1}$ is a component of $y_{i,t-1}$ (Nickell 1981).

Arellano and Bond (1991) show that you can use the generalized method of moments (GMM) to obtain a consistent estimator. In GMM parlance, the moment condition that $E\{(\Delta y_{i,t-1})\eta_{it}\} = 0$ is violated. Estimation requires a set of instrumental variables that do meet their moment conditions and that can adequately predict $\Delta y_{i,t-1}$. A natural set of instruments is $y_{i,t-2}$ and all other previous realizations of y . These lags of y are not correlated with $\epsilon_{i,t-1}$ because they occurred before time $t - 1$. Given the autoregressive nature of the model, $y_{i,t-1}$ (and hence $\Delta y_{i,t-1}$) is well predicted by its previous values.

Begin with $t = 3$, the first time period in which the differenced model holds. The dynamic regression model for individual i can be expressed as

$$\mathbf{y}_i^d = \mathbf{X}_i^d \boldsymbol{\gamma} + \boldsymbol{\eta}_i^d$$

where

$$\mathbf{y}_i^d = \begin{pmatrix} \Delta y_{i3} \\ \Delta y_{i4} \\ \vdots \\ \Delta y_{iT} \end{pmatrix} \quad \mathbf{X}_i^d = \begin{pmatrix} \Delta y_{i2} & \Delta x_{i3} \\ \Delta y_{i3} & \Delta x_{i4} \\ \vdots & \vdots \\ \Delta y_{i,T-1} & \Delta x_{iT} \end{pmatrix} \quad \boldsymbol{\gamma} = \begin{pmatrix} \phi \\ \beta \end{pmatrix} \quad \boldsymbol{\eta}_i^d = \begin{pmatrix} \eta_{i3} \\ \eta_{i4} \\ \vdots \\ \eta_{iT} \end{pmatrix}$$

Proceeding with the idea that you can use $(y_{i1}, \dots, y_{i,t-2})$ as instruments for the endogenous covariate $\Delta y_{i,t-1}$ for $t = 3, \dots, T$, the instrument matrix for the lagged dependent variables is

$$\mathbf{Z}_i^d = \begin{pmatrix} y_{i1} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & y_{i1} & y_{i2} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & y_{i1} & y_{i2} & y_{i3} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & y_{i1} & \cdots & y_{i,T-2} \end{pmatrix}$$

This extends naturally to $L > 1$ and $K > 1$; simply add columns to $\mathbf{X}_i^d$ and elements to $\boldsymbol{\gamma}$ as appropriate. When an observation is either missing or lost because of missing lags, delete the corresponding rows of $\mathbf{y}_i^d$, $\mathbf{X}_i^d$, $\boldsymbol{\eta}_i^d$, and $\mathbf{Z}_i^d$. Even if an observation is not missing with respect to the regression model, some of the lagged instruments might not be available because previous observations are missing. When that occurs, replace any missing instrument with 0.

When you specify the DYNDIFF option in the MODEL statement, PROC CPANEL by default treats x variables as exogenous and uses a projection that leaves these variables unchanged in the differenced regression. The full instrument matrix is then $\mathbf{Z}_i = (\mathbf{Z}_i^d, \mathbf{D}_i)$, where

$$\mathbf{D}_i = \begin{pmatrix} \Delta x_{i31} & \Delta x_{i32} & \cdots & \Delta x_{i3K} \\ \Delta x_{i41} & \Delta x_{i42} & \cdots & \Delta x_{i4K} \\ \vdots & \vdots & \vdots & \vdots \\ \Delta x_{iT1} & \Delta x_{iT2} & \cdots & \Delta x_{iT K} \end{pmatrix}$$

When $L = 1$, the default $\mathbf{Z}_i$ has $(T-1)(T-2)/2 + K$ columns. Each column $\mathbf{z}_c$ of $\mathbf{Z}_i$ satisfies the moment condition $E(\mathbf{z}_c' \boldsymbol{\eta}_i^d) = 0$.

System GMM

Blundell and Bond (1998) proposed a system GMM estimator that uses additional moment conditions to increase efficiency. The efficiency gain can be substantial when there is strong serial correlation in the dependent variable.

When either ϕ is near 1 or $\sigma_v^2/\sigma_\epsilon^2$ is large, the lagged dependent variables $(y_{i1}, \dots, y_{i, t-2})$ are weak instruments for the differenced variable $\Delta y_{i, t-1}$. System GMM solves the weak instrument problem by augmenting the difference equations described previously with a set of *level equations*. When $L = K = 1$, the level equations are

$$\mathbf{y}_i^\ell = \mathbf{X}_i^\ell \boldsymbol{\gamma} + \boldsymbol{\epsilon}_i^\ell$$

where

$$\mathbf{y}_i^\ell = \begin{pmatrix} y_{i2} \\ y_{i3} \\ \vdots \\ y_{iT} \end{pmatrix} \quad \mathbf{X}_i^\ell = \begin{pmatrix} y_{i1} & x_{i2} \\ y_{i2} & x_{i3} \\ \vdots & \vdots \\ y_{i, T-1} & x_{iT} \end{pmatrix} \quad \boldsymbol{\epsilon}_i^\ell = \begin{pmatrix} v_i + \epsilon_{i2} \\ v_i + \epsilon_{i3} \\ \vdots \\ v_i + \epsilon_{iT} \end{pmatrix}$$

Blundell and Bond (1998) note that you can use lagged differences of y as instruments for the levels of y . The main instrument matrix for the level equations is then

$$\mathbf{Z}_i^\ell = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & \Delta y_{i2} & 0 & \cdots & 0 \\ 0 & 0 & \Delta y_{i3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \Delta y_{i, T-1} \end{pmatrix}$$

where the first row corresponds to time $t = 2$. You can extend this to $L > 1$ and $K > 1$ by adding columns to $\mathbf{X}_i^\ell$ and elements to $\boldsymbol{\gamma}$ as appropriate. Higher-order lags require deletion of the leading rows of $\mathbf{y}_i^\ell$, $\mathbf{X}_i^\ell$, $\boldsymbol{\epsilon}_i^\ell$, and $\mathbf{Z}_i^\ell$.

Regression on the full system is obtained by stacking $\mathbf{y}_i^d$ and $\mathbf{y}_i^\ell$ to form $\mathbf{y}_i^s$, stacking $\mathbf{X}_i^d$ and $\mathbf{X}_i^\ell$ to form $\mathbf{X}_i^s$, and stacking $\boldsymbol{\eta}_i^d$ and $\boldsymbol{\epsilon}_i^\ell$ to form $\boldsymbol{\epsilon}_i^s$.

When you specify the DYNSSYS model option, the default instrument matrix for the full system is

$$\mathbf{Z}_i = \begin{pmatrix} \mathbf{Z}_i^d & \mathbf{0} & \mathbf{D}_i \\ \mathbf{0} & \mathbf{Z}_i^\ell & \mathbf{0} \end{pmatrix}$$

Estimation

The estimation in this section assumes system GMM. To obtain difference GMM, restrict estimation to the rows that correspond to the difference equations.

The initial moment matrix is derived from the theoretical variance of the combined residuals and is expressed as $\mathbf{H}_{1i} = \text{diag}(\mathbf{G}_{1i}, \mathbf{G}_{2i})$, where

$$\mathbf{G}_{1i} = \begin{pmatrix} 1 & -0.5 & 0 & \cdots & 0 & 0 & 0 \\ -0.5 & 1 & -0.5 & \cdots & 0 & 0 & 0 \\ 0 & -0.5 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -0.5 & 0 \\ 0 & 0 & 0 & \cdots & -0.5 & 1 & -0.5 \\ 0 & 0 & 0 & \cdots & 0 & -0.5 & 1 \end{pmatrix}$$

and $\mathbf{G}_{2i}$ is 0.5 times the identity matrix.

Define the weighting matrix as

$$\mathbf{W}_1 = \left(\sum_{i=1}^N \mathbf{Z}_i' \mathbf{H}_{1i} \mathbf{Z}_i \right)^{-1}$$

and define the projections as

$$\mathbf{P}_y = \sum_{i=1}^N \mathbf{Z}_i' \mathbf{y}_i^s; \quad \mathbf{P}_x = \sum_{i=1}^N \mathbf{Z}_i' \mathbf{X}_i^s$$

The one-step GMM estimate of $\boldsymbol{\gamma}$ is the weighted OLS estimator

$$\hat{\boldsymbol{\gamma}}_1 = \left(\mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_x \right)^{-1} \mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_y$$

The variance of $\hat{\boldsymbol{\gamma}}_1$ is

$$\text{Var}(\hat{\boldsymbol{\gamma}}_1) = \hat{\sigma}_\epsilon^2 \left(\mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_x \right)^{-1}$$

where $\hat{\sigma}_\epsilon^2$ is the mean square error (MSE) that is derived solely from the difference equations, namely

$$\hat{\sigma}_\epsilon^2 = (M - K)^{-1} \sum_{i=1}^N \left(\mathbf{y}_i^d - \mathbf{X}_i^d \hat{\boldsymbol{\gamma}}_1 \right)' \left(\mathbf{y}_i^d - \mathbf{X}_i^d \hat{\boldsymbol{\gamma}}_1 \right)$$

The total number of observations, M , is equal to the number of observations for which the difference equations hold.

A disadvantage of $\hat{\boldsymbol{\gamma}}_1$ is its reliance on the theoretical basis of $\mathbf{H}_{1i}$. The two-step GMM estimate of $\boldsymbol{\gamma}$ replaces $\mathbf{H}_{1i}$ with a version that is obtained from the observed one-step residuals. Let $\mathbf{H}_{2i}$ be the outer product of $\hat{\boldsymbol{\epsilon}}_i^s = \mathbf{y}_i^s - \mathbf{X}_i^s \hat{\boldsymbol{\gamma}}_1$. Then

$$\hat{\boldsymbol{\gamma}}_2 = \left(\mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_x \right)^{-1} \mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_y$$

where

$$\mathbf{W}_2 = \left(\sum_{i=1}^N \mathbf{Z}_i' \mathbf{H}_{2i} \mathbf{Z}_i \right)^{-1}$$

The variance of $\hat{\boldsymbol{\gamma}}_2$ is

$$\text{Var}(\hat{\boldsymbol{\gamma}}_2) = \left(\mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_x \right)^{-1}$$

Robust variances are calculated by the sandwich method. The robust variance of $\hat{\boldsymbol{\gamma}}_1$ is

$$\text{Var}^r(\hat{\boldsymbol{\gamma}}_1) = \left(\mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_x \right)^{-1} \mathbf{P}_x' \mathbf{W}_1 \mathbf{W}_2^{-1} \mathbf{W}_1 \mathbf{P}_x \left(\mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_x \right)^{-1}$$

The robust variance of $\hat{\boldsymbol{\gamma}}_2$ is

$$\text{Var}^r(\hat{\boldsymbol{\gamma}}_2) = \left(\mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_x \right)^{-1} \mathbf{P}_x' \mathbf{W}_2 \mathbf{W}_3^{-1} \mathbf{W}_2 \mathbf{P}_x \left(\mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_x \right)^{-1}$$

where

$$\mathbf{W}_3 = \left(\sum_{i=1}^N \mathbf{Z}_i' \mathbf{H}_{3i} \mathbf{Z}_i \right)^{-1}$$

and $\mathbf{H}_{3i}$ is the outer product of $\mathbf{y}_i^s - \mathbf{X}_i^s \hat{\boldsymbol{\gamma}}_2$.

Arellano and Bond (1991), among others, note that robust two-step variance estimators are biased. Windmeijer (2005) derived a bias-corrected variance of $\hat{\boldsymbol{\gamma}}_2$; you can obtain this correction by specifying the `BIASCORRECTED` option in the `MODEL` statement.

Define the one-step and two-step residuals as $\hat{\mathbf{e}}_{1i} = \mathbf{y}_i^s - \mathbf{X}_i^s \hat{\boldsymbol{\gamma}}_1$ and $\hat{\mathbf{e}}_{2i} = \mathbf{y}_i^s - \mathbf{X}_i^s \hat{\boldsymbol{\gamma}}_2$, respectively. Also define the projected two-step residual as

$$\mathbf{P}_e = \sum_{i=1}^N \mathbf{Z}_i' \hat{\mathbf{e}}_{2i}$$

Formulate the matrix $\mathbf{D}$ such that its k th column is $\mathbf{D}_k = \mathbf{V}_2 \mathbf{P}_x' \mathbf{W}_2 \mathbf{F}_k \mathbf{W}_2 \mathbf{P}_e$, where $\mathbf{V}_2 = \text{Var}(\hat{\boldsymbol{\gamma}}_2)$. The matrix $\mathbf{F}_k$ is the quadratic form

$$\mathbf{F}_k = \sum_{i=1}^N \mathbf{Z}_i' \left(\mathbf{x}_{ik} \hat{\mathbf{e}}_{1i}' + \hat{\mathbf{e}}_{1i} \mathbf{x}_{ik}' \right) \mathbf{Z}_i$$

where $\mathbf{x}_{ik}$ is the k th column of $\mathbf{X}_i^s$.

The Windmeijer (2005) bias-corrected variance is

$$\text{Var}^w(\hat{\boldsymbol{\gamma}}_2) = \mathbf{V}_2 + \mathbf{D} \mathbf{V}_2 + \mathbf{V}_2 \mathbf{D}' + \mathbf{D} \mathbf{V}_1' \mathbf{D}'$$

where $\mathbf{V}_1'$ is the robust variance estimate of $\hat{\boldsymbol{\gamma}}_1$.

Estimating the Intercept

The intercept term vanishes when you take first differences and is thus identified only in the level equations. If you specify the DYNDIFF option in the MODEL statement and your model includes an intercept, then PROC CPANEL fits the model by using system GMM with the (default) instrumentation

$$\mathbf{Z}_i = \begin{pmatrix} \mathbf{Z}_i^d & \mathbf{D}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{j}_i \end{pmatrix}$$

where $\mathbf{j}_i$ is a column of ones. Because all the level instruments are zero except the constant, parameter estimates other than the intercept are unaffected by the added level equations.

If you specify the DYNDIFF option in the MODEL statement and your model does not include an intercept, then the level equations are excluded from the estimation.

If you specify the DYNSSYS option in the MODEL statement, then there is no issue regarding the intercept. Under the default instrument specification, if $\mathbf{X}_i^\ell$ includes an intercept, then the level instruments include an added column of ones. That is,

$$\mathbf{Z}_i = \begin{pmatrix} \mathbf{Z}_i^d & \mathbf{0} & \mathbf{D}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{Z}_i^\ell & \mathbf{0} & \mathbf{j}_i \end{pmatrix}$$

Customizing Instruments

When you specify the DYNSSYS option to perform system GMM, the default instrument matrix is

$$\mathbf{Z}_i = \begin{pmatrix} \mathbf{Z}_i^d & \mathbf{0} & \mathbf{D}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{Z}_i^\ell & \mathbf{0} & \mathbf{c}_i \end{pmatrix}$$

where $\mathbf{c}_i$ is a column of ones or, if you specify the NOINT option, $\mathbf{c}_i$ is $\mathbf{0}$.

You can override the default set of instruments by specifying the ENDOGENOUS, PREDTERMINED, and INSTRUMENTS statements. The following discussion assumes that you are performing system GMM by using the DYNSSYS option in the MODEL statement. When you specify the DYNDIFF option instead, any specification (except the constant $\mathbf{c}_i$) that pertains to the level equations is ignored.

Endogenous Variables

A variable x_{it} is endogenous if $E(x_{it} \epsilon_{is}) \neq 0$ for $s \leq t$ and 0 otherwise.

The ENDOGENOUS statement specifies a list of endogenous variables that are used to form instrument matrices. For the difference equations, the instruments are “GMM-style” and mirror the form that is used in $\mathbf{Z}_i^d$ for the dependent variable. Suppose that the model includes one lag of the dependent variable ($L = 1$). Specifying the variable X in the ENDOGENOUS statement adds the following instruments to the difference equations:

$$\mathbf{G}_i^d = \begin{pmatrix} x_{i1} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & x_{i1} & x_{i2} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{i1} & x_{i2} & x_{i3} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_{i1} & \cdots & x_{i, T-2} \end{pmatrix}$$

The first row corresponds to time $t = 3$. The instruments are in lagged levels.

For the level equations, the ENDOGENOUS statement produces instruments that mirror the form used in $\mathbf{Z}_i^\ell$ for the dependent variable. Suppose that the model includes one lag of the dependent variable ($L = 1$). Specifying the variable X in the ENDOGENOUS statement adds the following instruments to the level equations:

$$\mathbf{G}_i^\ell = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & \Delta x_{i2} & 0 & \cdots & 0 \\ 0 & 0 & \Delta x_{i3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \Delta x_{i, T-1} \end{pmatrix}$$

The first row corresponds to time $t = 2$. Because the instruments are used for the level equations, they are in lagged differences.

By default, the ENDOGENOUS statement generates instruments for both the difference equations and the level equations. You can specify the EQ=DIFF option to generate instruments for only the difference equations. You can specify the EQ=LEVEL option to generate instruments for only the level equations.

The following code fits a dynamic panel model by using difference equations. It includes GMM-style instruments for both the dependent variable Sales and the regression variable Price.

```
proc cpanel data = mylib.a;
  id State Year;
  endogenous Sales Price / eq = diff;
  model Sales = Price PopDensity / dyndiff;
run;
```

Predetermined Variables

A variable x_{it} is predetermined if $E(x_{it}\epsilon_{is}) \neq 0$ for $s < t$ and 0 otherwise.

The PREDETERMINED statement specifies a list of predetermined variables that are used to form instrument matrices. For the difference equations, the instruments are similar to those that are formed for endogenous variables, except that each observation contains an extra instrument that reflects orthogonality in the current time period. If $L = 1$, specifying the variable X in the PREDETERMINED statement adds the following instruments to the difference equations:

$$\mathbf{P}_i^d = \begin{pmatrix} x_{i1} & x_{i2} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & x_{i1} & x_{i2} & x_{i3} & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_{i1} & \cdots & x_{i, T-1} \end{pmatrix}$$

The first row corresponds to time $t = 3$.

For the level equations, the PREDETERMINED statement produces instruments that are similar to those in $\mathbf{Z}_i^\ell$, except that the lag is shifted up to reflect orthogonality in the current time period. If $L = 1$, specifying the variable X in the PREDETERMINED statement adds the following instruments to the level equations:

$$\mathbf{P}_i^\ell = \begin{pmatrix} \Delta x_{i2} & 0 & 0 & \cdots & 0 \\ 0 & \Delta x_{i3} & 0 & \cdots & 0 \\ 0 & 0 & \Delta x_{i4} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \Delta x_{i, T} \end{pmatrix}$$

The first row corresponds to time $t = 2$.

By default, the `PREDETERMINED` statement generates instruments for both the difference equations and the level equations. You can specify the `EQ=DIFF` option to generate instruments for only the difference equations. You can specify the `EQ=LEVEL` option to generate instruments for only the level equations.

The following code fits a dynamic panel model by using difference equations. The instrument set includes GMM-style instruments for both the endogenous dependent variable `Sales` and the predetermined regression variable `Price`.

```
proc cpanel data = mylib.a;
  id State Year;
  endogenous Sales / eq = diff;
  predetermined Price / eq = diff;
  model Sales = Price PopDensity / dyndiff;
run;
```

Exogenous Variables (Standard Instruments)

Exogenous variables are uncorrelated with both the level residuals and the differenced residuals. If a regression variable is exogenous, you might want to include that variable in the instrument set as a standard instrument. The `INSTRUMENTS` statement specifies a list of variables to be included as standard instruments.

For the difference equations, the standard instruments form the matrix $\mathbf{D}_i$, as described in the section “[First Differencing](#)” on page 512. These variables are usually exogenous regressors that you want to preserve under the projection to the instrument space. Because these instruments belong to the difference equations, the variables are automatically differenced.

For the level equations, the standard instruments are left untransformed. They usually correspond to external instruments that are not part of the main regression but that can be used as instruments for the regression variables in levels. If $L = 1$, specifying the `X1` and `X2` variables in the `INSTRUMENTS` statement adds the following instruments to the level equations:

$$\mathbf{L}_i = \begin{pmatrix} x_{i21} & x_{i22} \\ x_{i31} & x_{i32} \\ \vdots & \vdots \\ x_{iT1} & x_{iT2} \end{pmatrix}$$

The first row corresponds to time $t = 2$.

By default, the `INSTRUMENTS` statement generates instruments for both the difference equations and the level equations. You can specify the `EQ=DIFF` option to generate instruments for only the difference equations. You can specify the `EQ=LEVEL` option to generate instruments for only the level equations.

The following example illustrates how you would use the `ENDOGENOUS` and `INSTRUMENTS` statements to obtain the default set of instruments for system GMM:

```
proc cpanel data = mylib.a;
  id State Year;
  endogenous Sales / eq = both;
  instruments Price PopDensity / eq = diff;
  model Sales = Price PopDensity / dynsys;
run;
```

Note that the standard instruments for `Price` and `PopDensity` are included only in the difference equations.

Limiting the Number of Instruments

Arellano and Bond's (1991) technique of expanding instruments is a useful method of dealing with autocorrelation in the response variable. However, using too many instruments can bias the estimator. The number of instruments grows quadratically with the number of time periods, making computations less feasible for larger T .

By default, PROC CPANEL uses all available lags. You can limit the number of instruments by specifying the MAXBAND= option in the MODEL statement. For example, specifying MAXBAND=5 limits the number of GMM-style instruments to five per observation, for each variable. The MAXBAND= option applies to all GMM-style instruments: those from the ENDOGENOUS statement and those from the PREDETERMINED statement.

Sargan Test of Overidentifying Restrictions

A Sargan test is a referendum on your choice of instruments in a dynamic panel model. The Sargan test statistic for one-step GMM is

$$J = \frac{1}{\hat{\sigma}_\epsilon^2} \left(\sum_{i=1}^N \mathbf{Z}_i' \hat{\epsilon}_{1i} \right)' \mathbf{W}_1 \left(\sum_{i=1}^N \mathbf{Z}_i' \hat{\epsilon}_{1i} \right)$$

The Sargan test statistic for two-step GMM is

$$J = \left(\sum_{i=1}^N \mathbf{Z}_i' \hat{\epsilon}_{2i} \right)' \mathbf{W}_2 \left(\sum_{i=1}^N \mathbf{Z}_i' \hat{\epsilon}_{2i} \right)$$

The null hypothesis of the Sargan test is that the moment conditions (as defined by the columns $\mathbf{Z}_i$) hold, and thus $\mathbf{Z}_i$ form an adequate set of instruments. Under the null, J is distributed as χ^2 with degrees of freedom equal to the rank of $\mathbf{W}_c$ ($c = 1, 2$) minus the number of parameters K . The nominal rank of $\mathbf{W}_c$ is equal to the number of instruments. However, this number can be reduced because of collinearity and redundancy in the instrument specification. Furthermore, the maximum rank of $\mathbf{W}_2$ is N , regardless of the number of instruments.

You should treat Sargan tests with caution when robust variances are used in the estimation. The theoretical distribution of J does not hold under conditions that favor robust variances.

AR(m) Tests

An AR(m) test is a test for autocorrelation of order m in the model residuals. Let $\mathbf{R}_i^s$ be the working variance of the residuals from the full system. The precise definition of $\mathbf{R}_i^s$ depends on the GMM stage and whether robust variances are specified. Table 11.4 provides detailed definitions.

Table 11.4 Definitions of the Working Residual Variance

Estimator	$\mathbf{R}_i^s$
One-step	$\hat{\sigma}_\epsilon^2 \mathbf{H}_{1i}$
One-step, robust	$\mathbf{H}_{2i}$
Two-step	$\mathbf{H}_{2i}$
Two-step, robust	$\mathbf{H}_{3i}$

Define the residual vector

$$\hat{\mathbf{e}}_i = \begin{pmatrix} \hat{\eta}_i^d \\ \mathbf{0} \end{pmatrix}$$

where $\hat{\eta}_i^d = \mathbf{y}_i^d - \mathbf{X}_i^d \hat{\boldsymbol{\gamma}}_c$ are the residuals from the difference equations, evaluated at the final estimate of $\hat{\boldsymbol{\gamma}}_c$. The trailing zeros correspond to the level equations. Define $\hat{\boldsymbol{\omega}}_{mi}$ as a lagged version of $\hat{\mathbf{e}}_i$ such that the following statements are true:

1. The first m elements of $\hat{\boldsymbol{\omega}}_{mi}$ are 0.
2. The next $p - m$ elements of $\hat{\boldsymbol{\omega}}_{mi}$ are the first $p - m$ elements of $\hat{\mathbf{e}}_i$, where p is the number of difference equations.
3. The trailing elements of $\hat{\boldsymbol{\omega}}_{mi}$ that correspond to the level equations are 0.

Define the following:

$$\begin{aligned} \mathbf{P}_m &= \sum_{i=1}^N \mathbf{Z}_i' \mathbf{R}_i^s \hat{\boldsymbol{\omega}}_{mi} \\ \mathbf{Q}_m &= \sum_{i=1}^N \hat{\boldsymbol{\omega}}_{mi}' \mathbf{X}_i^s \end{aligned}$$

The AR(m) test statistic is $Z_m = k_{0m} \{k_{1m} + k_{2m} + k_{3m}\}^{-1/2}$, where

$$\begin{aligned} k_{0m} &= \sum_{i=1}^N \hat{\boldsymbol{\omega}}_{mi}' \hat{\mathbf{e}}_i \\ k_{1m} &= \sum_{i=1}^N \hat{\boldsymbol{\omega}}_{mi}' \mathbf{R}_i^s \hat{\boldsymbol{\omega}}_{mi} \\ k_{2m} &= -2\mathbf{Q}_m (\mathbf{P}_x' \mathbf{W}_c \mathbf{P}_x)^{-1} \mathbf{P}_x' \mathbf{W}_c \mathbf{P}_m \\ k_{3m} &= \mathbf{Q}_m \mathbf{V} \mathbf{Q}_m' \end{aligned}$$

The matrix $\mathbf{V}$ is the estimated variance matrix of the parameters, corresponding to the GMM stage that is specified, and either model-based, robust, or bias-corrected.

Under the null hypothesis of no autocorrelation, Z_m follows a standard normal distribution. Because of the differencing in the errors, well-specified models present autocorrelation of order $m = 1$, but any autocorrelation at higher orders indicates a violation of assumptions.

Poolability Test for Fixed Effects

A poolability test is an F test of the null hypothesis that all fixed effects are jointly 0; it is obtained by comparing fixed-effects estimates to those from pooled regression. The F statistic is

$$F = \frac{(\text{SSE}_r - \text{SSE}_u) / df_1}{\text{SSE}_u / df_2} \sim F(df_1, df_2)$$

where SSE_r is the error sum of squares from the restricted model (pooled regression) and SSE_u is the error sum of squares from the unrestricted fixed-effects model.

The numerator degrees of freedom, df_1 , equals $N - 1$ for one-way models and $(N - 1) + (T - 1)$ for two-way models. The denominator degrees of freedom, df_2 , is equal to the error degrees of freedom from the fixed-effects estimation. If you specify the NOINT option, add 1 to df_1 to account for the added restriction to the pooled regression.

Hausman Specification Tests

For models that include random effects, the CPANEL procedure outputs the results of the Hausman (1978) specification test. This test was also proposed by Wu (1973) and further extended in Hausman and Taylor (1982).

Consider two estimators, $\hat{\beta}_e$ and $\hat{\beta}_c$, which under the null hypothesis are both consistent but only $\hat{\beta}_e$ is asymptotically efficient. Under the alternative hypothesis, only $\hat{\beta}_c$ is consistent. The m statistic is

$$m = (\hat{\beta}_c - \hat{\beta}_e)' (\hat{\Sigma}_c - \hat{\Sigma}_e)^{-1} (\hat{\beta}_c - \hat{\beta}_e)$$

where $\hat{\Sigma}_c$ and $\hat{\Sigma}_e$ are estimates of the asymptotic covariance matrices of $\hat{\beta}_c$ and $\hat{\beta}_e$. The statistic m follows a χ^2 distribution with k degrees of freedom, where k is the rank of $(\hat{\Sigma}_c - \hat{\Sigma}_e)^{-1}$. This rank is normally equal to the dimension of $\hat{\beta}_c - \hat{\beta}_e$, but is reduced when regressors that are constant within cross sections are dropped from the fixed-effects model.

The null hypothesis is that the effects are independent of the regressors. Under the null hypothesis, the fixed-effects estimator is consistent yet inefficient, whereas the random-effects estimator is both consistent and efficient. Failure to reject the null hypothesis favors the random-effects specification.

Restricted Estimation

The CPANEL procedure can fit models that have linear restrictions, producing a Lagrange multiplier (LM) test for each restriction. Consider a set of J linear restrictions $\mathbf{R}\beta = \mathbf{q}$, where $\mathbf{R}$ is $J \times K$ and $\mathbf{q}$ is $J \times 1$.

The restricted regression is performed by minimizing the error sum of squares subject to the restrictions. In matrix terms, the Lagrangian for this problem is

$$L = (\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta) + 2\lambda(\mathbf{R}\beta - \mathbf{q})$$

The Lagrangian is minimized by the restricted estimator β^* , and it can be shown that

$$\beta^* = \hat{\beta} - (\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}'\lambda$$

where $\hat{\beta}$ is the unrestricted estimator.

Because $\mathbf{R}\beta^* = \mathbf{q}$, you can solve for λ to obtain the Lagrange multipliers

$$\lambda^* = \left[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}' \right]^{-1} (\mathbf{R}\hat{\beta} - \mathbf{q})$$

The standard errors of the Lagrange multipliers are the square roots of the diagonal elements of the variance matrix

$$\text{Var}(\lambda^*) = \hat{\sigma}_e^2 \left[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}' \right]^{-1}$$

where $\hat{\sigma}_e^2$ is the mean square error (MSE) under the null hypothesis. A significant Lagrange multiplier indicates a restriction that is binding.

For each restriction, the Lagrange multiplier test is represented by a row in the parameter estimates table that displays the Lagrange multiplier, standard error, test statistic, and p -value. Although LM test statistics converge to a chi-square distribution under the null hypothesis, PROC CPANEL computes p -values by using the beta distribution, which is the exact distribution for finite samples.

Lagrange multiplier tests are not appropriate for instrumental variables regression and dynamic panel models. If you fit such a model, the linear restrictions are applied, but no formal test of their validity is performed.

Linear Hypothesis Testing

Consider a linear hypothesis of the form $\mathbf{R}\beta = \mathbf{q}$, where $\mathbf{R}$ is $J \times K$ and $\mathbf{q}$ is $J \times 1$. The Wald test statistic is

$$\chi_W^2 = (\mathbf{R}\hat{\beta} - \mathbf{q})' (\mathbf{R}\hat{\mathbf{V}}\mathbf{R}')^{-1} (\mathbf{R}\hat{\beta} - \mathbf{q})$$

where $\hat{\mathbf{V}}$ is the estimated variance of $\hat{\beta}$.

In simple linear models, the Wald test statistic is equal to the F test statistic

$$F = \frac{(\text{SSE}_r - \text{SSE}_u)/J}{\text{SSE}_u/df_e}$$

where SSE_r is the restricted error sum of squares, SSE_u is the unrestricted error sum of squares, and df_e is the unrestricted error degrees of freedom.

The F statistic represents a more direct comparison of the restricted model to the unrestricted model. Comparing error sums of squares is appealing in complex models for which restrictions are applied not only during the final regression but also during intermediate calculations.

The likelihood ratio (LR) test and the Lagrange multiplier (LM) test are derived from the F statistic. The LR test statistic is

$$\chi_{\text{LR}}^2 = M \ln \left[1 + \frac{JF}{M - K} \right]$$

The LM test statistic is

$$\chi_{\text{LM}}^2 = M \left[\frac{JF}{M - K + JF} \right]$$

The distribution of these test statistics is χ^2 with J degrees of freedom. The three tests are asymptotically equivalent, but they possess different small-sample properties. For more information, see Greene (2000, p. 392) and Davidson and MacKinnon (1993, pp. 456–458).

Heteroscedasticity-Corrected Covariance Matrices

The HCCME= option in the MODEL statement selects the type of heteroscedasticity-consistent covariance matrix. In the presence of heteroscedasticity, the covariance matrix has a complicated structure that can result in inefficiencies in the OLS estimates and biased estimates of the covariance matrix. The variances for cross-sectional and time dummy variables and the covariances with or between the dummy variables are not corrected for heteroscedasticity in the one-way and two-way models. Whether or not you specify the HCCME= option, these variances are the same. For the two-way models, the variance and the covariances for the intercept are not corrected.¹

Consider the following simple linear model:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

This discussion parallels the discussion in Davidson and MacKinnon (1993, pp. 548–562). For panel data models, heteroscedasticity-corrected covariance matrix estimation (HCCME) is applied to the transformed data ($\tilde{\mathbf{y}}$ and $\tilde{\mathbf{X}}$). In other words, first the random or fixed effects are removed through transforming the data,² and then the heteroscedasticity (also autocorrelation with the HAC option) is corrected in the residual. The assumptions that make the linear regression best linear unbiased estimator (BLUE) are $E(\boldsymbol{\epsilon}) = 0$ and $E(\boldsymbol{\epsilon}\boldsymbol{\epsilon}') = \Omega$, where Ω has the simple structure $\sigma^2\mathbf{I}$. Heteroscedasticity results in a general covariance structure, and it is not possible to simplify Ω . The result is as follows:

$$\tilde{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'(\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}) = \boldsymbol{\beta} + (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\boldsymbol{\epsilon}$$

As long as the following is true, you are assured that the OLS estimate is consistent and unbiased:

$$\text{plim}_{n \rightarrow \infty} \left(\frac{1}{n} \mathbf{X}'\boldsymbol{\epsilon} \right) = 0$$

If the regressors are nonrandom, then it is possible to write the variance of the estimated $\boldsymbol{\beta}$ as

$$\text{Var}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\Omega\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$$

You can ameliorate the effect of structure in the covariance matrix by using generalized least squares (GLS), provided that Ω^{-1} can be calculated. Using Ω^{-1} , you premultiply both sides of the regression equation,

$$\mathbf{L}^{-1}\mathbf{y} = \mathbf{L}^{-1}\mathbf{X}\boldsymbol{\beta} + \mathbf{L}^{-1}\boldsymbol{\epsilon}$$

¹The dummy variables are removed by the within transformations, so their variances and covariances cannot be calculated the same way as the other regressors. They are recovered by the formulas in the sections “One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)” on page 494 and “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 496. The formulas assume homoscedasticity, so they do not apply when HCCME is used. Therefore, standard errors, variances, and covariances are reported only when the HCCME= option is ignored. HCCME standard errors for dummy variables and intercept can be calculated by the dummy variable approach with the pooled model.

²For more information about transforming the data, see the sections “One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)” on page 494, “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 496, “One-Way Random-Effects Model (RANONE Option)” on page 497, and “Two-Way Random-Effects Model (RANTWO Option)” on page 500.

where L denotes the Cholesky root of Ω (that is, $\Omega = LL'$ with L lower triangular).

The resulting GLS β is

$$\hat{\beta} = (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\mathbf{y}$$

Using the GLS β , you can write

$$\begin{aligned}\hat{\beta} &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\mathbf{y} \\ &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'(\Omega^{-1}\mathbf{X}_e + \Omega^{-1}\epsilon) \\ &= \beta + (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\epsilon\end{aligned}$$

The resulting variance expression for the GLS estimator is

$$\begin{aligned}\text{Var}(\beta - \hat{\beta}) &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\epsilon\epsilon'\Omega^{-1}\mathbf{X}(\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1} \\ &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\Omega\Omega^{-1}\mathbf{X}(\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1} \\ &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\end{aligned}$$

The difference in variance between the OLS estimator and the GLS estimator can be written as

$$(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\Omega\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} - (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}$$

By the Gauss-Markov theorem, the difference matrix must be positive definite under most circumstances (zero if OLS and GLS are the same, when the usual classical regression assumptions are met). Thus, OLS is not efficient for a general error structure. It is crucial to realize that OLS does not produce biased results. It would suffice if you had a method of estimating a consistent covariance matrix and you used the OLS β . Estimation of the Ω matrix is certainly not simple. The matrix is square and has M^2 elements; unless some sort of structure is assumed, it becomes an impossible problem to solve. However, the heteroscedasticity can have quite a general structure. White (1980) shows that it is not necessary to have a consistent estimate of Ω . On the contrary, it suffices to calculate an estimate of the middle expression. That is, you need an estimate of

$$\Lambda = \mathbf{X}'\Omega\mathbf{X}$$

This matrix, Λ , is easier to estimate because its dimension is K . PROC CPANEL provides the following classical HCCME estimators for Λ . The approximation of the matrix depends on the value of the HCCME= option.

- HCCME=0

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \hat{\epsilon}_{it}^2 \mathbf{x}_{it} \mathbf{x}_{it}'$$

Here N is the number of cross sections and T_i is the number of observations in the i th cross section. The $\mathbf{x}_{it}'$ is from the t th observation in the i th cross section, constituting the $(\sum_{j=1}^{i-1} T_j + t)$ th row of the matrix $\mathbf{X}$. If you specify the CLUSTER option, one extra term is added to the preceding equation so that the estimator of matrix Λ is

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \hat{\epsilon}_{it}^2 \mathbf{x}_{it} \mathbf{x}_{it}' + \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{s=1}^{t-1} \hat{\epsilon}_{it} \hat{\epsilon}_{is} (\mathbf{x}_{it} \mathbf{x}_{is}' + \mathbf{x}_{is} \mathbf{x}_{it}')$$

The formula is the same as the formula for the robust variance matrix estimator in Wooldridge (2002, p. 152), and it is derived under the assumptions of section 7.3.2 of Wooldridge (2002). You can also obtain the preceding clustered standard error with HCCME=0 by specifying the ROBUST option alone.

- HCCME=1

$$\frac{M}{M-K} \sum_{i=1}^N \sum_{t=1}^{T_i} \hat{\epsilon}_{it}^2 \mathbf{x}_{it} \mathbf{x}_{it}'$$

Here M is the total number of observations, $\sum_{j=1}^N T_j$, and K is the number of parameters. If you specify the CLUSTER option, the estimator becomes

$$\frac{M}{M-K} \sum_{i=1}^N \sum_{t=1}^{T_i} \hat{\epsilon}_{it}^2 \mathbf{x}_{it} \mathbf{x}_{it}' + \frac{M}{M-K} \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{s=1}^{t-1} \hat{\epsilon}_{it} \hat{\epsilon}_{is} (\mathbf{x}_{it} \mathbf{x}_{is}' + \mathbf{x}_{is} \mathbf{x}_{it}')$$

This formula is similar to the formula for the robust variance matrix estimator in Wooldridge (2002, p. 152) with the heteroscedasticity adjustment term $M/(M-K)$.

- HCCME=2

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \frac{\hat{\epsilon}_{it}^2}{1 - \hat{h}_{it}} \mathbf{x}_{it} \mathbf{x}_{it}'$$

The $\hat{h}_{it}$ term is the $(\sum_{j=1}^{i-1} T_j + t)$ th diagonal element of the hat matrix. The expression for $\hat{h}_{it}$ is $\mathbf{x}_{it}'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_{it}$. The hat matrix attempts to adjust the estimates for the presence of influence or leverage points. If you specify the CLUSTER option, the estimator becomes

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \frac{\hat{\epsilon}_{it}^2}{1 - \hat{h}_{it}} \mathbf{x}_{it} \mathbf{x}_{it}' + 2 \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{s=1}^{t-1} \frac{\hat{\epsilon}_{it}}{\sqrt{1 - \hat{h}_{it}}} \frac{\hat{\epsilon}_{is}}{\sqrt{1 - \hat{h}_{is}}} (\mathbf{x}_{it} \mathbf{x}_{is}' + \mathbf{x}_{is} \mathbf{x}_{it}')$$

This formula is similar to the formula for the robust variance matrix estimator in Wooldridge (2002, p. 152) with the heteroscedasticity adjustment.

- HCCME=3

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \frac{\hat{\epsilon}_{it}^2}{(1 - \hat{h}_{it})^2} \mathbf{x}_{it} \mathbf{x}_{it}'$$

If you specify the CLUSTER option, the estimator becomes

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \frac{\hat{\epsilon}_{it}^2}{(1 - \hat{h}_{it})^2} \mathbf{x}_{it} \mathbf{x}_{it}' + 2 \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{s=1}^{t-1} \frac{\hat{\epsilon}_{it}}{1 - \hat{h}_{it}} \frac{\hat{\epsilon}_{is}}{1 - \hat{h}_{is}} (\mathbf{x}_{it} \mathbf{x}_{is}' + \mathbf{x}_{is} \mathbf{x}_{it}')$$

This formula is similar to the formula for the robust variance matrix estimator in Wooldridge (2002, p. 152) with the heteroscedasticity adjustment.

In their discussion, Davidson and MacKinnon (1993, p. 554) argue that the formula specified by HCCME=1 should always be preferred to the formula specified by HCCME=0. Although an HCCME= option value of 3 is generally preferred to 2, and 2 is preferred to 1, the calculation of HCCME=1 is as simple as the calculation of HCCME=0. Therefore, HCCME=1 is preferred when the calculation of the hat matrix is too tedious.

All HCCMEs have well-defined asymptotic properties. The small-sample properties are not well known, and you must exercise care when sample sizes are small.

The HCCME of $\text{Var}(\boldsymbol{\beta})$ is used to drive the covariance matrices for the fixed effects and the Lagrange multiplier standard errors. Robust estimates of the covariance matrix for $\boldsymbol{\beta}$ imply robust covariance matrices for all other parameters.

Heteroscedasticity- and Autocorrelation-Consistent Covariance Matrices

You use the HAC option in the MODEL statement to select the type of heteroscedasticity- and autocorrelation-consistent covariance matrix. As with the HCCME= option, an estimator of the middle expression Λ in sandwich form is needed. Define $g_{i,t} = x_{it}\epsilon_{it}$ as the gradient vector of the t th observation in the i th cross section. For the HAC option, it is estimated as

$$\Lambda_{\text{HAC}} = a \sum_{i=1}^N \left(\sum_{t=1}^T g_{i,t} g'_{i,t} + \sum_{j=1}^{T-1} k\left(\frac{j}{b_i}\right) \sum_{t=1}^{T-j} (g_{i,t} g'_{i,t+j} + g_{i,t+j} g'_{i,t}) \right)$$

where $k(\cdot)$ is the real-valued kernel function, b_i is the bandwidth parameter for the i th cross section, and a is the adjustment factor of small-sample degrees of freedom (that is, $a = 1$ if you do not specify the ADJUSTDF option, and $a = NT/(NT - k)$ otherwise, where k is the number of parameters, including dummy variables). If the BANDWIDTH option is specified as a number, b_i is equal to the same number for every cross section. The types of kernel functions are listed in Table 11.5.

Table 11.5 Kernel Functions

Kernel Type	Equation
Bartlett	$k(x) = \begin{cases} 1 - x & x \leq 1 \\ 0 & \text{otherwise} \end{cases}$
Parzen	$k(x) = \begin{cases} 1 - 6x^2 + 6 x ^3 & 0 \leq x \leq 1/2 \\ 2(1 - x)^3 & 1/2 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$
Quadratic spectral	$k(x) = \frac{25}{12\pi^2 x^2} \left(\frac{\sin(6\pi x/5)}{6\pi x/5} - \cos(6\pi x/5) \right)$
Truncated	$k(x) = \begin{cases} 1 & x \leq 1 \\ 0 & \text{otherwise} \end{cases}$
Tukey-Hanning	$k(x) = \begin{cases} (1 + \cos(\pi x))/2 & x \leq 1 \\ 0 & \text{otherwise} \end{cases}$

When you specify the BANDWIDTH=ANDREWS option, the bandwidth parameter is estimated as shown in Table 11.6.

Table 11.6 Bandwidth Parameter Estimation

Kernel Type	Bandwidth Parameter
Bartlett	$b_i = 1.1447(\alpha(1)T)^{1/3}$
Parzen	$b_i = 2.6614(\alpha(2)T)^{1/5}$
Quadratic spectral	$b_i = 1.3221(\alpha(2)T)^{1/5}$
Truncated	$b_i = 0.6611(\alpha(2)T)^{1/5}$
Tukey-Hanning	$b_i = 1.7462(\alpha(2)T)^{1/5}$

The values $\alpha(1)$ and $\alpha(2)$ are calculated separately for each cross section by using the following steps. Let g_{ait} denote the a th row of g_{it} , and let (ρ_a, σ_a^2) denote the corresponding estimates of the autoregressive and innovation variance parameters of the AR(1) model on $\{g_{ait}\}$, $a = 1, \dots, k$, where the AR(1) model is parameterized as $g_{ait} = \rho_a g_{ait-1} + \epsilon_{ait}$ with $\text{Var}(\epsilon_{ait}) = \sigma_a^2$. Notice that ρ_a and σ_a^2 are different for each cross section. Obtain $\alpha(1)$ and $\alpha(2)$ by using

$$\alpha(1) = \frac{\sum_{a=1}^k \frac{4\rho_a^2 \sigma_a^4}{(1-\rho_a)^6(1+\rho_a)^2}}{\sum_{a=1}^k \frac{\sigma_a^4}{(1-\rho_a)^4}} \quad \alpha(2) = \frac{\sum_{a=1}^k \frac{4\rho_a^2 \sigma_a^4}{(1-\rho_a)^8}}{\sum_{a=1}^k \frac{\sigma_a^4}{(1-\rho_a)^4}}$$

When you specify BANDWIDTH=NEWKEYWEST94, according to Newey and West (1994) the bandwidth parameter is estimated as shown in Table 11.7.

Table 11.7 Bandwidth Parameter Estimation

Kernel Type	Bandwidth Parameter
Bartlett	$b_i = 1.1447(\{s_1/s_0\}^2 T)^{1/3}$
Parzen	$b_i = 2.6614(\{s_1/s_0\}^2 T)^{1/5}$
Quadratic spectral	$b_i = 1.3221(\{s_1/s_0\}^2 T)^{1/5}$
Truncated	$b_i = 0.6611(\{s_1/s_0\}^2 T)^{1/5}$
Tukey-Hanning	$b_i = 1.7462(\{s_1/s_0\}^2 T)^{1/5}$

The terms s_0 and s_1 are estimated separately for each cross section by using

$$s_0 = \sigma_0 + 2 \sum_{j=1}^n \sigma_j \quad s_1 = 2 \sum_{j=1}^n j \sigma_j$$

where n is the lag selection parameter and is determined by kernels for each cross section, as listed in Table 11.8.

Table 11.8 Lag Selection Parameter Estimation

Kernel Type	Lag Selection Parameter
Bartlett	$n = c(T/100)^{2/9}$
Parzen	$n = c(T/100)^{4/25}$

Table 11.8 *continued*

Kernel Type	Lag Selection Parameter
Quadratic spectral	$n = c(T/100)^{2/25}$
Truncated	$n = c(T/100)^{1/5}$
Tukey-Hanning	$n = c(T/100)^{1/5}$

The c in Table 11.8 is specified by the C= option; by default, C=12 for all cross sections.

The σ_j are estimated for the i th cross section by the equation

$$\sigma_j = T^{-1} \sum_{t=j+1}^T \left(\sum_{a=s}^k g_{ait} \sum_{a=s}^k g_{ait-j} \right), \quad j = 0, \dots, n$$

where g_{ait} is the same as in the Andrews method and s is 1 if you specify the NOINT option in the MODEL statement, and 2 otherwise.

When you specify BANDWIDTH=SAMPLESIZE, the bandwidth parameter is estimated by the equation

$$b_i = \begin{cases} \lfloor \gamma T^r + c \rfloor & \text{if the BANDWIDTH=SAMPLESIZE(INTSS) option is specified} \\ \gamma T^r + c & \text{otherwise} \end{cases}$$

where T is the sample size; $\lfloor x \rfloor$ is the largest integer less than or equal to x ; and γ , r , and c are values specified by the BANDWIDTH=SAMPLESIZE(GAMMA= RATE= CONSTANT=) options, respectively. These specified values are the same for every cross section.

If you specify the PREWHITENING option in the MODEL statement, g_{it} is prewhitened by the VAR(1) model,

$$g_{it} = A_i g_{it-1} + w_{it}$$

Then Λ_{HAC} is calculated by

$$\Lambda_{\text{HAC}} = a \sum_{i=1}^N \left\{ \left(\sum_{t=1}^T w_{it} w'_{it} + \sum_{t=1}^T \sum_{s=1}^{t-1} k\left(\frac{s-t}{b_i}\right) (w_{it} w'_{is} + w_{is} w'_{it}) \right) (I - A_i)^{-1} ((I - A_i)^{-1})' \right\}$$

For unbalanced data, the HAC option treats those missing values as 0.

R-Square

The R-square statistic is the proportion of variability in the dependent variable that is attributed to the independent variables. Because of the transformations that are used prior to fitting the final regression model, the conventional R-square measure is not appropriate for most of the models that the CPANEL procedure supports. In random-effects models that use a GLS transform, PROC CPANEL calculates the modified R-square statistic proposed by Buse (1973),

$$R^2 = 1 - \frac{\text{SSE}}{\mathbf{y}'\mathbf{D}'\hat{\mathbf{\Omega}}^{-1}\mathbf{D}\mathbf{y}}$$

where SSE is the error sum of squares from the final model fit, $\hat{\mathbf{\Omega}}^{-1/2}$ represents the GLS transform, and $\mathbf{D} = \mathbf{I}_M - a^{-1}\mathbf{J}_M\hat{\mathbf{\Omega}}^{-1}$, for $a = \mathbf{j}_M'\hat{\mathbf{\Omega}}^{-1}\mathbf{j}_M$.

In GLS models that do not have an intercept, the alternative R-square measure, which is attributed to Theil (1961), is calculated as follows:

$$R^2 = 1 - \frac{\text{SSE}}{\mathbf{y}'\hat{\mathbf{\Omega}}^{-1}\mathbf{y}}$$

In fixed-effects models, the R-square measure is

$$R^2 = 1 - \frac{\text{SSE}}{\mathbf{y}_w'\mathbf{y}_w}$$

where $\mathbf{y}_w$ is the within-transformed dependent variable.

In the case of pooled OLS estimation, all three of the R-square formulas reduce to the usual R-square statistic for linear models.

The R-square statistic is not appropriate for IV regression and dynamic panel estimation, and thus it is not reported for these models.

Displayed Output

The following sections describe the output that PROC CPANEL produces. The output is organized into various tables, which are discussed in their order of appearance.

Class Level Information

The “Class Level Information” table lists the levels of every variable specified in the **CLASS** statement. You should check this information to make sure that the data are correct. You can adjust the order of the **CLASS** variable levels by using the **ORDER=** option in the **CLASS** statement.

Model Description

The “Model Information” table displays basic information about the model: the response variable, the name of the data table, the estimation method, the number of observations, the number of cross sections, and the length of the time series. The reported numbers are useful for verifying that observations were not dropped unexpectedly and for verifying that the data are balanced when they should be.

Fit Statistics

The “Fit Statistics” table displays model fit statistics that are widely used in linear models. This table reports the error sum of squares, mean square error, root mean square error, R-square statistic, and the error degrees of freedom.

Variance Component Estimates

If you fit a model that contains random effects, then the “Variance Component Estimates” table reports the variances and standard deviations of the observation-level errors, the cross-sectional effects, and the time effects (in two-way models).

Test for Poolability

If you fit a fixed-effects model, then an F test of poolability is produced. The null hypothesis is that all fixed effects are jointly 0, making pooled OLS regression a viable alternative. The “F Test for No Fixed Effects” table reports the numerator degrees of freedom, denominator degrees of freedom, test statistic, and p -value.

Hausman Specification Tests

If you fit a model that contains random effects, then a table that summarizes the Hausman test is produced. The table presents the number of coefficients included in the test, the degrees of freedom, the m -statistic, and the p -value.

If you fit a one-way random-effects model (RANONE), then the fitted model is compared to the alternative model, a one-way fixed-effects (FIXONE) model.

If you fit a two-way random-effects model (RANTWO), then the fitted model is compared to the alternative model, a two-way fixed-effects (FIXTWO) model.

If you fit a Hausman-Taylor model (HTAYLOR), then the fitted model is compared to the alternative model, a one-way fixed-effects (FIXONE) model.

If you fit an Amemiya-MaCurdy model (AMACURDY), then the fitted model is compared to the alternative model, a Hausman-Taylor (HTAYLOR) model.

If you fit an instrumental variables random-effects model (IVRANONE), then the fitted model is compared to the alternative model, an IV fixed-effects (IVFIXONE) model.

Structural Summary

If you fit an instrumental variables (IV) model, then the “Structural Summary” table lists the endogenous effects and the instrumental variables.

Instruments

If you fit a dynamic panel model, then the “Instruments” table lists the instruments that are used and tallies the columns of the instruments matrix. The instruments are categorized by the type of equation (difference or level) and by the type of variable (endogenous, predetermined, or exogenous).

Sargan Test

If you fit a dynamic panel model, then the CPANEL procedure computes a Sargan test of overidentifying restrictions. This test is a referendum on your choice of instruments, and the null hypothesis is that your choice is adequate. The “Sargan Test” table reports the degrees of freedom, test statistic, and p -value.

Parameter Estimates

The parameter estimates, degrees of freedom, standard errors, t statistics and p -values for the hypothesis that the parameter is 0 are presented in the “Parameter Estimates” table. If one or more of your regression variables are labeled, then a column that contains variable labels is added to this table. Use the [NOLABEL](#) option in the MODEL statement to suppress this column.

If you specify a RESTRICT statement, then the table includes a row for each linear restriction. This row reports the restriction label, Lagrange multiplier (LM) statistic, standard error, t statistic, p -value, and a label that describes the restriction.

If you fit an instrumental variables (IV) model, then the table includes a column that identifies the endogenous variables. If you fit a Hausman-Taylor or Amemiya-MaCurdy model, then both the correlated variables and the time-invariant variables are identified in the table.

AR(m) Tests

If you fit a dynamic panel model, then the CPANEL procedure performs tests for autocorrelation in the residuals. In well-fitting models, you can expect some autocorrelation of lag order 1, but any autocorrelation at higher lags indicates a violation of assumptions. By default, AR tests for the first two lags are reported, but you can request tests for more lags by specifying the ARTESTS= option in the MODEL statement. For each test, the “AR(m) Test” table reports the lag, test statistic, and p -value.

Parameter Estimates Covariance Matrix

When you specify the [COVB](#) option in the [PROC CPANEL](#) statement, the CPANEL procedure displays the covariance matrix of the parameter estimates.

Parameter Estimates Correlation Matrix

When you specify the **CORRB** option in the **PROC CPANEL** statement, the CPANEL procedure displays the correlation matrix of the parameter estimates.

Tests of Linear Hypotheses

If you specify a **TEST** statement, a table that describes the tests is provided. For each test of hypothesis, the table reports the test label, the type of test (Wald, LR, or LM), the test statistic, the *p*-value, and a label that states the null hypothesis.

Model Comparison Tables

If you specify a **COMPARE** statement, two comparison tables are produced.

The first table is the “Comparison of Model Statistics” table, which lists the number of observations, number of cross sections, time series length, mean square error (MSE), root MSE, and R-square for each fitted model.

The second is table is the “Comparison of Model Parameter Estimates” table, which presents the parameter estimate and standard error of each regressor for each fitted model.

Both tables can be customized; for more information, see the section “**COMPARE Statement**” on page 472.

ODS Table Names

Each table that the CPANEL procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. These names are listed in [Table 11.9](#).

Table 11.9 ODS Tables Produced by PROC CPANEL

Table Name	Description	Statement	Option
ARTest	Autoregression tests for the residuals	MODEL	DYNDIFF or DYN SYS
ClassInfo	Level information from the CLASS statement	CLASS	Default
CorrB	Correlation matrix of parameter estimates	MODEL	CORRB
CovB	Covariance matrix of parameter estimates	MODEL	COVB
FitStatistics	Fit statistics	Default	
FixedEffectsTest	<i>F</i> test for poolability	MODEL	FIXONE, FIXONETIME, FIXTWO, or IVFIXONE
HausmanTestForRandom	Hausman test for random effects	MODEL	RANONE, RANTWO, IVRANONE, or IVRANTWO

Table 11.9 *continued*

Table Name	Description	Statement	Option
HausmanTestVsFixed	Hausman test for Hausman-Taylor models	MODEL	HTAYLOR
HausmanTestVsHausmanTaylor	Hausman test for Amemiya-MaCurdy models	MODEL	AMACURDY
Instruments	Instruments summary	MODEL	DYNDIFF or DYSYS
ModelDescription	Model summary and data counts	Default	
ParameterComparisonTable	Comparison of model parameter estimates, standard errors, and <i>t</i> tests	COMPARE	
ParameterEstimates	Solutions for the parameter estimates associated with effects in MODEL statements	Default	
ParameterEstimatesLegend	Legend that defines the abbreviations that were used in the parameter estimates table	Default	
SarganTest	Sargan test for overidentifying restrictions	MODEL	DYNDIFF or DYSYS
StatComparisonTable	Comparison of model summary statistics	COMPARE	
StructuralSummary	Endogenous and instrumental variable lists	MODEL	IVPOOLED, IVBTWNG, IVFIXONE, IV-FIXONETIME, IVFIXTWO, IVG2SLS, IVRANONE, or IVRANTWO
TestResults	Hypothesis tests of linear restrictions	TEST	
VarianceComponents	Variance component estimates	MODEL	RANONE, RANTWO, HTAYLOR, AMACURDY, IVRANONE, or IVRANTWO

Examples: CPANEL Procedure

Example 11.1: Airline Cost Data: Fixed Effects

The Christenson Associates airline data are a frequently cited data set (Greene 2000). The data measure the costs, prices of inputs, and utilization rates for six airlines from 1970 to 1984. This example analyzes the log transformations of cost (variable C), quantity (variable Q), and price (variable PF), and the untransformed load factor (variable LF). You speculate the following model,

$$\log(C_{it}) = \alpha + \beta_1 \log(Q_{it}) + \beta_2 \log(PF_{it}) + \beta_3 LF_{it} + v_i + \epsilon_{it}$$

where the v_i are airline effects. The actual model in the original, untransformed variables is highly nonlinear:

$$C_{it} = \exp(\alpha + \beta_3 LF_{it} + v_i + \epsilon_{it}) Q_{it}^{\beta_1} PF_{it}^{\beta_2}$$

The following statements create the data and perform the necessary log transformations. These statements assume that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.airline;
  input Obs Airline T C Q PF LF;
  Year = T + 1969;
  lC   = log(C);
  lQ   = log(Q);
  lPF  = log(PF);
  label lC   = "Log Transformation of Costs";
  label lQ   = "Log Transformation of Quantity";
  label lPF  = "Log Transformation of Price of Fuel";
  label LF   = "Load Factor (utilization index)";
datalines;
1   1   1   1140640   0.95276   106650   0.53449
2   1   2   1215690   0.98676   110307   0.53233
3   1   3   1309570   1.09198   110574   0.54774
4   1   4   1511530   1.17578   121974   0.54085
5   1   5   1676730   1.16017   196606   0.59117
... more lines ...
```

The following statements fit a one-way fixed-effects model:

```
proc cpanel data = mylib.airline;
  id Airline Year;
  model lC = lQ lPF LF / fixone printfixed;
run;
```

Output 11.1.1 provides a model and data description. There are six cross sections and 15 time points.

Output 11.1.1 Airline Cost Data, Model Description

The CPANEL Procedure
One-Way Fixed Effects (FixOne)
Dependent Variable: IC (Log Transformation of Costs)

Model Description	
Estimation Method	FixOne
Variance Estimation	Model Based
Data Set	AIRLINE
Number of Observations	90
Number of Cross Sections	6
Time Series Length	15

The R-square and degrees of freedom can be seen in [Output 11.1.2](#). The R-square statistic nearly 1, indicating a reasonable fit. The error degrees of freedom is derived from 90 observations minus 5 cross sections, minus 4 regressors.

Output 11.1.2 Airline Cost Data, Fit Statistics

Fit Statistics			
SSE	0.2926	DFE	81
MSE	0.0036	Root MSE	0.0601
R-Square	0.9974		

The F test for fixed effects is shown in [Output 11.1.3](#). You easily reject the null hypothesis of poolability. There are significant effects due to airlines, and it would be unreasonable to perform pooled OLS regression that ignores these effects.

Output 11.1.3 Airline Cost Data, Test for Fixed Effects

F Test for No Fixed Effects				
Num DF	Den DF	F Value	Pr > F	
5	81	57.74	<.0001	

The PRINTFIXED option in the MODEL statement provides estimates of the airline effects (which are not displayed by default). Looking at the parameters, only two airlines (3 and 4) are significantly different from average. Quantity and fuel price have positive effects on cost, but load factors negatively affect costs. Because cost, quantity, and fuel price are log-transformed, the coefficients for quantity and price are interpreted as elasticities of cost. The coefficient for (log) fuel price is 0.417, meaning that you would associate a 10% increase in fuel price with a 4.17% increase in costs.

Output 11.1.4 Airline Cost Data, Parameter Estimates

Parameter Estimates						
Variable	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	1	9.713556	0.2296	42.30	<.0001	Intercept
IQ	1	0.919293	0.0299	30.76	<.0001	Log Transformation of Quantity
IPF	1	0.417492	0.0152	27.47	<.0001	Log Transformation of Price of Fuel
LF	1	-1.07044	0.2017	-5.31	<.0001	Load Factor (utilization index)
CS 1	1	-0.00759	0.0465	-0.16	0.8707	Cross-Sectional Effect: 1
CS 2	1	-0.04883	0.0390	-1.25	0.2144	Cross-Sectional Effect: 2
CS 3	1	-0.21651	0.0184	-11.77	<.0001	Cross-Sectional Effect: 3
CS 4	1	0.176971	0.0214	8.27	<.0001	Cross-Sectional Effect: 4
CS 5	1	0.016475	0.0378	0.44	0.6638	Cross-Sectional Effect: 5
CS 6	1	0.079484	0.0415	1.92	0.0588	Cross-Sectional Effect: 6

You suspect that there might be other factors at play, and so you augment your model to include time effects. The following statements fit a two-way model, a model with both airline and time effects:

```
proc cpanel data = mylib.airline;
  id Airline Year;
  model lC = lQ lPF LF / fixtwo printfixed;
run;
```

The F test and parameter estimates for the two-way model are provided in [Output 11.1.5](#).

Output 11.1.5 Airline Cost Data, Two-Way Fixed Effects

The CPANEL Procedure
Two-Way Fixed Effects (FixTwo)
Dependent Variable: IC (Log Transformation of Costs)

F Test for No Fixed Effects				
Num DF	Den DF	F Value	Pr > F	
19	67	23.10	<.0001	

Output 11.1.5 *continued*

Parameter Estimates						
Variable	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	1	12.66527	2.0810	6.09	<.0001	Intercept
IQ	1	0.817264	0.0318	25.66	<.0001	Log Transformation of Quantity
IPF	1	0.168732	0.1635	1.03	0.3057	Log Transformation of Price of Fuel
LF	1	-0.88267	0.2617	-3.37	0.0012	Load Factor (utilization index)
CS 1	1	0.128306	0.0460	2.79	0.0069	Cross-Sectional Effect: 1
CS 2	1	0.065481	0.0390	1.68	0.0975	Cross-Sectional Effect: 2
CS 3	1	-0.18948	0.0156	-12.14	<.0001	Cross-Sectional Effect: 3
CS 4	1	0.134259	0.0183	7.33	<.0001	Cross-Sectional Effect: 4
CS 5	1	-0.09264	0.0373	-2.48	0.0155	Cross-Sectional Effect: 5
CS 6	1	-0.04593	0.0416	-1.10	0.2736	Cross-Sectional Effect: 6
Time 1970	1	-0.37386	0.1919	-1.95	0.0555	Time Effect: 1970
Time 1971	1	-0.31916	0.1861	-1.72	0.0909	Time Effect: 1971
Time 1972	1	-0.27654	0.1833	-1.51	0.1362	Time Effect: 1972
Time 1973	1	-0.22292	0.1730	-1.29	0.2019	Time Effect: 1973
Time 1974	1	-0.15388	0.0864	-1.78	0.0796	Time Effect: 1974
Time 1975	1	-0.10805	0.0449	-2.41	0.0188	Time Effect: 1975
Time 1976	1	-0.07686	0.0319	-2.41	0.0188	Time Effect: 1976
Time 1977	1	-0.02072	0.0204	-1.01	0.3145	Time Effect: 1977
Time 1978	1	0.047197	0.0291	1.62	0.1093	Time Effect: 1978
Time 1979	1	0.091657	0.0811	1.13	0.2627	Time Effect: 1979
Time 1980	1	0.207197	0.1491	1.39	0.1693	Time Effect: 1980
Time 1981	1	0.28534	0.1756	1.62	0.1089	Time Effect: 1981
Time 1982	1	0.301255	0.1660	1.81	0.0741	Time Effect: 1982
Time 1983	1	0.300347	0.1536	1.96	0.0547	Time Effect: 1983
Time 1984	1	0.319001	0.1475	2.16	0.0341	Time Effect: 1984

Only a few of the time effects are significant, but there is an overall time trend of increasing costs. The time period of the data spans the OPEC oil embargoes and the dissolution of the Civil Aeronautics Board (CAB). These are two possible explanations for the rising costs.

A surprising result is that the fuel cost is not significant in the two-way model. If the time effects are proxies for the effect of the oil embargoes, then the effect of fuel price might be subsumed by the time effects. If the time dummy variables are proxies for the dissolution of the CAB, then the effect of load factors is not precisely estimated.

Example 11.2: Analyzing Demand for Liquid Assets: Random Effects

Feige (1964) provides data on the demand for liquid assets. The data are for six states and the District of Columbia (CA, DC, FL, IL, NY, TX, and WA) and were collected each year from 1949 to 1959. All variables are log-transformed.

The following statements create the `Assets` data table. These statements assume that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
data mylib.Assets;
  length state $ 2;
  input state $ year d t s y rd rt rs;
  label d = 'Per Capita Demand Deposits'
        t = 'Per Capita Time Deposits'
        s = 'Per Capita S & L Association Shares'
        y = 'Permanent Per Capita Personal Income'
        rd = 'Service Charge on Demand Deposits'
        rt = 'Interest on Time Deposits'
        rs = 'Interest on S & L Association Shares';
datalines;
CA  1949  6.2785  6.1924  4.4998  7.2056 -1.0700  0.1080  1.0664
CA  1950  6.4019  6.2106  4.6821  7.2889 -1.0106  0.1501  1.0767
CA  1951  6.5058  6.2729  4.8598  7.3827 -1.0024  0.4008  1.1291
CA  1952  6.4785  6.2729  5.0039  7.4000 -0.9970  0.4492  1.1227
CA  1953  6.4118  6.2538  5.1761  7.4200 -0.8916  0.4662  1.2110
CA  1954  6.4520  6.2971  5.3613  7.4478 -0.6951  0.4756  1.1924

... more lines ...
```

The data contain per capita consumptions for three liquid assets: demand deposits such as checking, time deposits, and savings and loan (S&L) shares. You posit a linear model for per capita demand deposits, with random effects for states.

The following statements fit a one-way random-effects model:

```
proc cpanel data = mylib.Assets;
  id state year;
  model d = y rd rt rs / ranone;
run;
```

The regression results are provided in [Output 11.2.1](#).

The “Variance Component Estimates” table provides the estimated variances and standard deviations for the cross-sectional (state) effects in addition to the overall errors. A majority of the overall error variance can be attributed to differences between states, not differences within states.

The “Hausman Test for Random Effects” table shows the result of a Hausman specification test. The null hypothesis is that state effects can be treated as random (random-effects model) and that they do not need to be estimated directly (fixed-effects model). The test results favor the random-effects specification that is used to generate this output.

Output 11.2.1 Demand for Demand Deposits, One-Way Random-Effects Model

The CPANEL Procedure
One-Way Random Effects (RanOne)
Swamy and Arora Variance Components
Dependent Variable: d (Per Capita Demand Deposits)

Model Description					
Estimation Method	RanOne				
Variance Estimation	Model Based				
Data Set	ASSETS				
Number of Observations	77				
Number of Cross Sections	7				
Time Series Length	11				

Fit Statistics			
SSE	0.0939	DFE	72
MSE	0.0013	Root MSE	0.0361
R-Square	0.7651		

Variance Component Estimates		
Source	Variance	Standard Deviation
Cross Sections	0.046394	0.215393
Error	0.00134	0.036608

Hausman Test For Random Effects				
Coefficients	DF	m	Value	Pr > m
	4	4	1.52	0.8235

Parameter Estimates					
Variable	DF	Estimate	Standard Error	t Value	Pr > t
Intercept	1	-1.64325	0.6841	-2.40	0.0189
y	1	1.136776	0.0998	11.39	<.0001
rd	1	-0.26428	0.0517	-5.11	<.0001
rt	1	0.032076	0.0291	1.10	0.2746
rs	1	-0.41213	0.1192	-3.46	0.0009

The parameter estimate for the variable Y is greater than 1, indicating that demand is elastic to income—income has a more than proportional positive association with the demand for demand deposits. The coefficient on the variable RD indicates that demand deposits increase significantly as the service charge is reduced.

The variables RT and RS represent positive aspects of competing products, and you would expect these variables to affect demand negatively. The coefficient for RS meets that expectation, but the coefficient for RT is not significant.

The previous analysis used the default Swamy-Arora method to estimate the variance components. The CPANEL procedure supports four other methods, and you might be interested in how the different methods affect the analysis.

The following statements fit the model by using all five methods and include a COMPARE statement to compare the results:

```
proc cpanel data = mylib.Assets;
  id state year;
  sa: model d = y rd rt rs / ranone vcomp = sa;
  wh: model d = y rd rt rs / ranone vcomp = wh;
  wk: model d = y rd rt rs / ranone vcomp = wk;
  fb: model d = y rd rt rs / ranone vcomp = fb;
  nl: model d = y rd rt rs / ranone vcomp = nl;
  compare / mstat(varcs varerr);
run;
```

The tables produced by the COMPARE statement are provided in [Output 11.2.2](#).

Output 11.2.2 Comparison of Variance-Component Methods, Assets Data

**The CPANEL Procedure
Model Comparison**

Model: NL

Dependent Variable: d (Per Capita Demand Deposits)

Comparison of Model Statistics					
Statistic	SA RanOne	WH RanOne	WK RanOne	FB RanOne	NL RanOne
Var due to Cross Sections	0.0464	0.0315	0.0315	0.0291	0.0327
Var due to Error	0.001340	0.000107	0.001340	0.001340	0.001149

Comparison of Model Parameter Estimates					
Variable	SA RanOne	WH RanOne	WK RanOne	FB RanOne	NL RanOne
Intercept	Estimate -1.643246	-1.472425	-1.723092	-1.742581	-1.680406
	Std Err 0.684144	0.719067	0.681184	0.680541	0.682676
y	Estimate 1.136776	1.117252	1.145844	1.148051	1.141001
	Std Err 0.099816	0.099799	0.099776	0.099761	0.099802
rd	Estimate -0.264276	-0.245861	-0.272995	-0.275135	-0.268325
	Std Err 0.051729	0.052260	0.051445	0.051372	0.051600
rt	Estimate 0.032076	0.029227	0.033397	0.033718	0.032692
	Std Err 0.029138	0.028570	0.029416	0.029485	0.029266
rs	Estimate -0.412126	-0.414540	-0.410731	-0.410361	-0.411500
	Std Err 0.119181	0.117486	0.119968	0.120160	0.119548

You conclude that how you estimate variance components has little bearing on the regression results.

Example 11.3: Panel Study of Income Dynamics (PSID): Hausman-Taylor Models

Cornwell and Rupert (1988) analyze data from the Panel Study of Income Dynamics (PSID), an income study of 595 individuals over the seven-year period, 1976–1982 inclusive. Of particular interest is the effect of additional schooling on wages. The analysis here replicates that of Baltagi (2013, sec. 7.5), where it is surmised that covariate correlation with individual effects makes a standard random-effects model inadequate.

The following statements create the PSID data. These statements assume that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.psid;
  input id t lwage wks south smsa ms exp exp2 occ ind union fem blk ed;
  label id      = 'Person ID'
        t       = 'Time'
        lwage   = 'Log of wages'
        wks     = 'Weeks worked'
        south   = '1 if resides in the South'
        smsa    = '1 if resides in SMSA'
        ms      = '1 if married'
        exp     = 'Years full-time experience'
        exp2    = 'exp squared'
        occ     = '1 if blue-collar occupation'
        ind     = '1 if manufacturing'
        union   = '1 if union contract'
        fem     = '1 if female'
        blk     = '1 if black'
        ed      = 'Years of education';
datalines;
1   1  5.5606799126  32  1  0  1  3   9   0  0  0  0  0  0  9
1   2  5.7203102112  43  1  0  1  4  16   0  0  0  0  0  0  9
1   3  5.9964499474  40  1  0  1  5  25   0  0  0  0  0  0  9
1   4  5.9964499474  39  1  0  1  6  36   0  0  0  0  0  0  9
1   5  6.0614600182  42  1  0  1  7  49   0  1  0  0  0  0  9
1   6  6.1737899780  35  1  0  1  8  64   0  1  0  0  0  0  9
1   7  6.2441701889  32  1  0  1  9  81   0  1  0  0  0  0  9
2   1  6.1633100510  34  0  0  1  30  900  1  0  0  0  0  0  11
2   2  6.2146100998  27  0  0  1  31  961  1  0  0  0  0  0  11
2   3  6.2634000778  33  0  0  1  32 1024  1  1  1  0  0  0  11
2   4  6.5439100266  30  0  0  1  33 1089  1  1  0  0  0  0  11
2   5  6.6970300674  30  0  0  1  34 1156  1  1  0  0  0  0  11
2   6  6.7912201881  37  0  0  1  35 1225  1  1  0  0  0  0  11
2   7  6.8156399727  30  0  0  1  36 1296  1  1  0  0  0  0  11

... more lines ...
```

You begin by fitting a one-way random-effects model:

```
proc cpanel data = mylib.psid;
  id id t;
  model lwage = wks south smsa ms exp exp2 occ
              ind union fem blk ed / ranone;
run;
```

The output is shown in [Output 11.3.1](#). The coefficient on the variable ED (which represents years of education) estimates that an additional year of schooling is associated with about a 10% increase in wages. However, the results of the Hausman test for random effects show a serious violation of the random-effects assumptions, namely that the regressors are independent of the individual effects.

Output 11.3.1 One-Way Random Effects Estimation

The CPANEL Procedure
One-Way Random Effects (RanOne)
Swamy and Arora Variance Components
Dependent Variable: lwage (Log of wages)

Model Description						
Estimation Method	RanOne					
Variance Estimation	Model Based					
Data Set	PSID					
Number of Observations	4165					
Number of Cross Sections	595					
Time Series Length	7					

Variance Component Estimates			
Source	Variance	Standard Deviation	
Cross Sections	0.068972	0.262626	
Error	0.023102	0.151994	

Hausman Test For Random Effects				
Coefficients	DF	m Value	Pr > m	
9	9	5074.87	<.0001	

Parameter Estimates						
Variable	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	1	4.26382	0.0977	43.64	<.0001	Intercept
wks	1	0.001035	0.000773	1.34	0.1810	Weeks worked
south	1	-0.01662	0.0265	-0.63	0.5309	1 if resides in the South
smsa	1	-0.01381	0.0200	-0.69	0.4897	1 if resides in SMSA
ms	1	-0.07463	0.0230	-3.24	0.0012	1 if married
exp	1	0.082051	0.00285	28.81	<.0001	Years full-time experience
exp2	1	-0.00081	0.000063	-12.87	<.0001	exp squared
occ	1	-0.05007	0.0166	-3.01	0.0026	1 if blue-collar occupation
ind	1	0.003745	0.0173	0.22	0.8283	1 if manufacturing
union	1	0.063226	0.0171	3.70	0.0002	1 if union contract
fem	1	-0.33923	0.0513	-6.61	<.0001	1 if female
blk	1	-0.21027	0.0580	-3.63	0.0003	1 if black
ed	1	0.099653	0.00575	17.34	<.0001	Years of education

An alternative could be a fixed-effects (FIXONE) model, but that model would not permit estimation of the coefficient for ED, which does not vary within individuals. A compromise is the Hausman-Taylor model, for which you stipulate a set of covariates that are correlated with the individual effects (but uncorrelated with

the observation-level errors). You specify the correlated variables in the CORRELATED statement:

```
proc cpanel data = mylib.psid;
  id id t;
  model lwage = wks south smsa ms exp exp2 occ
              ind union fem blk ed / htaylor;
  correlated wks ms exp exp2 union ed;
run;
```

The results are shown in [Output 11.3.2](#). The table of parameter estimates has an added column, Type, which identifies the regressors that are assumed to be correlated with individual effects (C) and the regressors that do not vary within cross sections (TI). It was stated previously that the Hausman-Taylor model is a compromise between fixed-effects and random-effects models, and you can think of the compromise this way: You want to fit a random-effects model, but the correlated (C) variables make that model invalid. So you fall back to the consistent fixed-effects model, but then the time-invariant (TI) variables are the problem because they would be dropped from that model. The solution is to use the Hausman-Taylor estimator.

The estimation results show that an additional year of schooling is now associated with a 13.8% increase in wages. Also presented is a Hausman test that compares this model to the fixed-effects model. As was the case previously when you fit the random-effects model, you can think of the Hausman test as a referendum on the assumptions you are making. For this estimation, it seems that your choice of variables to treat as correlated is adequate.

Output 11.3.2 Hausman-Taylor Estimation

The CPANEL Procedure Hausman and Taylor Model for Correlated Individual Effects Dependent Variable: lwage (Log of wages)

Variance Component Estimates			
Source	Variance	Standard Deviation	
Cross Sections	0.886993	0.941803	
Error	0.023044	0.151803	

Hausman Test vs. Fixed Effects				
Coefficients	DF	m Value	Pr > m	
9	3	5.26	0.1539	

Output 11.3.2 *continued*

Parameter Estimates							
Variable	Type	DF	Estimate	Standard	t Value	Pr > t	Label
				Error			
Intercept	TI	1	2.912726	0.2837	10.27	<.0001	Intercept
wks	C	1	0.000837	0.000600	1.40	0.1627	Weeks worked
south		1	0.00744	0.0320	0.23	0.8159	1 if resides in the South
smsa		1	-0.04183	0.0190	-2.21	0.0274	1 if resides in SMSA
ms	C	1	-0.02985	0.0190	-1.57	0.1159	1 if married
exp	C	1	0.113133	0.00247	45.79	<.0001	Years full-time experience
exp2	C	1	-0.00042	0.000055	-7.67	<.0001	exp squared
occ		1	-0.0207	0.0138	-1.50	0.1331	1 if blue-collar occupation
ind		1	0.013604	0.0152	0.89	0.3720	1 if manufacturing
union	C	1	0.032771	0.0149	2.20	0.0280	1 if union contract
fem	TI	1	-0.13092	0.1267	-1.03	0.3014	1 if female
blk	TI	1	-0.28575	0.1557	-1.84	0.0665	1 if black
ed	C TI	1	0.137944	0.0212	6.49	<.0001	Years of education

C: correlated with the individual effects

TI: constant (time-invariant) within cross sections

At its core, the Hausman-Taylor estimator is an instrumental variables regression, where the instruments are derived from regressors that are assumed to be uncorrelated with the individual effects. Technically, it is the cross-sectional means of these variables that need to be uncorrelated, not the variables themselves.

The Amemiya-MaCurdy model is a close relative of the Hausman-Taylor model. The only difference between the two is that the Amemiya-MaCurdy model makes the added assumption that the regressors (and not just their means) are uncorrelated with the individual effects. By making that assumption, the Amemiya-MaCurdy model can take advantage of a more efficient set of instrumental variables.

The following statements fit the Amemiya-MaCurdy model:

```
proc cpanel data = mylib.psid;
  id id t;
  model lwage = wks south smsa ms exp exp2 occ
               ind union fem blk ed / amacurdy;
  correlated wks ms exp exp2 union ed;
run;
```

The results are shown in [Output 11.3.3](#). Little is changed from the Hausman-Taylor model. The Hausman test compares the Amemiya-MaCurdy model to the Hausman-Taylor model (not the fixed-effects model as previously) and shows that the one additional assumption is acceptable. You even gained a bit of efficiency in the process; compare the standard deviations of the coefficient on the variable ED from both models.

Output 11.3.3 Amemiya-MaCurdy Estimation

The CPANEL Procedure
Amemiya and MaCurdy Model for Correlated Individual Effects
Dependent Variable: lwage (Log of wages)

Variance Component Estimates							
Source			Variance	Standard			
				Deviation			
Cross Sections			0.886993	0.941803			
Error			0.023044	0.151803			

Hausman Test vs. Hausman-Taylor				
Coefficients	DF	m	Value	Pr > m
13	13		14.67	0.3287

Parameter Estimates							
Variable	Type	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	TI	1	2.927338	0.2751	10.64	<.0001	Intercept
wks	C	1	0.000838	0.000599	1.40	0.1622	Weeks worked
south		1	0.007282	0.0319	0.23	0.8197	1 if resides in the South
smsa		1	-0.04195	0.0189	-2.21	0.0269	1 if resides in SMSA
ms	C	1	-0.03009	0.0190	-1.59	0.1127	1 if married
exp	C	1	0.11297	0.00247	45.76	<.0001	Years full-time experience
exp2	C	1	-0.00042	0.000055	-7.72	<.0001	exp squared
occ		1	-0.02085	0.0138	-1.51	0.1299	1 if blue-collar occupation
ind		1	0.013629	0.0152	0.89	0.3709	1 if manufacturing
union	C	1	0.032475	0.0149	2.18	0.0293	1 if union contract
fem	TI	1	-0.13201	0.1266	-1.04	0.2972	1 if female
blk	TI	1	-0.2859	0.1555	-1.84	0.0660	1 if black
ed	C TI	1	0.137205	0.0206	6.67	<.0001	Years of education

C: correlated with the individual effects							
TI: constant (time-invariant) within cross sections							

Finally, you should realize that the Hausman-Taylor and Amemiya-MaCurdy estimators are not cure-alls for correlated individual effects. Estimation tacitly relies on the uncorrelated regressors being sufficient to predict the correlated regressors. Otherwise, you run into the problem of weak instruments. If you have weak instruments, you will obtain biased estimates that have very large standard errors. However, that does not seem to be the case here.

Example 11.4: Crime Rates in North Carolina: Instrumental Variables Regression

Cornwell and Trumbull (1994) examined data on crime rates from 90 counties in North Carolina and collected data each year from 1981 to 1987. The following statements create the Crime data table. These statements assume that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.Crime;
  input county year west central urban d82 d83 d84 d85 d86 d87
        lcrmrte lprbarr lprbconv lprbpris lavgsen lpolpc
        ldensity ltaxpc lwcon lwtuc lwtrd lwfir lwser lwmfg
        lwfed lwsta lwloc lmix lpctymle lpctmin;
  label county      = 'county identifier'
        year        = '81 to 87'
        west        = '1 if in western N.C.'
        central     = '1 if in central N.C.'
        urban       = '1 if in SMSA'

  ... more lines ...
```

The variables are too numerous to describe individually, but they are fully labeled in the regression output and are summarized as follows:

- The panel ID variables are County and Year.
- The dependent variable, LCrMrtE, is the log of the crime rate (number of crimes divided by total population).
- The variables West, Central, and Urban provide geographical background for each county.
- The variables D82 through D87 are dummy variables for year effects.
- The variables with the LPrb prefix are log-probabilities of police or legal actions. For example, LPrbArr is the log-probability of an arrest given that a crime was committed.
- The variables with the LW prefix are log weekly wages for various occupations and industries.
- Other variables that begin with the letter L are log-transformed. For example, the variable LPolPC is the log of police per capita.

Cornwell and Trumbull (1994) fit a fixed-effects model for the log of crime rate, with the log-probability of arrest (LPrbArr), and the log of police per capita (LPolPC) among the regressors. They noted that these two regressors were endogenous because of the following causality loops:

- Although police presence might decrease the crime rate because of deterrence (or increase the crime rate because of better reporting), an increase in crime might cause a desire for more police presence.
- Although the crime rate might fall if more crimes resulted in arrest, a rising crime rate could cause the probability of arrest to decrease because police resources would be spread thin.

To deal with the endogeneity, Cornwell and Trumbull (1994) fit a two-stage least squares (2SLS) model with county-level fixed effects, with two external instruments: LTaxPC (log tax revenue per capita) and LMix (log ratio of crimes that involve face-to-face contact). Their rationale for using these instruments is that counties with higher tax revenues have more police presence and that face-to-face crimes have a higher probability of arrest because victims could identify their offenders.

Baltagi (2006) proposed a similar 2SLS model, but with random effects instead of fixed effects for counties. The following statements recreate that estimation:

```
proc cpanel data = mylib.crime;
  id county year;
  model lcrmte = lprbarr lpolpc lprbconv lprbpris lavgsen ldensity
               lwcon lwtuc lwtrd lwfir lwser lwmfg lwfed lwsta
               lwloc lpctymle lpctmin west central urban
               d82 d83 d84 d85 d86 d87 / ivranone;
  endogenous lprbarr lpolpc;
  instruments ltaxpc lmix;
run;
```

The results are shown in [Output 11.4.1](#). The output includes a “Structural Summary” table, which lists the endogenous variables and the instrumental variables. The list of instrumental variables includes both the specified external instruments (variables LTaxPC and LMix) and the exogenous regression variables.

Output 11.4.1 Crime Rates in North Carolina, IV Random Effects

The CPANEL Procedure EC2SLS One-Way Random Effects (IVRanOne) Dependent Variable: lcrmte (log crimes committed per person)

Model Description			
Estimation Method	IVRanOne		
Variance Estimation	Model Based		
Data Set	CRIME		
Number of Observations	630		
Number of Cross Sections	90		
Time Series Length	7		

Fit Statistics			
SSE	12.1250	DFE	603
MSE	0.0201	Root MSE	0.1418

Variance Component Estimates		
Source	Variance	Standard Deviation
Cross Sections	0.046036	0.21456
Error	0.022272	0.149239

Hausman Test For Random Effects			
Coefficients	DF	m Value	Pr > m
22	22	19.50	0.6140

Output 11.4.1 *continued*

Structural Summary							
Endogenous Variables	lprbarr lpolpc						
Instruments	lprbconv lprbpris lavgsen ldensity lwcon lwtuc lwtrd lwfir lwser lwmgf lwfed lwsta lwloc lpctymle lpctmin west central urban d82 d83 d84 d85 d86 d87 ltaxpc lmix						
Parameter Estimates							
Variable	Type	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept		1	-0.9538	1.2840	-0.74	0.4579	Intercept
lprbarr	EN	1	-0.41293	0.0974	-4.24	<.0001	log probability of arrest
lpolpc	EN	1	0.434749	0.0897	4.85	<.0001	log police per capita
lprbconv		1	-0.32289	0.0536	-6.03	<.0001	log probability of conviction
lprbpris		1	-0.18632	0.0419	-4.44	<.0001	log probability of prison sentence
lavgsen		1	-0.01018	0.0270	-0.38	0.7066	log avg. sentence in days
ldensity		1	0.429028	0.0548	7.82	<.0001	log people per sq. mile
lwcon		1	-0.00748	0.0396	-0.19	0.8503	log weekly wage, construction
lwtuc		1	0.045445	0.0198	2.30	0.0220	log wkly wge, trns, util, commun
lwtrd		1	-0.00814	0.0414	-0.20	0.8441	log wkly wge, whlesle, retail trade
lwfir		1	-0.00364	0.0289	-0.13	0.8999	log wkly wge, fin, ins, real est
lwser		1	0.00561	0.0201	0.28	0.7805	log wkly wge, service industry
lwmgf		1	-0.20414	0.0804	-2.54	0.0114	log wkly wge, manufacturing
lwfed		1	-0.16351	0.1594	-1.03	0.3056	log wkly wge, fed employees
lwsta		1	-0.05405	0.1057	-0.51	0.6092	log wkly wge, state employees
lwloc		1	0.163052	0.1196	1.36	0.1734	log wkly wge, local gov emps
lpctymle		1	-0.10811	0.1397	-0.77	0.4393	log percent young male
lpctmin		1	0.189037	0.0415	4.56	<.0001	log perc. minority, 1980
west		1	-0.22684	0.0996	-2.28	0.0231	1 if in western N.C.
central		1	-0.19404	0.0598	-3.24	0.0012	1 if in central N.C.
urban		1	-0.22515	0.1156	-1.95	0.0520	1 if in SMSA
d82		1	0.010745	0.0258	0.42	0.6772	1 if year == 82
d83		1	-0.08379	0.0307	-2.73	0.0065	1 if year == 83
d84		1	-0.1035	0.0371	-2.79	0.0054	1 if year == 84
d85		1	-0.0957	0.0495	-1.94	0.0534	1 if year == 85
d86		1	-0.0689	0.0596	-1.16	0.2481	1 if year == 86
d87		1	-0.03141	0.0705	-0.45	0.6562	1 if year == 87

EN: endogenous variable

Cornwell and Trumbull (1994) favored a fixed-effects approach because a Hausman specification test rejected the null hypothesis of random effects. Baltagi (2006) argued that the Hausman test used by Cornwell and Trumbull (1994) was inaccurate because it was based on standard methodology that did not account for endogeneity in the regressors.

Baltagi (2006) proposed a modified 2SLS Hausman test, and the results of that test are provided in the “Hausman Test For Random Effects” table in the output. The results favor the random-effects specification.

A higher probability of arrest is shown to diminish crime, and police presence increases the crime rate because more crimes are reported.

The following statements fit the model by two-step GMM with robust standard errors:

```
proc cpanel data = mylib.crime;
  id county year;
  model lcrmte = lprbarr lpolpc lprbconv lprbpris lavgsen ldensity
              lwcon lwtuc lwtrd lwfir lwser lwmfg lwfed lwsta
              lwloc lpctymle lpctmin west central urban
              d82 d83 d84 d85 d86 d87 / ivranone gmm = twostep
              robust;
  endogenous lprbarr lpolpc;
  instruments ltaxpc lmix;
run;
```

The results are shown in [Output 11.4.2](#). The coefficients do not change much, but the standard errors are now more reliable. You can also obtain bias-corrected standard errors by using the `BIASCORRECTED` option according to the method of Windmeijer (2005).

Output 11.4.2 Crime Rates in North Carolina, IV Random Effects by Two-Step GMM

The CPANEL Procedure
EC2SLS One-Way Random Effects (IVRanOne)
Dependent Variable: lcrmte (log crimes committed per person)

Model Description	
Estimation Method	IVRanOne
Estimation Technique	Two-Step GMM
Variance Estimation	Robust
Data Set	CRIME
Number of Observations	630
Number of Cross Sections	90
Time Series Length	7

Fit Statistics			
SSE	12.3814	DFE	603
MSE	0.0205	Root MSE	0.1433

Variance Component Estimates		
Source	Variance	Standard Deviation
Cross Sections	0.046036	0.21456
Error	0.022272	0.149239

Hausman Test For Random Effects			
Coefficients	DF	m Value	Pr > m
22	22	15.77	0.8271

Structural Summary	
Endogenous Variables	lprbarr lpolpc
Instruments	lprbconv lprbpris lavgsen ldensity lwcon lwtuc lwtrd lwfir lwser lwmfg lwfed lwsta lwloc lpctymle lpctmin west central urban d82 d83 d84 d85 d86 d87 ltaxpc lmix

Output 11.4.2 continued

Parameter Estimates							
Variable	Type	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept		1	-1.66797	1.1204	-1.49	0.1371	Intercept
lprbarr	EN	1	-0.37858	0.1137	-3.33	0.0009	log probability of arrest
lpolpc	EN	1	0.426573	0.0858	4.97	<.0001	log police per capita
lprbconv		1	-0.31245	0.0587	-5.32	<.0001	log probability of conviction
lprbpris		1	-0.14817	0.0458	-3.23	0.0013	log probability of prison sentence
lavgsen		1	0.030359	0.0309	0.98	0.3255	log avg. sentence in days
ldensity		1	0.340629	0.0585	5.83	<.0001	log people per sq. mile
lwcon		1	-0.02737	0.0246	-1.11	0.2658	log weekly wage, construction
lwtuc		1	0.029904	0.0195	1.54	0.1248	log wkly wge, trns, util, commun
lwtrd		1	0.003937	0.0422	0.09	0.9258	log wkly wge, whlesle, retail trade
lwfir		1	-0.00248	0.0131	-0.19	0.8498	log wkly wge, fin, ins, real est
lwser		1	-0.00434	0.0130	-0.33	0.7379	log wkly wge, service industry
lwmfg		1	-0.16174	0.0835	-1.94	0.0532	log wkly wge, manufacturing
lwfed		1	-0.00709	0.1499	-0.05	0.9623	log wkly wge, fed employees
lwsta		1	-0.02507	0.0798	-0.31	0.7535	log wkly wge, state employees
lwloc		1	0.090092	0.1089	0.83	0.4086	log wkly wge, local gov emps
lpctymle		1	-0.07707	0.1143	-0.67	0.5002	log percent young male
lpctmin		1	0.181526	0.0478	3.79	0.0002	log perc. minority, 1980
west		1	-0.2151	0.1031	-2.09	0.0374	1 if in western N.C.
central		1	-0.16708	0.0537	-3.11	0.0020	1 if in central N.C.
urban		1	-0.12948	0.0816	-1.59	0.1129	1 if in SMSA
d82		1	0.01654	0.0242	0.68	0.4939	1 if year == 82
d83		1	-0.07028	0.0313	-2.25	0.0251	1 if year == 83
d84		1	-0.09766	0.0339	-2.88	0.0041	1 if year == 84
d85		1	-0.08787	0.0462	-1.90	0.0578	1 if year == 85
d86		1	-0.0586	0.0508	-1.15	0.2492	1 if year == 86
d87		1	-0.02047	0.0580	-0.35	0.7244	1 if year == 87

EN: endogenous variable

Example 11.5: Cigarette Sales Data: Dynamic Panel Estimation

Consider a dynamic panel demand model for cigarette sales that illustrates the methods described in the section “Dynamic Panel Estimation (DYNDIFF and DYNSSYS Options)” on page 511. The data are from a panel of 46 American states over the period 1963–1992. The dependent variable is the logarithm of per capita cigarette sales (variable LSales). Other factors that were measured include the log of price (LPrice), the log of disposable income (LDisp), and the log of minimum price in adjoining states (LMin). For a full description of the data, see Baltagi (2013, sec. 8.9).

The following statements create the Cigar data set. These statements assume that your libref is named mylib, but you can substitute any appropriately defined libref.


```

data mylib.Cigar;
  input State Year Price Pop Pop_16 Cpi Disp Sales Min;
  LSales = log(Sales);
  LPrice = log(Price);
  LDisp  = log(Disp);
  LMin   = log(Min);
  label
    State   = 'State abbreviation'
    Year    = 'Year'
    LSales  = 'Log cigarette sales in packs per capita'
    LPrice  = 'Log price per pack of cigarettes'
    LDisp   = 'Log per capita disposable income'
    LMin    = 'Log minimum price in adjoining states per pack of cigarettes';
datalines;
1 63 28.6 3383 2236.5 30.6 1558.3045298 93.9 26.1
1 64 29.8 3431 2276.7 31.0 1684.0732025 95.4 27.5
1 65 29.8 3486 2327.5 31.5 1809.8418752 98.5 28.9
1 66 31.5 3524 2369.7 32.4 1915.1603572 96.4 29.5
1 67 31.6 3533 2393.7 33.4 2023.5463678 95.5 29.6
1 68 35.6 3522 2405.2 34.8 2202.4855362 88.4 32
1 69 36.6 3531 2411.9 36.7 2377.3346665 90.1 32.8
1 70 39.6 3444 2394.6 38.8 2591.0391591 89.8 34.3
1 71 42.7 3481 2443.5 40.5 2785.3159706 95.4 35.8
... more lines ...

```

You propose a panel model for cigarette sales that contains fixed effects for states. Because you believe that the data are insufficient to explain all possible shocks in yearly sales, you include lagged sales in the model as a regressor. By construction, lagged sales are an endogenous regressor, and you thus specify dynamic panel estimation by using the DYNDIFF option. The following statements fit the model:

```

proc cpanel data = mylib.Cigar;
  id State Year;
  model LSales = LPrice LDisp LMin / dyndiff;
run;

```

The results are shown in [Output 11.5.1](#). Note that it was not necessary to explicitly include lagged sales on the right-hand side of the model; PROC CPANEL generates it for you. The coefficient on lagged sales is 0.732, indicating a high degree of autocorrelation in the dependent variable. When cigarette sales are unusually high or low because of unforeseen circumstances, the effects tend to linger for several years. The results also show that demand is highly elastic to price.

Output 11.5.1 Dynamic Panel Estimation for Cigarette Sales

The CPANEL Procedure
Dynamic Panel Estimation Using Difference Equations (DynDiff)
Dependent Variable: LSales (Log cigarette sales in packs per capita)

Model Description						
Estimation Method	DynDiff					
Estimation Technique	One-Step GMM					
Variance Estimation	Model Based					
Data Set	CIGAR					
Number of Observations	1380					
Number of Cross Sections	46					
Time Series Length	30					
GMM Bandwidth	30					

Fit Statistics			
SSE	3.1373	DFE	1283
MSE	0.0024	Root MSE	0.0494

Instruments			
Type	Variables	Number of Instruments	
Difference Equations, Endogenous	LSales	406	
Difference Equations, Standard	LPrice LDisp LMin	3	
Level Equations, Standard	Intercept	1	
Total		410	

Sargan Test			
DF	Chi-Square	Pr > ChiSq	
405	712.45	<.0001	

Parameter Estimates						
Variable	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	1	0.769092	0.0658	11.69	<.0001	Intercept
LSales_1	1	0.732212	0.0178	41.07	<.0001	Lag 1: Log cigarette sales in packs per capita
LPrice	1	-0.26328	0.0255	-10.31	<.0001	Log price per pack of cigarettes
LDisp	1	0.166116	0.0105	15.88	<.0001	Log per capita disposable income
LMin	1	0.032726	0.0233	1.40	0.1604	Log minimum price in adjoining states per pack of cigarettes

AR(m) Tests		
Lag	Z	Pr > Z
1	-15.44	<.0001
2	2.47	0.0134

Included in [Output 11.5.1](#) are two diagnostic measures. The first, a Sargan test, is a test of the validity of the moment conditions that are conferred by the GMM instruments that were used. The p -value indicates that the moment conditions are not valid and that you should probably look for a set of instruments other than the default set that PROC CPANEL provides.

The second diagnostic test is the $AR(m)$ test for autocorrelation in the residuals. In well-fitting dynamic panel models, you expect to see some autocorrelation of lag 1, but any autocorrelation at higher lags indicates a poor fit. The autocorrelation at lag 2 is significant, leading you to seek a better-fitting alternative.

One possible explanation for the poor fit is that, by default, PROC CPANEL uses the one-step generalized method of moments (GMM). One-step GMM is known for being too reliant on the assumption that the residuals from the difference equations are not serially correlated. An alternative is two-step GMM, which instead uses a data-driven variance matrix for the differenced residuals.

The following statements fit the model by two-step GMM:

```
proc cpanel data = mylib.Cigar;
  id State Year;
  endogenous LSales;
  instruments LPrice LDisp LMin;
  model LSales = LPrice LDisp LMin / dyndiff gmm = twostep biascorrected;
run;
```

The code includes ENDOGENOUS and INSTRUMENTS statements that, for demonstration purposes, reproduce the default instrument set. That set includes the following:

- GMM-style instruments based on the endogenous dependent variable, LSales
- standard instruments for the exogenous regressors LPrice, LDisp, and LMin
- a column of ones in the instrument matrix for the level equations that corresponds to the model intercept

The code also includes the BIASCORRECTED option, which produces bias-corrected standard errors according to the method of Windmeijer (2005).

The results are shown in [Output 11.5.2](#). The coefficients do not change much, but the standard errors are now more reliable. The model diagnostic tests indicate a better fit, although you should use caution when interpreting Sargan test results. Sargan tests lack power when the number of instruments is large, and their distributional properties come into question under conditions that favor either robust or bias-corrected standard errors.

Output 11.5.2 Dynamic Panel Estimation by Two-Step GMM

The CPANEL Procedure Dynamic Panel Estimation Using Difference Equations (DynDiff) Dependent Variable: LSales (Log cigarette sales in packs per capita)

Model Description	
Estimation Method	DynDiff
Estimation Technique	Two-Step GMM
Variance Estimation	Windmeijer Bias Corrected
Data Set	CIGAR
Number of Observations	1380
Number of Cross Sections	46
Time Series Length	30
GMM Bandwidth	30

Output 11.5.2 *continued*

Fit Statistics					
SSE	3.1277	DFE	1283		
MSE	0.0024	Root MSE	0.0494		

Instruments		
Type	Variables	Number of Instruments
Difference Equations, Endogenous	LSales	406
Difference Equations, Standard	LPrice LDisp LMin	3
Level Equations, Standard	Intercept	1
Total		410

Sargan Test		
DF	Chi-Square	Pr > ChiSq
41	45.20	0.3008

Parameter Estimates					
Variable	DF	Estimate	Standard Error	t Value	Pr > t Label
Intercept	1	0.776914	0.1458	5.33	<.0001 Intercept
LSales_1	1	0.728347	0.0492	14.79	<.0001 Lag 1: Log cigarette sales in packs per capita
LPrice	1	-0.25707	0.0421	-6.11	<.0001 Log price per pack of cigarettes
LDisp	1	0.167345	0.0248	6.75	<.0001 Log per capita disposable income
LMin	1	0.025064	0.0432	0.58	0.5616 Log minimum price in adjoining states per pack of cigarettes

AR(m) Tests		
Lag	Z	Pr > Z
1	-5.00	<.0001
2	1.90	0.0575

The previous estimation treats regressors such as LPrice as exogenous. If you believe that price is endogenous, you can create GMM-style instruments for LPrice to replace the default standard instruments.

The following statements fit the model by using GMM-style instruments for LPrice:

```
proc cpanel data = mylib.Cigar;
  id State Year;
  endogenous LSales LPrice;
  instruments LDisp LMin;
  model LSales = LPrice LDisp LMin / dyndiff gmm = twostep biascorrected;
run;
```

The results are shown in [Output 11.5.3](#). Treating LPrice as endogenous nearly doubles the number of instruments. Although this is not the case here, when the number of instruments is so large that it makes estimation infeasible, you can limit the number of instruments by specifying the MAXBAND= option in the MODEL statement.

Output 11.5.3 Dynamic Panel Estimation, Custom Instrument Set

The CPANEL Procedure
Dynamic Panel Estimation Using Difference Equations (DynDiff)
Dependent Variable: LSales (Log cigarette sales in packs per capita)

Model Description	
Estimation Method	DynDiff
Estimation Technique	Two-Step GMM
Variance Estimation	Windmeijer Bias Corrected
Data Set	CIGAR
Number of Observations	1380
Number of Cross Sections	46
Time Series Length	30
GMM Bandwidth	30

Fit Statistics			
SSE	3.3942	DFE	1283
MSE	0.0026	Root MSE	0.0514

Instruments		
Type	Variables	Number of Instruments
Difference Equations, Endogenous	LSales LPrice	812
Difference Equations, Standard	LDisp LMin	2
Level Equations, Standard	Intercept	1
Total		815

Sargan Test		
DF	Chi-Square	Pr > ChiSq
41	44.98	0.3088

Parameter Estimates						
Variable	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	1	0.531106	0.1271	4.18	<.0001	Intercept
LSales_1	1	0.799028	0.0412	19.38	<.0001	Lag 1: Log cigarette sales in packs per capita
LPrice	1	-0.22491	0.0346	-6.50	<.0001	Log price per pack of cigarettes
LDisp	1	0.139713	0.0202	6.92	<.0001	Log per capita disposable income
LMin	1	0.029462	0.0383	0.77	0.4416	Log minimum price in adjoining states per pack of cigarettes

AR(m) Tests		
Lag	Z	Pr > Z
1	-5.02	<.0001
2	1.94	0.0521

References

- Amemiya, T. (1971). "The Estimation of the Variances in a Variance-Component Model." *International Economic Review* 12:1–13.
- Amemiya, T., and MaCurdy, T. E. (1986). "Instrumental-Variable Estimation of an Error-Components Model." *Econometrica* 54:869–880.
- Andrews, D. W. K. (1991). "Heteroscedasticity and Autocorrelation Consistent Covariance Matrix Estimation." *Econometrica* 59:817–858.
- Arellano, M., and Bond, S. (1991). "Some Tests of Specification for Panel Data: Monte Carlo Evidence and an Application to Employment Equations." *Review of Economic Studies* 58:277–297.
- Baltagi, B. H. (1981). "Simultaneous Equations with Error Components." *Journal of Econometrics* 17:189–200.
- Baltagi, B. H. (2006). "Estimating an Economic Model of Crime Using Panel Data from North Carolina." *Journal of Applied Econometrics* 21:543–547.
- Baltagi, B. H. (2013). *Econometric Analysis of Panel Data*. 5th ed. Chichester, UK: John Wiley & Sons.
- Baltagi, B. H., and Chang, Y. (1994). "Incomplete Panels: A Comparative Study of Alternative Estimators for the Unbalanced One-Way Error Component Regression Model." *Journal of Econometrics* 62:67–89.
- Baltagi, B. H., Song, S. H., and Jung, B. C. (2002). "A Comparative Study of Alternative Estimators for the Unbalanced Two-Way Error Component Regression Model." *Econometrics Journal* 5:480–493.
- Blundell, R., and Bond, S. (1998). "Initial Conditions and Moment Restrictions in Dynamic Panel Data Models." *Journal of Econometrics* 87:115–143.
- Buse, A. (1973). "Goodness of Fit in Generalized Least Squares Estimation." *American Statistician* 27:106–108.
- Cornwell, C., and Rupert, P. (1988). "Efficient Estimation with Panel Data: An Empirical Comparison of Instrumental Variables Estimators." *Journal of Applied Econometrics* 3:149–155.
- Cornwell, C., and Trumbull, W. N. (1994). "Estimating the Economic Problem of Crime with Panel Data." *Review of Econometrics and Statistics* 76:360–366.
- Da Silva, J. G. C. (1975). "The Analysis of Cross-Sectional Time Series Data." Ph.D. diss., Department of Statistics, North Carolina State University.
- Davidson, R., and MacKinnon, J. G. (1993). *Estimation and Inference in Econometrics*. New York: Oxford University Press.
- Feige, E. L. (1964). *The Demand for Liquid Assets: A Temporal Cross-Section Analysis*. Englewood Cliffs, NJ: Prentice-Hall.
- Fuller, W. A., and Battese, G. E. (1974). "Estimation of Linear Models with Crossed-Error Structure." *Journal of Econometrics* 2:67–78.

- Gardner, R. (1998). "Unobservable Individual Effects in Unbalanced Panel Data." *Economics Letters* 58:39–42.
- Greene, W. H. (1990). *Econometric Analysis*. New York: Macmillan.
- Greene, W. H. (2000). *Econometric Analysis*. 4th ed. Upper Saddle River, NJ: Prentice-Hall.
- Hausman, J. A. (1978). "Specification Tests in Econometrics." *Econometrica* 46:1251–1271.
- Hausman, J. A., and Taylor, W. E. (1981). "Panel Data and Unobservable Individual Effects." *Econometrica* 49:1377–1398.
- Hausman, J. A., and Taylor, W. E. (1982). "A Generalized Specification Test." *Economics Letters* 8:239–245.
- Nerlove, M. (1971). "Further Evidence on the Estimation of Dynamic Relations from a Time Series of Cross Sections." *Econometrica* 39:359–382.
- Newey, W. K., and West, D. W. (1994). "Automatic Lag Selection in Covariance Matrix Estimation." *Review of Economic Studies* 61:631–653.
- Nickell, S. J. (1981). "Biases in Dynamic Models with Fixed Effects." *Econometrica* 49:1417–1426.
- Parks, R. W. (1967). "Efficient Estimation of a System of Regression Equations When Disturbances Are Both Serially and Contemporaneously Correlated." *Journal of the American Statistical Association* 62:500–509.
- Swamy, P. A. V. B., and Arora, S. S. (1972). "The Exact Finite Sample Properties of the Estimators of Coefficients in the Error Components Regression Models." *Econometrica* 40:261–275.
- Theil, H. (1961). *Economic Forecasts and Policy*. 2nd ed. Amsterdam: North-Holland.
- Wallace, T., and Hussain, A. (1969). "The Use of Error Components Model in Combining Cross Section with Time Series Data." *Econometrica* 37:55–72.
- Wansbeek, T., and Kapteyn, A. (1989). "Estimation of the Error-Components Model with Incomplete Panels." *Journal of Econometrics* 41:341–361.
- White, H. (1980). "A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity." *Econometrica* 48:817–838.
- Windmeijer, F. (2005). "A Finite Sample Correction for the Variance of Linear Efficient Two-Step GMM Estimators." *Journal of Econometrics* 126:25–51.
- Wooldridge, J. M. (2002). *Econometric Analysis of Cross Section and Panel Data*. Cambridge, MA: MIT Press.
- Wu, D. M. (1973). "Alternative Tests of Independence between Stochastic Regressors and Disturbances." *Econometrica* 41:733–750.

Chapter 12

CQLIM Procedure

Contents

Overview: CQLIM Procedure	560
PROC CQLIM Features	561
PROC CQLIM Compared with Other SAS Procedures	562
Using CAS Sessions and CAS Engine Librefs	562
Getting Started: CQLIM Procedure	563
Syntax: CQLIM Procedure	565
Functional Summary	565
PROC CQLIM Statement	568
BAYES Statement	570
BOUNDS Statement	579
BY Statement	580
CLASS Statement	580
DISPLAY Statement	580
DISPLAYOUT Statement	581
ENDOGENOUS Statement	582
FREQ Statement	584
HETERO Statement	584
INIT Statement	585
MODEL Statement	586
OUTPUT Statement	588
PRIOR Statement	589
RESTRICT Statement	596
TEST Statement	597
WEIGHT Statement	598
Details: CQLIM Procedure	599
Ordinal Discrete Choice Modeling	599
Limited Dependent Variable Models	600
Stochastic Frontier Production and Cost Models	600
Heteroscedasticity	602
Bivariate Censored Dependent Variable Modeling	603
Multivariate Limited Dependent Models	604
Tests on Parameters	604
Bayesian Analysis	606
Prior Distributions	608
Output to SAS Data Table	610
Naming	614

ODS Table Names	615
Examples: The CQLIM Procedure	617
Example 12.1: Model with Censoring	617
Example 12.2: Bivariate Probit Analysis	618
Example 12.3: Bayesian Analysis	620
Example 12.4: Marginal Effects Using Bayesian Analysis with NUTS	623
Example 12.5: Bayesian Analysis with Multiple Chains	627
References	632

Overview: CQLIM Procedure

The CQLIM procedure analyzes univariate and multivariate limited dependent variable models in which dependent variables take discrete values or are observed within only a limited range of values. When the dependent variable takes discrete values, you can use either a logit model, a probit model, or an ordered logit or probit model, depending on the distribution of the dependent variable. In addition to the discrete variable models, the CQLIM procedure includes Tobit models, truncated regression models, and multivariate models. The multivariate model can contain discrete choice and limited dependent variables in addition to linear continuous dependent variables.

Another class of limited dependent variable models that the CQLIM procedure supports is stochastic frontier models. These models were first introduced by Aigner, Lovell, and Schmidt (1977) and Meeusen and van den Broeck (1977); they are capable of analyzing technical inefficiencies in production functions or cost functions.

The CQLIM procedure is similar to the QLIM procedure in SAS/ETS software; PROC CQLIM requires SAS Cloud Analytic Services (CAS) in order to run. Both procedures analyze univariate and multivariate limited dependent variable models. Unlike the QLIM procedure, which can run only on a single workstation, the CQLIM procedure takes advantage of the distributed computing environment of CAS, which enables it to distribute the optimization task to one or more nodes. In addition, each node can use one or more threads to perform the optimization on its subset of the data. When several nodes are used and each node uses several threads to carry out its part of the work, the result is a highly parallel computation that can provide a dramatic gain in performance.

The CQLIM procedure can use the maximum likelihood method or the Bayesian method to perform the inference of the available models. By default, it uses multiple threads to perform computations.

PROC CQLIM Features

The CQLIM procedure supports the following models:

- linear regression models with heteroscedasticity
- probit models with heteroscedasticity
- logit models with heteroscedasticity
- Tobit models with heteroscedasticity
- truncated regression models with heteroscedasticity
- bivariate probit models
- bivariate Tobit models
- multivariate limited dependent variables
- stochastic frontier production and cost models

In linear regression models with heteroscedasticity, the assumption that error variance is constant across observations is relaxed. The CQLIM procedure allows for a number of different linear and nonlinear variance specifications.

The CQLIM procedure is also capable of modeling problems in which the dependent variable takes discrete values. For these problems, you can use either a logit model or a probit model. In both cases, the dependent variable is modeled by using a binary distribution (typically, $y = 0, 1$). The main difference is that the logit model is characterized by an underlying logistic distribution, whereas the probit model has a Gaussian distribution. When the dependent variable takes more than two levels, the binary assumption is no longer valid. In these cases, you can use the ordered logit regression and the ordered probit regression. Assumptions about the error variance can also be relaxed in order to estimate a probit or logit model with heteroscedasticity.

PROC CQLIM also offers a class of models in which the dependent variable is censored or truncated from below, above, or both. When a continuous dependent variable is observed within only a certain range, and values outside this range are not available, PROC CQLIM offers a class of models that adjust for truncation. In some cases, the dependent variable is continuous within only a certain range, and all values outside this range are reported as being on its boundary. For example, if it is not possible to observe negative values, the value of the dependent variable is reported as equal to 0. Because the data are censored, ordinary least squares (OLS) results are inconsistent, and you cannot be certain that the predicted values from the model will fall in the appropriate region.

Most of the models in the CQLIM procedure can be extended to accommodate bivariate and multivariate scenarios.

The stochastic frontier production and cost models available in PROC CQLIM allow for random shocks of the production or cost. These models include a systematic positive component in the error term that adjusts for technical or cost inefficiency.

The CQLIM procedure can use the maximum likelihood method for both univariate and multivariate models. Initial starting values for the nonlinear optimizations are usually calculated by OLS. In addition, the CQLIM procedure can use the Bayesian method for univariate models, with the following exceptions: stochastic frontier models, models with heterogeneous variances, and ordinal models.

PROC CQLIM Compared with Other SAS Procedures

The CQLIM procedure provides limited dependent data modeling functionality comparable to that of the HPQLIM and QLIM procedures in SAS/ETS software.

PROC CQLIM Compared with the HPQLIM Procedure

The functionality of the CQLIM procedure closely resembles that of the HPQLIM procedure, which is a high-performance procedure. The CQLIM procedure is the next generation of the HPQLIM procedure, and it was developed specifically for SAS Viya. Both procedures are designed to run on a cluster of machines that distribute the data and the computations. Both procedures perform computations in multiple threads on multiple machines.

The CQLIM procedure supports maximum likelihood estimation of all the models available in PROC HPQLIM. It also supports Bayesian estimation of most single-equation models.

PROC CQLIM Compared with the QLIM Procedure

The CQLIM procedure is specifically designed to operate in SAS Viya and performs computations in multiple threads on multiple machines. The capability of the QLIM procedure to perform computations in a distributed environment is limited to the parallel implementation of Markov chain Monte Carlo sampling algorithms for Bayesian analysis.

This release of the CQLIM procedure contains some, but not all, of the functionality available in the QLIM procedure. PROC QLIM provides the following features, which are not available in PROC CQLIM:

- sample selection models
- variable selection
- Bayesian analysis for all models

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: CQLIM Procedure

This example illustrates the use of the CQLIM procedure. The data were originally published by Mroz (1987), and the following DATA steps load a subset of the data. The assumption here is that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
title1 'Estimating a Tobit Model';

data subset;
  input Hours Yrs_Ed Yrs_Exp @@;
datalines;
0 8 9 0 8 12 0 9 10 0 10 15 0 11 4 0 11 6
1000 12 1 1960 12 29 0 13 3 2100 13 36
3686 14 11 1920 14 38 0 15 14 1728 16 3
1568 16 19 1316 17 7 0 17 15
;

data mylib.subset;
  set subset;
run;
```

In these data, `Hours` is the number of hours that a wife worked outside the household in a particular year, `Yrs_Ed` is years of education, and `Yrs_Exp` is years of work experience.

From the nature of the data, it is clear that there are a number of women who committed some positive number of hours to outside work ($y_i > 0$ is observed). There are also a number of women who did not work outside the household at all ($y_i = 0$ is observed). This yields the following model,

$$y_i^* = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

$$y_i = \begin{cases} y_i^* & \text{if } y_i^* > 0 \\ 0 & \text{if } y_i^* \leq 0 \end{cases}$$

where $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$ and $\mathbf{x}_i$ denotes the set of explanatory variables. The following statements fit a Tobit model to the number of hours worked, with years of education and years of work experience as covariates:

```

/*-- Tobit Model --*/
proc cqlim data=mylib.subset;
  model hours = yrs_ed yrs_exp;
  endogenous hours ~ censored(lb=0);
run;

```

The output of the CQLIM procedure is shown in [Output 12.1](#).

Figure 12.1 Tobit Analysis Results

Estimating a Tobit Model

The CQLIM Procedure

Model Fit Summary	
Dependent Variable	Hours
Number of Observations	17
Data Set	SUBSET
Log Likelihood	-74.937
Maximum Absolute Gradient	6.748E-6
Number of Iterations	5
Optimization Method	Newton-Raphson
AIC	157.874
SBC	161.2069
Covariance Estimation	Hessian

Convergence criterion (ABSGCONV=0.00001) satisfied.

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	-5595.665232	27.645559	-202.41	<.0001
Yrs_Ed	1	372.981321	53.962540	6.91	<.0001
Yrs_Exp	1	63.319498	36.536554	1.73	0.0831
_Sigma	1	1582.240492	389.785175	4.06	<.0001

The “Parameter Estimates” table contains four rows. The first three rows correspond to the vector estimate of the regression coefficients β . The last row is called `_Sigma`, which corresponds to the estimate of the error variance σ .

Syntax: CQLIM Procedure

The following statements are available in the CQLIM procedure:

```

PROC CQLIM options ;
  BAYES <options> ;
  BOUNDS bound1 < , bound2 ... > ;
  BY variables ;
  CLASS variable <( options )> ... <variable <( options )>> </ global-options > ;
  DISPLAY <table-list> </ options > ;
  DISPLAYOUT table-spec-list </ options > ;
  FREQ variable ;
  ENDOGENOUS variables ~ options ;
  HETERO dependent-variables ~ exogenous-variables / options ;
  INIT initvalue1 < , initvalue2 ... > ;
  MODEL dependent-variables = regressors / options ;
  OUTPUT OUT=libref.data-table <output-options> ;
  PRIOR parameter-list ~ distribution ;
  RESTRICT restriction1 < , restriction2 ... > ;
  TEST options ;
  WEIGHT variable </ option > ;

```

At least one MODEL statement is required. If you specify more than one MODEL statement, the CQLIM procedure estimates a system of models. The BAYES statement is not compatible with multiple MODEL statements or with the BOUNDS, HETERO, RESTRICT, or TEST statement. If you specify more than one FREQ or WEIGHT statement, only the variable that is specified in the first instance of the statement is used.

Functional Summary

Table 12.1 summarizes the statements and options available in the CQLIM procedure.

Table 12.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CQLIM	DATA=
Writes predictions to an output data table	OUTPUT	OUT=
Declaring the Role of Variables		
Specifies BY-group processing	BY	
Specifies classification variables	CLASS	
Specifies a frequency variable	FREQ	
Specifies a weight variable	WEIGHT	NONNORMALIZE

Table 12.1 *continued*

Description	Statement	Option
Printing Control Options		
Prints the correlation matrix of the estimates	PROC CQLIM	CORRB
Prints the covariance matrix of the estimates	PROC CQLIM	COVB
Prints a summary iteration listing	PROC CQLIM	ITPRINT
Optimization Process Control Options		
Specifies the maximum number of iterations	PROC CQLIM	MAXITER=
Selects the iterative minimization method to use	PROC CQLIM	METHOD=
Sets boundary restrictions on parameters	BOUNDS	
Sets initial values for parameters	INIT	
Sets linear restrictions on parameters	RESTRICT	
Model Estimation Options		
Specifies the method to calculate parameter covariance	PROC CQLIM	COVEST=
Specifies the number of draws for Monte Carlo integration	PROC CQLIM	NDRAW=
Suppresses the intercept parameter	MODEL	NOINT
Specifies a seed for pseudorandom number generation	PROC CQLIM	SEED=
Bayesian Markov Chain Monte Carlo (MCMC) Options		
Specifies the number of burn-in iterations	BAYES	NBURNIN=
Specifies the number of Markov chains to run during the sampling phase	BAYES	NCHAIN=
Specifies the number of iterations to run during the sampling phase	BAYES	NSAMPLE=
Specifies the MCMC sampler and its options	BAYES	SAMPLER=
Specifies the random number generator seed	BAYES	SEED=
Controls the thinning of the Markov chain	BAYES	THIN=
Bayesian Output Tables		
Displays MCMC diagnostics	BAYES	DIAGNOSTICS=
Displays MCMC diagnostics for each Markov chain	BAYES	DIAGNOSTICS(BYCHAIN)=
Displays the settings used for all MCMC diagnostics	BAYES	DIAGNOSTICS(SETTINGS)=
Displays a summary of all parameters and their priors	BAYES	PRIORSUMMARY
Displays the settings used for the MCMC sampler	BAYES	SAMPLERSETTINGS
Displays a summary of the MCMC sampler used	BAYES	SAMPLERSUMMARY
Displays posterior summary statistics	BAYES	STATISTICS=
Displays posterior summary statistics for each Markov chain	BAYES	STATISTICS(BYCHAIN)=

Table 12.1 *continued*

Description	Statement	Option
Bayesian Posterior Sample		
Specifies options for saving a SAS data table for the posterior sample	BAYES	OUTPOST=
Bayesian Prior Options		
Specifies the Cauchy prior distribution	PRIOR	CAUCHY()
Specifies the gamma prior distribution	PRIOR	GAMMA()
Specifies the inverse gamma prior distribution	PRIOR	IGAMMA()
Specifies the normal prior distribution	PRIOR	NORMAL()
Specifies the square root gamma prior distribution	PRIOR	SQRTGAMMA()
Specifies the square root inverse gamma prior distribution	PRIOR	SQRTIGAMMA()
Specifies the <i>t</i> prior distribution	PRIOR	T()
Specifies the uniform prior distribution	PRIOR	UNIFORM()
Endogenous Variable Options		
Specifies a discrete variable	ENDOGENOUS	DISCRETE()
Specifies a censored variable	ENDOGENOUS	CENSORED()
Specifies a truncated variable	ENDOGENOUS	TRUNCATED()
Specifies a stochastic frontier variable	ENDOGENOUS	FRONTIER()
Heteroscedasticity Model Options		
Specifies the function for heteroscedasticity models	HETERO	LINK=
Squares the function for heteroscedasticity models	HETERO	SQUARE
Specifies no constant for heteroscedasticity models	HETERO	NOCONST
Output Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	
Outputs the predicted values	OUTPUT	PREDICTED
Outputs the structured part	OUTPUT	XBETA
Outputs the residuals	OUTPUT	RESIDUAL
Outputs the error standard deviation	OUTPUT	ERRSTD
Outputs the marginal effects	OUTPUT	MARGINAL
Outputs the probability for the current response	OUTPUT	PROB
Outputs the probability for all responses	OUTPUT	PROBALL
Outputs the expected value	OUTPUT	EXPECTED
Outputs the conditional expected value	OUTPUT	CONDITIONAL
Outputs the inverse Mills ratio	OUTPUT	MILLS
Outputs the technical efficiency measures	OUTPUT	TE1
	OUTPUT	TE2

Table 12.1 *continued*

Description	Statement	Option
Test Request Options		
Requests Wald, Lagrange multiplier, and likelihood ratio tests	TEST	ALL
Requests the Wald test	TEST	WALD
Requests the Lagrange multiplier test	TEST	LM
Requests the likelihood ratio test	TEST	LR

PROC CQLIM Statement

PROC CQLIM < *options* > ;

The PROC CQLIM statement invokes the CQLIM procedure. You can specify the following *options*.

Data Table Options

DATA=*libref.data-table*

names the input data table for PROC CQLIM to use. The default is the most recently created data table. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 562.

data-table specifies the name of the input data table.

Printing Options

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

ITPRINT

prints the initial parameter estimates, convergence criteria, and all constraints of the optimization. At each iteration, the objective function value, step size, maximum gradient, and slope of search direction are also printed.

Model Estimation Options

COVEST=HESSIAN | OP | QML

specifies the method to use for calculating the covariance matrix of parameter estimates. You can specify the following covariance options:

HESSIAN	specifies the covariance from the inverse Hessian matrix.
OP	specifies the covariance from the outer product matrix.
QML	specifies the covariance from the outer product and Hessian matrices (the quasi-maximum likelihood estimates).

By default, COVEST=HESSIAN.

NDRAW=*value*

specifies the number of draws for Monte Carlo integration.

SEED=*value*

specifies a seed for pseudorandom number generation in Monte Carlo integration.

Optimization Control Options

PROC CQLIM uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. You can specify the following *options*:

MAXITER=*i*

MAXIT=*i*

specifies the maximum number of iterations in the optimization process. By default, MAXITER=200.

METHOD=CONGRA | DBLDOG | NEWRAP | NONE | NRRIDG | QUANEW | TRUREG

specifies the iterative minimization method to use. You can specify the following methods:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double-dogleg method.
NEWRAP	specifies the Newton-Raphson method.
NONE	specifies that no optimization be performed beyond using the ordinary least squares method to compute the parameter estimates.
NRRIDG	specifies the Newton-Raphson ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

By default, METHOD=NEWRAP.

BAYES Statement

BAYES < options > ;

The BAYES statement controls the Markov chain Monte Carlo sampling scheme that is used to obtain a sample from the posterior distribution, as well as the Bayesian output tables, including diagnostics and statistics. The BAYES statement is incompatible with multiple MODEL statements; with the BOUNDS, HETERO, RESTRICT, and TEST statements; with stochastic frontier models; and with ordinal models. You can specify the following *options*:

DIAGNOSTICS < (diag-options) >=ALL | NONE | keyword-list | (keyword-list)

DIAG < (diag-options) >=ALL | NONE | keyword-list | (keyword-list)

specifies which diagnostics to produce. If the value of the NCHAIN= option is greater than 1, then instead of the diagnostic tables being produced for each Markov chain, by default the results of the diagnostics for all the chains are appropriately combined and summarized. For more information, see the section “[Using Multiple Chains](#)” on page 37 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or both of the following *diag-options*:

BYCHAIN

produces diagnostic tables for each Markov chain. If NCHAIN=1, then the BYCHAIN option does nothing.

SETTINGS

produces a table of all MCMC diagnostic settings.

If you specify both the BYCHAIN and SETTINGS options, separate them by using a space or a comma. To produce all diagnostics, specify DIAGNOSTICS=ALL. To produce no diagnostics, specify DIAGNOSTICS=NONE. To produce some but not all diagnostics, or to change certain settings of these diagnostics, specify one or more of the following keywords. If you specify multiple keywords, separate them by using spaces.

AUTOCORR < (LAGS=numeric-list) >

AC < (LAGS=numeric-list) >

computes the autocorrelations at lags that you specify in the *numeric-list*. For more information, see the section “[Autocorrelations](#)” on page 34 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” Elements in the *numeric-list* must be positive integers, and values can be separated either by spaces or by commas. Repeated values are removed. If you omit the LAGS= option, autocorrelations of lags 1, 5, 10, and 50 are computed.

GEWEKE < (geweke-options) >

GEW < (geweke-options) >

computes the Geweke spectral density diagnostics. For more information, see the section “[Geweke Diagnostics](#)” on page 28 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *geweke-options*:

ALPHA=*value*

specifies the α level ($0 < \alpha < 1$) for the Geweke test. By default, ALPHA=0.05.

FRAC1=*value*

F1=*value*

specifies the fraction f_1 for the first window. By default, FRAC1=0.1.

FRAC2=*value*

F2=*value*

specifies the fraction f_2 for the second window. By default, FRAC2=0.5.

HEIDELBERGER <(heidel-options)>

HEIDEL <(heidel-options)>

HB <(heidel-options)>

computes the Heidelberg-Welch diagnostic for each variable. For more information, see the section “[Heidelberg-Welch Diagnostics](#)” on page 30 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or more of the following *heidel-options*:

EPSILON=*value*

EPS=*value*

specifies a positive number ϵ such that if the halfwidth is less than ϵ times the sample mean of the retained iterates, the halfwidth test is passed. By default, EPSILON=0.1.

HALFWIDTH_ALPHA=*value*

HALPHA=*value*

specifies the α level ($0 < \alpha < 1$) for the halfwidth test. By default, HALFWIDTH_ALPHA=0.05.

STATIONARY_ALPHA=*value*

SALPHA=*value*

specifies the α level ($0 < \alpha < 1$) for the stationarity test. By default, STATIONARY_ALPHA=0.05.

RAFTERY <(raftery-options)>

RL <(raftery-options)>

computes the Raftery-Lewis diagnostics. For more information, see the section “[Raftery-Lewis Diagnostics](#)” on page 31 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or more of the following *raftery-options*:

ACCURACY=*value*

ACC=*value*

R=*value*

specifies a small positive number r as the margin of error for measuring the accuracy of estimation of the quantile. By default, ACCURACY=0.005.

EPSILON=*value*

EPS=*value*

specifies the tolerance level (ϵ , a small positive number) for the stationary test. By default, EPSILON=0.001.

PROBABILITY=*value*

PROB=*value*

S=*value*

specifies the probability s of attaining the accuracy of the estimation of the quantile. By default, PROBABILITY=0.95.

QUANTILE=*value*

QUANT=*value*

Q=*value*

specifies the order (q , a value between 0 and 1) of the quantile of interest. By default, QUANTILE=0.025.

SUMMARIES <(MAXLAG=*value*)>

SUMMARY <(MAXLAG=*value*)>

SUM <(MAXLAG=*value*)>

computes a summary of MCMC diagnostics for each parameter in the Markov chain, including effective sample size (ESS), Monte Carlo standard error (MCSE), and related diagnostics. For more information about ESS, see the section “[Effective Sample Size](#)” on page 34 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#)”; for more information about MCSE, see the section “[Standard Error of the Mean Estimate](#)” on page 36 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” MAXLAG=*value* specifies the maximum autocorrelation lag to use for computing ESS, which is used to compute MCSE, where *value* must be a positive integer. By default, MAXLAG=250.

By default, DIAGNOSTICS=SUMMARIES.

DISTRIBUTED

DISTRIBUTE

keeps the data in the input table distributed on the CAS server when the server runs in massively parallel processing (MPP) mode. If you omit this option, instead the data are copied onto each worker for Bayesian inference, and multiple Markov chains are run in parallel. This option is ignored when the CAS server runs in symmetric multiprocessing (SMP) mode. For more information, see the section “[Distributed Data and MCMC](#)” on page 607.

NBURNIN=*number*

BURNIN=*number*

NBI=*number*

specifies the number of burn-in iterations to perform in each Markov chain before the chains are saved. For more information, see the section “[Burn-In, Thinning, and Markov Chain Samples](#)” on page 27 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, NBURNIN=0.

NCHAIN=*number*

NCHAINS=*number*

CHAINS=*number*

specifies the number of Markov chains to run. By default, NCHAIN=1.

NSAMPLE=*number*

NSAM=*number*

NMC=*number*

specifies the number of iterations after tuning and burn-in to save in each Markov chain. For more information, see the section “[Burn-In, Thinning, and Markov Chain Samples](#)” on page 27 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, NSAMPLE=1000.

OUTPOST<(*outpost-option*)>=*libref.data-table*

names the SAS data table to contain the posterior sample. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 562. You can specify the following *outpost-option*:

DETAILED

saves any relevant additional variables, such as auxiliary variables, in the SAS data table. The random walk Metropolis (RWM) sampler without the [TRANSFORM](#) option has no additional variables to save. With the TRANSFORM option, the SAS data table also includes the transformed version of each parameter and the log posterior of the transformed parameters. When you use the No-U-Turn Sampler (NUTS) by specifying the [SAMPLER=NUTS](#) option, the SAS data table includes those additional variables, as well as the tree height that is achieved at every iteration.

WARMUP

saves warmup (tuning and burn-in) iterations in the SAS data table. This option is not available for the random walk Metropolis (RWM) sampler.

PRIORSUMMARY <(*priorsum-option*)>

PRIORSUM <(*priorsum-option*)>

produces a summary of all parameters in the model and their prior distributions. You can specify the following *priorsum-option*:

SHOWNAMES

NAMES

adds a column that displays the names of all parameters in the model. These are the names that you use to specify prior distributions on those parameters.

SAMPLER=NUTS <(*nuts-options*)> | **RWM** <(*rwm-options*)>

SAMPLING=NUTS <(*nuts-options*)> | **RWM** <(*rwm-options*)>

specifies how to sample from the posterior distribution. By default, SAMPLER=RWM.

NUTS <(nuts-options)>

samples from the posterior distribution by using the No-U-Turn Sampler. You can specify the following *nuts-options*:

DELTA=*number*

TARGETRATE=*number*

RATE=*number*

specifies the target Metropolis acceptance rate δ for the No-U-Turn Sampler. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. By default, DELTA=0.8.

MASSTUNE=NONE | WINDOWED <(windowed-options)>

specifies the mass matrix tuning method for the No-U-Turn Sampler. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. By default, MASSTUNE=WINDOWED. You can specify the following values:

NONE

specifies no mass matrix tuning. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607.

WINDOWED <(windowed-options)>**WINDOW** <(windowed-options)>**WIN** <(windowed-options)>

specifies the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. You can specify the following *windowed-options*:

EXPANDRATE=*number*

EXPRATE=*number*

RATE=*number*

specifies the window expansion rate γ for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. By default, EXPANDRATE=2.

FINALWINDOW=*number*

FINAL=*number*

specifies the size of the final tuning window n_K for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. By default, FINALWINDOW=50.

INITWINDOW=*number*

INIT=*number*

specifies the size of the initial tuning window n_1 for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. By default, INITWINDOW=75.

MIDWINDOW=number

MID=number

specifies the size of the first middle tuning window n_2 for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. By default, MIDWINDOW=25.

OVERLAPPING

OVERLAP

OVER

uses overlapping windows for the windowed mass matrix tuning method. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607.

MAXHEIGHT=number

MAXDEPTH=number

MAXTREE=number

specifies the maximum height of the binary tree $J_{\max}$ for the No-U-Turn Sampler. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. To see how many times the algorithm reaches the maximum tree height during sampling, use the [SAMPLERSUMMARY](#) option. By default, MAXHEIGHT=10.

MAXHEIGHTKILL= k | INFINITY

MAXDEPTHKILL= k | INFINITY

MAXTREEKILL= k | INFINITY

specifies the maximum number of times the No-U-Turn Sampler can reach the maximum binary tree height before the simulation stops. To see how many times the algorithm reaches the maximum tree height during sampling, use the [SAMPLERSUMMARY](#) option. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. You can specify the following values:

k

specifies the maximum number of times. If $k=0$, then there is no maximum.

INFINITY

INFTY

INF

specifies no maximum number of times.

By default, MAXHEIGHTKILL=0.

NTUNE=number

NTU=number

specifies the number of iterations n_{tune} for the No-U-Turn Sampler. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607. By default, NTUNE=2000.

RWM <(rwm-options)>

samples from the posterior distribution by using the random walk Metropolis scheme. You can specify the following *rwm-options*:

ACCTOL=*value*

ATOL=*value*

specifies the tolerance ϵ for the acceptance rate during the tuning process of the random walk Metropolis (RWM) sampler. For more information, see the section “[Scale Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, ACCTOL=0.075.

INITSCALE=*value*

SCALE=*value*

specifies the initial scale parameter(s) c^{cur} of the RWM proposal distribution. For more information, see the section “[Scale Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, INITSCALE=2.38.

NTUNE=*number*

NTU=*number*

specifies the number of iterations n_{tune} for each tuning stage of the random walk Metropolis sampler. For more information, see the section “[Tuning the Random Walk Metropolis Sampler](#)” on page 606. By default, NTUNE=500.

NTUNESTAGE=*number*

specifies the number of tuning stages n_{stage} . For more information, see the section “[Tuning the Random Walk Metropolis Sampler](#)” on page 606. By default, NTUNESTAGE=24.

TARGETRATE=*value*

RATE=*value*

DELTA=*value*

specifies the target acceptance rate δ of the random walk Metropolis sampler. For more information, see the section “[Tuning Random Walk Metropolis Algorithms](#)” on page 19 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, TARGETRATE=0.234.

TRANSFORM

TRANS

transforms all parameters to an unconstrained space and implements the random walk Metropolis sampler on the unconstrained space. For more information, see the section “[Parameter Transformation](#)” on page 25 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).”

TUNEWEIGHT=*value*

WEIGHT=*value*

WT=*value*

specifies the tuning weight w to use during the tuning phase of the random walk Metropolis sampler to tune the proposal covariance matrix. For more information, see the section “[Covariance Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, TUNEWEIGHT=0.75.

SAMPLERSETTINGS**SAMSETTINGS****SAMSET**

produces a summary of the MCMC sampler settings that are used to sample from the posterior distribution, including tuning settings.

SAMPLERSUMMARY<(VERBOSE)>**SAMSUMMARY<(VERBOSE)>****SAMSUM<(VERBOSE)>**

produces a summary of the MCMC sampler that is used to sample from the posterior distribution. For the RWM sampler, this option produces a table of the Metropolis acceptance rates. For NUTS, this option produces a table of the Metropolis acceptance rates as well as a table of NUTS diagnostics. You can specify the following suboption:

VERBOSE

produces a more detailed summary of the MCMC sampler. For the RWM sampler, this option does nothing. For NUTS, this option produces a table of the final mass matrix that is used after tuning. For more information, see the section “[Tuning the No-U-Turn Sampler](#)” on page 607.

SEED=number

specifies an integer seed in the range 1 to $2^{31} - 1$ for the random number generator in the simulation. Specifying a seed enables you to reproduce identical Markov chains for the same specification. If you do not specify this option, or if you specify a nonpositive seed, a random seed is generated by reading the time of day from the computer’s clock.

STATISTICS<(stat-option)>=ALL | NONE | keyword | (keyword-list)**STATISTIC<(stat-option)>=ALL | NONE | keyword | (keyword-list)****STATS<(stat-option)>=ALL | NONE | keyword | (keyword-list)****STAT<(stat-option)>=ALL | NONE | keyword | (keyword-list)**

specifies which posterior summary statistics to produce. If the value of the NCHAIN= option is greater than 1, then instead of the posterior summary statistics being computed separately for each Markov chain, by default the posterior summary statistics are computed by using all chains. For more information, see the section “[Using Multiple Chains](#)” on page 37 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *stat-option*:

BYCHAIN

produces posterior summary statistics tables for each Markov chain. If NCHAIN=1, then the BYCHAIN option does nothing.

To produce all statistics, specify STATISTICS=ALL. To produce no statistics, specify STATISTICS=NONE. To produce some but not all statistics, or to change certain settings of these statistics, specify one or more of the following keywords. If you specify multiple keywords, separate them by using spaces.

CORR**COR**

produces the posterior correlation matrix.

COV

produces the posterior covariance matrix.

INTERVAL < (*interval-option*) >**INTERVALS** < (*interval-option*) >**INT** < (*interval-option*) >

produces equal-tail posterior credible intervals and highest posterior density (HPD) credible intervals. For more information, see the sections “[Interval Estimation](#)” on page 10 and “[Summary Statistics](#)” on page 35 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *interval-option*:

ALPHA=*numeric-list*

ALPHAS=*numeric-list*

computes posterior credible intervals at the levels that you specify in the *numeric-list*. Elements in the *numeric-list* must be between 0 and 1, and values can be separated either by spaces or by commas. Repeated values are removed. Each value in the *numeric-list* produces a pair of 100(1-*value*)% equal-tail and HPD intervals for each parameter. By default, ALPHA=0.05, which yields the 95% credible intervals for each parameter.

SUMMARIES < (*summary-option*) >**SUMMARY** < (*summary-option*) >**SUM** < (*summary-option*) >

produces the means, standard deviations, and percentiles for the posterior sample. For more information, see the section “[Summary Statistics](#)” on page 35 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *summary-option*:

PERCENTILES=*numeric-list*

PERCENTILE=*numeric-list*

PERCENT=*numeric-list*

PERC=*numeric-list*

PCT=*numeric-list*

computes the percentiles of the posterior sample that you specify in the *numeric-list*. Elements in the *numeric-list* must be between 0 and 100, and values can be separated either by spaces or by commas. Repeated values are removed. By default, PERCENTILES=(2.5 25 50 75 97.5), which yields the 2.5th, 25th, 50th, 75th, and 97.5th percentiles, respectively, for each parameter.

By default, STATISTICS=SUMMARIES.

THIN=*k*

THINNING=*k*

thins the Markov chain by using only one of every *k* iterations. For more information, see the section “Burn-In, Thinning, and Markov Chain Samples” on page 27 in Chapter 2, “Introduction to Bayesian Analysis Procedures.” By default, THIN=1.

BOUNDS Statement

BOUNDS *bound1* < , *bound2* ... > ;

The BOUNDS statement imposes simple boundary constraints on the parameter estimates. BOUNDS statement constraints refer to the parameters that are estimated by the CQLIM procedure. You can specify any number of BOUNDS statements.

Each *bound* is composed of parameters, constants, and inequality operators. Parameters that are associated with regressor variables are referred to by the names of the corresponding regressor variables. Specify each bound as follows:

item operator item < *operator item* < *operator item* ... > >

Each *item* is a constant, the name of a parameter, or a list of parameter names. For more information about how parameters are named in the CQLIM procedure, see the section “Naming of Parameters” on page 614. Each *operator* is <, >, <=, or >=.

You can use both the BOUNDS statement and the RESTRICT statement to impose boundary constraints; however, the BOUNDS statement provides a simpler syntax for specifying these types of constraints. For more information, see the section “RESTRICT Statement” on page 596.

The following BOUNDS statement constrains the estimates of the parameters that are associated with the variable *ttime* and the variables *x1* through *x10* to be between 0 and 1. The following example illustrates the use of parameter lists to specify boundary constraints:

```
bounds 0 < ttime x1-x10 < 1;
```

The following BOUNDS statement constrains the estimates of the correlation (_RHO) and sigma (_SIGMA) in the bivariate model:

```
bounds _rho >= 0, _sigma.y1 > 1, _sigma.y2 < 5;
```

The BOUNDS statement is not supported if a BAYES statement is also specified. In Bayesian analysis, the restrictions on parameters are usually introduced through the prior distribution.

BY Statement

BY *variables* ;

A BY statement can be used in PROC CQLIM to obtain separate analyses of observations in groups defined by the BY variables.

CLASS Statement

CLASS *variable* <(options)>...<*variable* <(options)>> </global-options> ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. You are not required to list the response variable for discrete choice models in the CLASS statement. Table 12.2 summarizes the values that you can use for either an *option* or a *global-option*. By default, REF=LAST. The options are fully documented in the section “CLASS Statement” on page 56 in Chapter 4, “Shared Concepts.”

Table 12.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable

DISPLAY Statement

DISPLAY <*table-list*> </options> ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 615. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 615. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

ENDOGENOUS Statement

ENDOGENOUS *variables* ~ *options* ;

The ENDOGENOUS statement specifies the type of dependent variables that appear on the left-hand side of the equation. Endogenous variables listed refer to the dependent variables that appear on the left-hand side of the equation. You can specify the following options.

Discrete Variable Options

DISCRETE < (*discrete-options*) >

specifies that the endogenous variables in this statement be discrete. The BAYES statement is not compatible with ordinal models—that is, discrete models that have more than two levels of the endogenous variable. You can specify the following *discrete-options*:

DISTRIBUTION=LOGISTIC | NORMAL

DIST=LOGISTIC | NORMAL

D=LOGISTIC | NORMAL

specifies the cumulative distribution function to use to model the response probabilities. You can specify the following distribution types:

LOGISTIC specifies the logistic distribution for the logit model.

NORMAL specifies the normal distribution for the probit model.

By default, DISTRIBUTION=NORMAL.

If a multivariate model is specified, logistic distribution is not allowed. Only normal distribution is supported.

ORDER=FORMATTED | FREQ | INTERNAL

specifies the sort order for the levels of the discrete variables that you specify in the ENDOGENOUS statement. This sort order determines which parameters in the model correspond to each level in the data. You can specify the following sort orders:

DATA	sorts levels by order of appearance in the input data table.
FORMATTED	sorts levels by formatted value. The sort order is machine-dependent.
FREQ	sorts levels by descending frequency count; levels that have the most observations come first in the order.
INTERNAL	sorts levels by unformatted value. The sort order is machine-dependent.

By default, ORDER=FORMATTED. For more information about sort order, see the chapter on the SORT procedure in the *Base SAS Procedures Guide*.

Censored Variable Options

CENSORED (*censored-options*)

censors the endogenous variables in this statement. You can specify the following *censored-options*:

LB=*value* | *variable*

LOWERBOUND=*value* | *variable*

specifies the lower bound of the censored variables. If *value* is missing or the value of *variable* is missing, no lower bound is set. By default, no lower bound is set.

UB=*value* | *variable*

UPPERBOUND=*value* | *variable*

specifies the upper bound of the censored variables. If *value* is missing or the value of *variable* is missing, no upper bound is set. By default, no upper bound is set.

Truncated Variable Options

TRUNCATED (*truncated-options*)

truncates the endogenous variables in the statement. You can specify the following *truncated-options*:

LB=*value* | *variable*

LOWERBOUND=*value* | *variable*

specifies the lower bound of the truncated variables. If *value* is missing or the value of *variable* is missing, no lower bound is set. By default, no lower bound is set.

UB=*value* | *variable*

UPPERBOUND=*value* | *variable*

specifies the upper bound of the truncated variables. If *value* is missing or the value of *variable* is missing, no upper bound is set. By default, no upper bound is set.

Stochastic Frontier Variable Options

FRONTIER <(*frontier-options*)>

specifies a stochastic frontier model. You can specify the following *frontier-options*:

COST

specifies that the estimated model be a cost function.

PRODUCTION

specifies that the estimated model be a production function.

TYPE=EXPONENTIAL | HALF | TRUNCATED

specifies the model type. You can specify the following types:

EXPONENTIAL specifies an exponential model.

HALF specifies a half-normal model.

TRUNCATED specifies a truncated normal model.

If neither PRODUCTION nor COST is specified, a production function is estimated by default. The BAYES statement is not compatible with stochastic frontier models.

FREQ Statement

FREQ *variable* ;

The FREQ statement identifies a variable that contains the frequency of occurrence of each observation. PROC CQLIM treats each observation as if it appeared n times, where n is the value of the FREQ variable for the observation. If the frequency value is not an integer, it is truncated to an integer. If the frequency value is less than 1 or missing, the observation is not used in the model fitting. When the FREQ statement is not specified, each observation is assigned a frequency of 1. If you specify more than one FREQ statement, then only the first FREQ statement is used.

HETERO Statement

HETERO *dependent-variables* ~ *exogenous-variables* </ options > ;

The HETERO statement specifies variables that are related to the heteroscedasticity of the residuals and the way that these variables are used to model the error variance. PROC CQLIM supports the following heteroscedastic regression model:

$$y_i = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

$$\epsilon_i \sim N(0, \sigma_i^2)$$

For more information about the specification of functional forms, see the section “[Heteroscedasticity](#)” on page 602.

The HETERO statement is currently not supported with a BAYES statement.

You can use the HETERO statement only for univariate models.

The following *options* specify the functional forms of heteroscedasticity:

LINK=EXP | LINEAR

specifies the functional form. You can specify the following values:

EXP specifies the exponential link function:

$$\sigma_i^2 = \sigma^2(1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma}))$$

LINEAR specifies the linear link function:

$$\sigma_i^2 = \sigma^2(1 + \mathbf{z}_i' \boldsymbol{\gamma})$$

By default, LINK=EXP.

NOCONST

specifies that no constant appear in the exponential heteroscedasticity model:

$$\sigma_i^2 = \sigma^2 \exp(\mathbf{z}_i' \boldsymbol{\gamma})$$

SQUARE

estimates the model by using the square of the linear heteroscedasticity function. For example, you can specify the heteroscedasticity function

$$\sigma_i^2 = \sigma^2(1 + (\mathbf{z}_i' \boldsymbol{\gamma})^2)$$

by using the following code:

```
model y = x1 x2 / censored(lb=0);
hetero y ~ z1 / link=linear square;
```

The SQUARE option does not apply to the exponential heteroscedasticity function, because the square of an exponential function of $\mathbf{z}_i' \boldsymbol{\gamma}$ is the same as the exponential of $2\mathbf{z}_i' \boldsymbol{\gamma}$. Hence, the only difference is that all $\boldsymbol{\gamma}$ estimates are divided by two.

This option is ignored if you do not specify the LINK= option.

INIT Statement

INIT *initvalue1* < , *initvalue2* ... > ;

The INIT statement sets initial values for parameters in the optimization. You can specify any number of INIT statements.

Each *initvalue* is written as a parameter or parameter list, followed by an optional equality operator (=), followed by a number:

parameter <=> *number*

MODEL Statement

MODEL *dependent-variables* < (*order-options*) > = *regressors* < / *options* > ;

The MODEL statement specifies the dependent variable and independent regressor variables for the regression model.

You can specify the following *option* after a slash (/):

NOINT

suppresses the intercept parameter.

You can also specify the following endogenous variable options, which are the same as the options in the ENDOGENOUS statement. If you specify an endogenous variable option in both the MODEL statement and the ENDOGENOUS statement, only the option in the ENDOGENOUS statement is used.

Discrete Variable Options

DISCRETE < (*discrete-options*) >

specifies that the endogenous variables in the MODEL statement be discrete. The BAYES statement is not compatible with ordinal models—that is, discrete models that have more than two levels of the response variable. You can specify the following *discrete-options*:

DISTRIBUTION=LOGISTIC | NORMAL

DIST=LOGISTIC | NORMAL

D=LOGISTIC | NORMAL

specifies the cumulative distribution function to use to model the response probabilities. You can specify the following distribution types:

LOGISTIC specifies the logistic distribution for the logit model.

NORMAL specifies the normal distribution for the probit model.

By default, DISTRIBUTION=NORMAL.

If a multivariate model is specified, logistic distribution is not allowed. Only normal distribution is supported.

ORDER=FORMATTED | FREQ | INTERNAL

specifies the sort order for the levels of the discrete variables that you specify in the ENDOGENOUS statement. The sort order determines which parameters in the model correspond to each level in the data. You can specify the following sort orders:

FORMATTED sorts levels by formatted value. The sort order is machine-dependent.

FREQ sorts levels by descending frequency count; levels that have the most observations come first in the order.

INTERNAL sorts levels by unformatted value. The sort order is machine-dependent.

By default, ORDER=FORMATTED. For more information about sort order, see the chapter on the SORT procedure in the *Base SAS Procedures Guide*.

Censored Variable Options

CENSORED <(censored-options)>

censors the endogenous variables in this statement. You can specify the following *censored-options*:

LB=*value* | *variable*

LOWERBOUND=*value* | *variable*

specifies the lower bound of the censored variables. If *value* is missing or the value of *variable* is missing, no lower bound is set. By default, no lower bound is set.

UB=*value* | *variable*

UPPERBOUND=*value* | *variable*

specifies the upper bound of the censored variables. If *value* is missing or the value of *variable* is missing, no upper bound is set. By default, no upper bound is set.

Truncated Variable Options

TRUNCATED <(truncated-options)>

truncates the endogenous variables in this statement. You can specify the following *truncated-options*:

LB=*value* | *variable*

LOWERBOUND=*value* | *variable*

specifies the lower bound of the truncated variables. If *value* is missing or the value of *variable* is missing, no lower bound is set. By default, no lower bound is set.

UB=*value* | *variable*

UPPERBOUND=*value* | *variable*

specifies the upper bound of the truncated variables. If *value* is missing or the value of *variable* is missing, no upper bound is set. By default, no upper bound is set.

Stochastic Frontier Variable Options

FRONTIER <(frontier-options)>

specifies the stochastic frontier model. You can specify the following *frontier-options*:

COST

specifies that the estimated model be a cost function.

PRODUCTION

specifies that the estimated model be a production function.

TYPE=EXPONENTIAL | HALF | TRUNCATED

specifies the model type.

EXPONENTIAL specifies an exponential model.

HALF specifies a half-normal model.

TRUNCATED specifies a truncated normal model.

If neither PRODUCTION nor COST is specified, a production function is estimated by default. The BAYES statement is not compatible with stochastic frontier models.

OUTPUT Statement

OUTPUT **OUT=***libref.data-table* < *output-options* > ;

The OUTPUT statement creates a new SAS data table to contain variables that you specify using the COPYVAR option and the following quantities of interest if they are specified as *output-options*: estimates of $\mathbf{x}'\boldsymbol{\beta}$, predicted value, residual, marginal effects, probability, standard deviation of the error, expected value, conditional expected value, technical efficiency measures, and inverse Mills ratio. When the response values are missing for an observation, all output estimates except the residual for that observation are still computed as long as none of the explanatory variables are missing. This enables you to compute these statistics for prediction. You can specify only one OUTPUT statement.

You must specify the OUT= option:

OUT=*libref.data-table*

names the output data table for PROC CQLIM to use. You must specify this option before any other options. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 562.

data-table specifies the name of the output data table.

You can specify one or more of the following *output-options*:

CONDITIONAL

outputs estimates of conditional expected values of continuous endogenous variables.

COPYVAR=*SAS-variable-names*

COPYVARS=(*SAS-variable-names*)

adds any SAS variables that appear in the input data table to the output data table.

ERRSTD

outputs estimates of σ_j , the standard deviation of the error term.

EXPECTED

outputs estimates of expected values of continuous endogenous variables.

MARGINAL

outputs marginal effects.

MILLS

outputs estimates of inverse Mills ratios of censored or truncated continuous variables and binary discrete variables.

PREDICTED

outputs estimates of predicted endogenous variables.

PROB

outputs estimates of probability of discrete endogenous variables that take the current observed responses.

PROBALL

outputs estimates of probability of discrete endogenous variables for all possible responses.

RESIDUAL

outputs estimates of residuals of continuous endogenous variables.

TE1

outputs estimates of technical efficiency for each producer in the stochastic frontier model that is suggested by Battese and Coelli (1988).

TE2

outputs estimates of technical efficiency for each producer in the stochastic frontier model that is suggested by Jondrow et al. (1982).

XBETA

outputs estimates of $\mathbf{x}'\boldsymbol{\beta}$.

PRIOR Statement

PRIOR *parameter-list* $\sim$ *distribution* ;

The PRIOR statement specifies the prior distribution of the model parameters for Bayesian analysis. You must specify a single parameter or a list of parameters, a tilde ($\sim$), and then a distribution with its parameters. You can specify multiple PRIOR statements, although PRIOR statements for parameters that do not exist in the model are ignored. For information about how parameters are named so that you can refer to them when specifying prior distributions, see the section “[Naming of Parameters](#)” on page 614. Alternatively, specifying the SHOWNAMES suboption of the [PRIORSUMMARY](#) option in the BAYES statement displays the names that are used for all parameters in the prior summary table.

You can specify the following *distributions*. All hyperparameters of the distribution must be specified, except LOWER and UPPER, which determine the support of the distribution and allow for truncated versions of many of them. For more information about these *distributions*, see the section “[Standard Prior Distributions](#)” on page 12 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” For more information about truncated prior distributions, see the section “[Truncated Prior Distributions](#)” on page 609.

CAUCHY(*cauchy-options*)

specifies independently and identically distributed [Cauchy distributions](#) for the *parameter-list*.

You must specify the following *cauchy-options*:

LOCATION= μ

LOC= μ

specifies the location parameter of the Cauchy distribution.

SCALE= γ

specifies the scale parameter of the Cauchy distribution, where $\gamma > 0$.

You can also specify the following *cauchy-options*:

LOWER= L | NEGATIVEINFINITY**LB= L | NEGATIVEINFINITY**

specifies the lower bound of the Cauchy distribution, where $L < U$. You can specify the following values:

L

uses L as the lower bound, such that the prior becomes a truncated Cauchy distribution.

NEGATIVEINFINITY**NEGINF****NEGINF**

uses negative infinity as the lower bound.

By default, LOWER=NEGATIVEINFINITY.

UPPER= U | INFINITY**UB= U | INFINITY**

specifies the upper bound of the Cauchy distribution, where $U > L$. You can specify the following values:

U

uses U as the upper bound, such that the prior becomes a truncated Cauchy distribution.

INFINITY**INF****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

GAMMA(*gamma-options*)

specifies independently and identically distributed [gamma distributions](#) for the *parameter-list*. You must specify the following *gamma-options*:

SCALE= b

specifies the scale parameter of the gamma distribution, where $b > 0$.

SHAPE= a

specifies the shape parameter of the gamma distribution, where $a > 0$.

You can also specify the following *gamma-options*:

LOWER=L**LB=L**

specifies the lower bound of the gamma distribution, where $L \geq 0$ and $L < U$. If $L > 0$, the prior becomes a truncated gamma distribution.

By default, LOWER=0.

UPPER=U | INFINITY**UB=U | INFINITY**

specifies the upper bound of the gamma distribution, where $U > L$. You can specify the following values:

U

uses *U* as the upper bound, such that the prior becomes a truncated gamma distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

IGAMMA(igamma-options)

specifies independently and identically distributed [inverse gamma distributions](#) for the *parameter-list*. You must specify the following *igamma-options*:

SCALE=b

specifies the scale parameter of the inverse gamma distribution, where $b > 0$.

SHAPE=a

specifies the shape parameter of the inverse gamma distribution, where $a > 0$.

You can also specify the following *igamma-options*:

LOWER=L**LB=L**

specifies the lower bound of the inverse gamma distribution, where $L \geq 0$ and $L < U$. If $L > 0$, the prior becomes a truncated inverse gamma distribution.

By default, LOWER=0.

UPPER=U | INFINITY**UB=U | INFINITY**

specifies the upper bound of the inverse gamma distribution, where $U > L$. You can specify the following values:

U

uses *U* as the upper bound, such that the prior becomes a truncated inverse gamma distribution.

INFINITY

INFTY

INF

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

NORMAL(*normal-options*)

specifies independently and identically distributed **normal distributions** for the *parameter-list*. You must specify the MEAN= *normal-option* and either the SD= or VAR= *normal-option*:

MEAN= μ

specifies the mean parameter of the normal distribution.

SD= σ

specifies the standard deviation parameter of the normal distribution, where $\sigma > 0$.

VAR= σ^2

specifies the variance parameter of the normal distribution, where $\sigma^2 > 0$.

You can also specify the following *normal-options*:

LOWER=*L* | **NEGATIVEINFINITY**

LB=*L* | **NEGATIVEINFINITY**

specifies the lower bound of the normal distribution, where $L < U$. You can specify the following values:

L

uses *L* as the lower bound, such that the prior becomes a truncated normal distribution.

NEGATIVEINFINITY

NEGINFTY

NEGINF

uses negative infinity as the lower bound.

By default, LOWER=NEGATIVEINFINITY.

UPPER=*U* | **INFINITY**

UB=*U* | **INFINITY**

specifies the upper bound of the normal distribution, where $U > L$. You can specify the following values:

U

uses *U* as the upper bound, such that the prior becomes a truncated normal distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

SQRTGAMMA(*sqrtgamma-options*)

specifies independently and identically distributed [square root gamma distributions](#) for the *parameter-list*. You must specify the following *sqrtgamma-options*:

SCALE=*b*

specifies the scale parameter of the square root gamma distribution, where $b > 0$.

SHAPE=*a*

specifies the shape parameter of the square root gamma distribution, where $a > 0$.

You can also specify the following *sqrtgamma-options*:

LOWER=*L***LB=*L***

specifies the lower bound of the square root gamma distribution, where $L \geq 0$ and $L < U$. If $L > 0$, the prior becomes a truncated square root gamma distribution.

By default, LOWER=0.

UPPER=*U* | INFINITY**UB=*U* | INFINITY**

specifies the upper bound of the square root gamma distribution, where $U > L$. You can specify the following values:

U

uses *U* as the upper bound, such that the prior becomes a truncated square root gamma distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

SQRTIGAMMA(*sqrtigamma-options*)

specifies independently and identically distributed [square root inverse gamma distributions](#) for the *parameter-list*. You must specify the following *sqrtigamma-options*:

SCALE= b

specifies the scale parameter of the square root inverse gamma distribution, where $b > 0$.

SHAPE= a

specifies the shape parameter of the square root inverse gamma distribution, where $a > 0$.

You can also specify the following *sqrtigamma-options*:

LOWER= L **LB= L**

specifies the lower bound of the square root inverse gamma distribution, where $L \geq 0$ and $L < U$. If $L > 0$, the prior becomes a truncated square root inverse gamma distribution.

By default, LOWER=0.

UPPER= U | INFINITY**UB= U | INFINITY**

specifies the upper bound of the square root inverse gamma distribution, where $U > L$. You can specify the following values:

U

uses U as the upper bound, such that the prior becomes a truncated square root inverse gamma distribution.

INFINITY

INFTY

INF

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

T(*t-options*)

specifies independently and identically distributed *t distributions* for the *parameter-list*.

You must specify the following *t-options*:

DF= ν

specifies the degrees-of-freedom parameter of the *t* distribution, where $\nu > 0$.

LOCATION= μ **LOC= μ**

specifies the location parameter of the *t* distribution.

SCALE= γ

specifies the scale parameter of the *t* distribution, where $\gamma > 0$.

You can also specify the following *t-options*:

LOWER=L | NEGATIVEINFINITY**LB=L | NEGATIVEINFINITY**

specifies the lower bound of the t distribution, where $L < U$. You can specify the following values:

L

uses L as the lower bound, such that the prior becomes a truncated t distribution.

NEGATIVEINFINITY**NEGINF****NEGINF**

uses negative infinity as the lower bound.

By default, LOWER=NEGATIVEINFINITY.

UPPER=U | INFINITY**UB=U | INFINITY**

specifies the upper bound of the t distribution, where $U > L$. You can specify the following values:

U

uses U as the upper bound, such that the prior becomes a truncated t distribution.

INFINITY**INFTY****INF**

uses positive infinity as the upper bound.

By default, UPPER=INFINITY.

UNIFORM(uniform-options)

specifies independently and identically distributed **uniform distributions** for the *parameter-list*. You must specify both the following *uniform-options*:

MAX=M | INFINITY

specifies the maximum value of the uniform distribution, where $M < m$. You can specify the following values:

M

uses M as the maximum value.

INFINITY**INFTY****INF**

uses positive infinity as the maximum value, such that the prior becomes an improper uniform prior.

MIN=*m* | NEGATIVEINFINITY

specifies the minimum value of the uniform distribution, where $m < M$. You can specify the following values:

m

uses *m* as the minimum value.

NEGATIVEINFINITY**NEGINFTY****NEGINF**

uses negative infinity as the minimum value, such that the prior becomes an improper uniform prior.

RESTRICT Statement

RESTRICT *restriction1* <, *restriction2* ... > ;

The RESTRICT statement imposes linear restrictions on the parameter estimates. You can specify any number of RESTRICT statements, but the number of restrictions that are imposed is limited by the number of regressors.

Each *restriction* is written as an expression, followed by an equality operator (=) or an inequality operator (<, >, <=, >=), followed by a second expression:

expression operator expression

The *operator* can be =, <, >, <=, or >=. The *operator* and second *expression* are optional.

Restriction expressions can be composed of parameter names; multiplication (*), addition (+), and subtraction (−) operators; and constants. Parameters that are named in restriction expressions must be among the parameters that are estimated by the model. Parameters that are associated with a regressor variable are referred to by the name of the corresponding regressor variable. The restriction expressions must be a linear function of the parameters.

The following statements illustrate the use of the RESTRICT statement:

```
proc cqlim data=mylib.dataset;
  model y = x1-x10 / censored(lb=0);
  restrict x1*x2 <= x2 + x3;
run;
```

The RESTRICT statement can also be used to impose cross-equation restrictions in multivariate models. The following RESTRICT statement imposes an equality restriction on coefficients of *x1* in equation *y1* and *x1* in equation *y2*:

```
proc cqlim data=mylib.dataset;
  model y1 = x1-x10 / discrete;
  model y2 = x1-x4 / discrete;
  restrict y1.x1=y2.x1;
run;
```

Lagrange multipliers are reported in the “Parameter Estimates” table for all the active linear constraints. They are identified by the names Restrict1, Restrict2, and so on. The test statistics and p -values for these restrictions are computed using the Lagrange multiplier test that is described in the section “Tests on Parameters” on page 604. Nonactive (nonbinding) restrictions have no effect on the estimation results and are not noted in the output.

The RESTRICT statement is not supported if a BAYES statement is also specified. In Bayesian analysis, the restrictions on parameters are usually introduced through the prior distribution.

TEST Statement

`<label:> TEST <'string'> equation <, equation .../ options > ;`

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the regression parameters in the preceding MODEL statement. Each equation specifies a linear hypothesis to be tested. All hypotheses in one TEST statement are tested jointly. Variable names in the equations must correspond to regressors in the preceding MODEL statement, and each name represents the coefficient of the corresponding regressor. Use the keyword INTERCEPT for a test that includes a constant.

You cannot specify both the TEST statement and the BAYES statement.

You can specify the following *options* after the slash (/):

ALL

requests Wald, Lagrange multiplier, and likelihood ratio tests.

LM

requests the Lagrange multiplier test.

LR

requests the likelihood ratio test.

WALD

requests the Wald test.

The following statements illustrate the use of the TEST statement (note the use of the INTERCEPT keyword in the second TEST statement):

```
proc cqlim;
  model y = x1 x2 x3;
  test x1 = 0, x2 * .5 + 2 * x3 = 0;
  test_int: test intercept = 0, x3 = 0;
run;
```

The first TEST statement investigates the joint hypothesis that

$$\beta_1 = 0$$

and

$$0.5\beta_2 + 2\beta_3 = 0$$

In case there is more than one MODEL statement in one CQLIM procedure, then TEST statement is capable of testing cross equation restrictions. Each parameter reference should be preceded by the name of the dependent variable of the particular model and the dot sign. For example,

```
proc cqlim;
  model y1 = x1 x2 x3;
  model y2 = x3 x5 x6;
  test y1.x1 + y2.x6 = 1;
run;
```

This cross equation test investigates the null hypothesis that

$$\beta_{1,1} + \beta_{2,3} = 1$$

in the system of equations

$$y_{1,i} = \alpha_1 + \beta_{1,1}x_{1,i} + \beta_{1,2}x_{2,i} + \beta_{1,3}x_{3,i}$$

$$y_{2,i} = \alpha_2 + \beta_{2,1}x_{3,i} + \beta_{2,2}x_{5,i} + \beta_{2,3}x_{6,i}$$

Only linear equality restrictions and tests are permitted in PROC CQLIM. Test expressions can be composed only of algebraic operations that involve the addition symbol (+), subtraction symbol (–), and multiplication symbol (*).

The TEST statement accepts labels that are reproduced in the printed output. You can label a TEST statement in two ways: you can specify a label followed by a colon before the TEST keyword, or you can specify a string in quotation marks after the TEST keyword. If you specify both a label before the TEST keyword and a quoted string after the keyword, PROC CQLIM uses the label that precedes the colon. If no label or quoted string is specified, PROC CQLIM labels the test automatically.

WEIGHT Statement

WEIGHT *variable* </option> ;

The WEIGHT statement specifies a variable that supplies weighting values to use for each observation in estimating parameters. The log likelihood for each observation is multiplied by the corresponding weight variable value.

If the weight of an observation is nonpositive, that observation is not used in the estimation.

You can add the following *option* after a slash (/):

NONNORMALIZE

specifies that the weights must be used as is. When you omit this option, the weights are normalized so that they add up to the actual sample size. Weights w_i are normalized by multiplying them by $\frac{n}{\sum_{i=1}^n w_i}$, where n is the sample size.

For more information about combining the WEIGHT statement with the [BAYES](#) statement, see the section “[Bayesian Analysis with Weighted Likelihoods](#)” on page 607.

Details: CQLIM Procedure

Ordinal Discrete Choice Modeling

Binary Probit and Logit Model

The binary choice model is

$$y_i^* = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

where the value of the latent dependent variable, y_i^* , is observed only as follows:

$$\begin{aligned} y_i &= 1 && \text{if } y_i^* > 0 \\ &= 0 && \text{otherwise} \end{aligned}$$

The disturbance, ϵ_i , of the probit model has a standard normal distribution with the cumulative distribution function (CDF)

$$\Phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} \exp(-t^2/2) dt$$

The disturbance of the logit model has a standard logistic distribution with the CDF

$$\Lambda(x) = \frac{\exp(x)}{1 + \exp(x)} = \frac{1}{1 + \exp(-x)}$$

The binary discrete choice model has the following probability that the event $\{y_i = 1\}$ occurs:

$$P(y_i = 1) = F(\mathbf{x}_i' \boldsymbol{\beta}) = \begin{cases} \Phi(\mathbf{x}_i' \boldsymbol{\beta}) & \text{(probit)} \\ \Lambda(\mathbf{x}_i' \boldsymbol{\beta}) & \text{(logit)} \end{cases}$$

For more information, see *SAS/ETS User's Guide*.

Ordinal Probit/Logit

When the dependent variable is observed in sequence with M categories, binary discrete choice modeling is not appropriate for data analysis. McKelvey and Zavoina (1975) propose the ordinal (or ordered) probit model.

Consider the regression equation

$$y_i^* = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

where error disturbances, ϵ_i , have the distribution function F . The unobserved continuous random variable, y_i^* , is identified as M categories. Suppose there are $M + 1$ real numbers, $\mu_0, \dots, \mu_M$, where $\mu_0 = -\infty$, $\mu_1 = 0$, $\mu_M = \infty$, and $\mu_0 \leq \mu_1 \leq \dots \leq \mu_M$. Define

$$R_{i,j} = \mu_j - \mathbf{x}_i' \boldsymbol{\beta}$$

The probability that the unobserved dependent variable is contained in the j th category can be written as

$$P[\mu_{j-1} < y_i^* \leq \mu_j] = F(R_{i,j}) - F(R_{i,j-1})$$

For more information, see *SAS/ETS User's Guide*.

Limited Dependent Variable Models

Censored Regression Models

When the dependent variable is censored, all values within a certain range are transformed to a single value. For example, the standard Tobit model can be defined as

$$y_i^* = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

$$y_i = \begin{cases} y_i^* & \text{if } y_i^* > 0 \\ 0 & \text{if } y_i^* \leq 0 \end{cases}$$

where $\epsilon_i \sim \text{iid}N(0, \sigma^2)$.

The Tobit model can be generalized to handle observation-by-observation censoring. The model that is censored on both the lower and upper limits can be defined as

$$y_i = \begin{cases} R_i & \text{if } y_i^* \geq R_i \\ y_i^* & \text{if } L_i < y_i^* < R_i \\ L_i & \text{if } y_i^* \leq L_i \end{cases}$$

For more information, see *SAS/ETS User's Guide*.

Truncated Regression Models

In a truncated model, the observed sample is a subset of the population in which the dependent variable falls within a certain range. For example, when neither a dependent variable nor exogenous variables are observed for $y_i^* \leq 0$, the truncated regression model can be specified as

$$\ell = \sum_{i \in \{y_i > 0\}} \left\{ -\ln \Phi(\mathbf{x}_i' \boldsymbol{\beta} / \sigma) + \ln \left[\frac{\phi((y_i - \mathbf{x}_i' \boldsymbol{\beta}) / \sigma)}{\sigma} \right] \right\}$$

For more information, see *SAS/ETS User's Guide*.

Stochastic Frontier Production and Cost Models

Stochastic frontier production models were first developed by Aigner, Lovell, and Schmidt (1977); Meeusen and van den Broeck (1977). Specification of these models allows for random shocks of the production or cost but also includes a term for technical or cost inefficiency. Assuming that the production function takes a log-linear Cobb-Douglas form, the stochastic frontier production model can be written as

$$\ln(y_i) = \beta_0 + \sum_n \beta_n \ln(x_{ni}) + \epsilon_i$$

where $\epsilon_i = v_i - u_i$. The v_i term represents the stochastic error component, and the u_i term represents the nonnegative, technical inefficiency error component. The v_i error component is assumed to be distributed iid normal and independent from u_i . If $u_i > 0$, the error term ϵ_i is negatively skewed and represents technical inefficiency. If $u_i < 0$, the error term ϵ_i is positively skewed and represents cost inefficiency. PROC CQLIM models the u_i error component as a half-normal, exponential, or truncated normal distribution.

The Normal-Half-Normal Model

When v_i is iid $N(0, \sigma_v^2)$ in a normal-half-normal model, u_i is iid $N^+(0, \sigma_u^2)$, with v_i and u_i independent of each other. Given the independence of error terms, the joint density of v and u can be written as

$$f(u, v) = \frac{2}{2\pi\sigma_u\sigma_v} \exp\left\{-\frac{u^2}{2\sigma_u^2} - \frac{v^2}{2\sigma_v^2}\right\}$$

Substituting $v = \epsilon + u$ into the preceding equation and integrating u out gives

$$f(\epsilon) = \frac{2}{\sigma} \phi\left(\frac{\epsilon}{\sigma}\right) \Phi\left(-\frac{\epsilon\lambda}{\sigma}\right)$$

where $\lambda = \sigma_u/\sigma_v$ and $\sigma = \sqrt{\sigma_u^2 + \sigma_v^2}$.

In the case of a stochastic frontier cost model, $v = \epsilon - u$ and

$$f(\epsilon) = \frac{2}{\sigma} \phi\left(\frac{\epsilon}{\sigma}\right) \Phi\left(\frac{\epsilon\lambda}{\sigma}\right)$$

For more information, see *SAS/ETS User's Guide*.

The Normal-Exponential Model

Under the normal-exponential model, v_i is iid $N(0, \sigma_v^2)$ and u_i is iid exponential. Given the independence of the error term components u_i and v_i , the joint density of v and u can be written as

$$f(u, v) = \frac{1}{\sqrt{2\pi}\sigma_u\sigma_v} \exp\left\{-\frac{u}{\sigma_u} - \frac{v^2}{2\sigma_v^2}\right\}$$

The marginal density function of ϵ for the production function is

$$f(\epsilon) = \left(\frac{1}{\sigma_u}\right) \Phi\left(-\frac{\epsilon}{\sigma_v} - \frac{\sigma_v}{\sigma_u}\right) \exp\left\{\frac{\epsilon}{\sigma_u} + \frac{\sigma_v^2}{2\sigma_u^2}\right\}$$

The marginal density function for the cost function is equal to

$$f(\epsilon) = \left(\frac{1}{\sigma_u}\right) \Phi\left(\frac{\epsilon}{\sigma_v} - \frac{\sigma_v}{\sigma_u}\right) \exp\left\{-\frac{\epsilon}{\sigma_u} + \frac{\sigma_v^2}{2\sigma_u^2}\right\}$$

For more information, see *SAS/ETS User's Guide*.

The Normal–Truncated Normal Model

The normal–truncated normal model is a generalization of the normal-half-normal model that allows the mean of u_i to differ from 0. Under the normal–truncated normal model, the error term component v_i is iid $N^+(0, \sigma_v^2)$, and the error term component u_i is iid $N(\mu, \sigma_u^2)$. The joint density of v_i and u_i can be written as

$$f(u, v) = \frac{1}{\sqrt{2\pi}\sigma_u\sigma_v\Phi(\mu/\sigma_u)} \exp\left\{-\frac{(u-\mu)^2}{2\sigma_u^2} - \frac{v^2}{2\sigma_v^2}\right\}$$

The marginal density function of ϵ for the production function is

$$f(\epsilon) = \frac{1}{\sigma}\phi\left(\frac{\epsilon + \mu}{\sigma}\right)\Phi\left(\frac{\mu}{\sigma\lambda} - \frac{\epsilon\lambda}{\sigma}\right)\left[\Phi\left(\frac{\mu}{\sigma_u}\right)\right]^{-1}$$

The marginal density function for the cost function is

$$f(\epsilon) = \frac{1}{\sigma}\phi\left(\frac{\epsilon - \mu}{\sigma}\right)\Phi\left(\frac{\mu}{\sigma\lambda} + \frac{\epsilon\lambda}{\sigma}\right)\left[\Phi\left(\frac{\mu}{\sigma_u}\right)\right]^{-1}$$

For more information, see *SAS/ETS User's Guide*.

For more information about normal-half-normal, normal-exponential, and normal–truncated normal models, see Kumbhakar and Lovell (2000); Coelli, Prasada Rao, and Battese (1998).

Heteroscedasticity

If the variance of regression disturbance ϵ_i is heteroscedastic, the variance can be specified as a function of variables

$$E(\epsilon_i^2) = \sigma_i^2 = f(\mathbf{z}_i'\boldsymbol{\gamma})$$

Table 12.3 shows various functional forms of heteroscedasticity and the corresponding options to request each model.

Table 12.3 Specification Summary for Modeling Heteroscedasticity

Number	Model	Options
1	$f(\mathbf{z}_i'\boldsymbol{\gamma}) = \sigma^2(1 + \exp(\mathbf{z}_i'\boldsymbol{\gamma}))$	LINK=EXP (default)
2	$f(\mathbf{z}_i'\boldsymbol{\gamma}) = \sigma^2 \exp(\mathbf{z}_i'\boldsymbol{\gamma})$	LINK=EXP NOCONST
3	$f(\mathbf{z}_i'\boldsymbol{\gamma}) = \sigma^2(1 + \sum_{l=1}^L \gamma_l z_{li})$	LINK=LINEAR
4	$f(\mathbf{z}_i'\boldsymbol{\gamma}) = \sigma^2(1 + (\sum_{l=1}^L \gamma_l z_{li})^2)$	LINK=LINEAR SQUARE

In model 3, variance of some observations might be negative. Although the CQLIM procedure assigns a large penalty to move the optimization away from such a region, the optimization might not be able to improve the objective function value and might become locked in the region. Signs of such an outcome include extremely small likelihood values or missing standard errors in the estimates. In model 2, variance is guaranteed to be greater than or equal to 0, but variances of some observations might be very close to 0. In these scenarios, standard errors might be missing. Models 1 and 4 do not have such problems. Variances in these models are always positive and never close to 0.

Bivariate Censored Dependent Variable Modeling

The generic form of a bivariate censored dependent variable model is

$$\begin{aligned} y_{1i}^* &= \mathbf{x}_{1i}'\boldsymbol{\beta}_1 + \epsilon_{1i} \\ y_{2i}^* &= \mathbf{x}_{2i}'\boldsymbol{\beta}_2 + \epsilon_{2i} \end{aligned}$$

where the disturbances, ϵ_{1i} and ϵ_{2i} , have a joint normal distribution with zero mean, standard deviations σ_1 and σ_2 , and correlation ρ . The variables y_1^* and y_2^* are latent variables. The dependent variables y_1 and y_2 might or might not be censored at the edges of the bivariate interval $\{[L_1, R_1], [L_2, R_2]\}$, depending on the behavior of the latent variables y_1^* and y_2^* :

$$y_{1i} = \begin{cases} R_1 & \text{if } R_1 < y_{1i}^* \\ y_{1i}^* & \text{if } L_1 \leq y_{1i}^* \leq R_1 \\ L_1 & \text{if } y_{1i}^* < L_1 \end{cases}$$

$$y_{2i} = \begin{cases} R_2 & \text{if } R_2 < y_{2i}^* \\ y_{2i}^* & \text{if } L_2 \leq y_{2i}^* \leq R_2 \\ L_2 & \text{if } y_{2i}^* < L_2 \end{cases}$$

There are three cases for the log likelihood of (y_{1i}, y_{2i}) . In the first case, $y_{1i} = y_{1i}^*$ and $y_{2i} = y_{2i}^*$. That is, both dependent variables are uncensored. The log likelihood is computed from a bivariate normal density,

$$\ell_i = \ln [\text{pdf}(y_{1i}^*, y_{2i}^*)] = \ln \left[\phi_2 \left(\frac{y_{1i} - \mathbf{x}_{1i}'\boldsymbol{\beta}_1}{\sigma_1}, \frac{y_{2i} - \mathbf{x}_{2i}'\boldsymbol{\beta}_2}{\sigma_2}, \rho \right) \right] - \ln \sigma_1 - \ln \sigma_2$$

where $\phi_2(u, v, \rho)$ is the probability density function (pdf) for a standardized bivariate normal distribution with correlation ρ ,

$$\phi_2(u, v, \rho) = \frac{e^{-(1/2)(u^2 + v^2 - 2\rho uv)/(1 - \rho^2)}}{2\pi(1 - \rho^2)^{1/2}}$$

In the second case, one dependent variable is censored and the other is not. For example, if $y_{1i} = y_{1i}^*$ and $y_{2i} = L_2$, then the log likelihood is computed as

$$\begin{aligned} \ell_i &= \ln \left[\int_{-\infty}^{L_2} \text{pdf}(y_{1i}^*, y_{2i}^*) dy_{2i}^* \right] = \ln \left[\int_{-\infty}^{L_2} \text{pdf}(y_{2i}^* | y_{1i}^*) \text{pdf}(y_{1i}^*) dy_{2i}^* \right] \\ &= \ln \left[\phi \left(\frac{y_{1i}^* - \mathbf{x}_{1i}'\boldsymbol{\beta}_1}{\sigma_1} \right) \right] - \ln \sigma_1 + \ln \left[\Phi \left(\frac{L_2 - \mathbf{x}_{2i}'\boldsymbol{\beta}_2 - \sigma_2 \rho \frac{y_{1i}^* - \mathbf{x}_{1i}'\boldsymbol{\beta}_1}{\sigma_1}}{\sigma_2 \sqrt{1 - \rho^2}} \right) \right] \end{aligned}$$

where ϕ and Φ are the probability density function and the cumulative probability function for a standardized univariate normal distribution, respectively.

In the third case, both dependent variables are censored. For example, if $y_{1i} = R_1$ and $y_{2i} = L_2$, then the log likelihood is

$$\ell_i = \ln \left[\int_{u=\frac{R_1 - \mathbf{x}_1' \boldsymbol{\beta}_1}{\sigma_1}}^{\infty} \int_{v=-\infty}^{\frac{L_2 - \mathbf{x}_2' \boldsymbol{\beta}_2}{\sigma_2}} \phi_2(u, v, \rho) du dv \right]$$

Multivariate Limited Dependent Models

Multivariate models are similar to bivariate models. The generic form of the multivariate limited dependent variable model is

$$\begin{aligned} y_{1i}^* &= \mathbf{x}_{1i}' \boldsymbol{\beta}_1 + \epsilon_{1i} \\ y_{2i}^* &= \mathbf{x}_{2i}' \boldsymbol{\beta}_2 + \epsilon_{2i} \\ &\vdots \\ y_{mi}^* &= \mathbf{x}_{mi}' \boldsymbol{\beta}_m + \epsilon_{mi} \end{aligned}$$

where m is the number of models to be estimated. The vector ϵ has a multivariate normal distribution with mean 0 and variance-covariance matrix Σ . As with bivariate models, calculating the likelihood can involve computing multivariate normal integrations. This is done using Monte Carlo integration. (See Genz 1992; Hajivassiliou and McFadden 1998.)

When the number of equations, N , increases in a system, the number of parameters increases at the rate of N^2 because of the correlation matrix. When the number of parameters is large, sometimes the optimization converges but some of the standard deviations are missing. This usually means that the model is overparameterized. The default method for computing the covariance is to use the inverse Hessian matrix. The Hessian is computed by finite differences, and in overparameterized cases, the inverse cannot be computed. It is recommended that you reduce the number of parameters in such cases. Sometimes it might also help to use the outer product covariance matrix (the COVEST=OP option in the PROC CQLIM statement).

Tests on Parameters

In general, you can write the tested hypothesis as

$$H_0 : \mathbf{h}(\theta) = 0$$

where $\mathbf{h}(\theta)$ is an $r \times 1$ vector-valued function of the parameters θ that is given by the r expressions that you specify in the TEST statement.

Let $\hat{V}$ be the estimate of the covariance matrix of $\hat{\theta}$. Let $\hat{\theta}$ be the unconstrained estimate of θ , and let $\tilde{\theta}$ be the constrained estimate of θ such that $\mathbf{h}(\tilde{\theta}) = 0$. Let

$$A(\theta) = \partial \mathbf{h}(\theta) / \partial \theta \big|_{\hat{\theta}}$$

Using this notation, you compute the test statistics for the three types of tests as follows:

- The Wald test statistic is defined as

$$W = h'(\hat{\theta}) \left(A(\hat{\theta}) \hat{V} A'(\hat{\theta}) \right)^{-1} h(\hat{\theta})$$

The Wald test is not invariant to reparameterization of the model (Gregory and Veall 1985; Gallant 1987, p. 219). For more information about the theoretical properties of the Wald test, see Phillips and Park (1988).

- The Lagrange multiplier test statistic is

$$LM = \lambda' A(\tilde{\theta}) \tilde{V} A'(\tilde{\theta}) \lambda$$

where λ is the vector of Lagrange multipliers from the computation of the restricted estimate $\tilde{\theta}$.

- The likelihood ratio test statistic is

$$LR = 2 \left(L(\hat{\theta}) - L(\tilde{\theta}) \right)$$

where $\tilde{\theta}$ represents the constrained estimate of θ and L is the concentrated log-likelihood value.

For each type of test, under the null hypothesis the test statistic is asymptotically distributed as a χ^2 random variable with r degrees of freedom, where r is the number of expressions in the TEST statement. The p -values that are reported for the tests are computed from the $\chi^2(r)$ distribution and are only asymptotically valid.

Monte Carlo simulations suggest that the asymptotic distribution of the Wald test is a poorer approximation of its small sample distribution than that of the other two tests. However, the Wald test has the lowest computational cost, because it does not require computation of the constrained estimate $\tilde{\theta}$.

The following statements use the TEST statement to perform a likelihood ratio test:

```
proc cqlim;
  model y = x1 x2 x3;
  test x1 = 0, x2 * .5 + 2 * x3 = 0 /lr;
run;
```

For more information, see [SAS/ETS User's Guide](#).

Bayesian Analysis

To perform Bayesian analysis, you must specify a BAYES statement. Unless otherwise stated, all options in this section are options in the BAYES statement.

Bayes' theorem states that the posterior density $\pi(\theta|y)$ can be obtained via

$$\pi(\theta|y) \propto \pi(\theta)L(y|\theta)$$

where θ is a parameter or a vector of parameters and $\pi(\theta)$ is the product of the prior densities that you specify in the PRIOR statement. The term $L(y|\theta)$ is the likelihood associated with the model that is implied by the MODEL statement. For an introduction to Bayesian analysis, see the section “Overview” on page 5 in Chapter 2, “Introduction to Bayesian Analysis Procedures.”

To perform Bayesian analysis, you must draw a random sample from the posterior density $\pi(\theta|y)$ by using the Markov chain Monte Carlo (MCMC) method. For an introduction to MCMC, see the section “Markov Chain Monte Carlo Method” on page 16 in Chapter 2, “Introduction to Bayesian Analysis Procedures.” By default, PROC CQLIM uses the random walk Metropolis (RWM) algorithm to obtain the posterior sample. For an overview of Metropolis algorithms, see the section “Metropolis and Metropolis-Hastings Algorithms” on page 17 in Chapter 2, “Introduction to Bayesian Analysis Procedures.” The RWM algorithm requires a proposal distribution to be tuned to match certain features of the posterior distribution. For more information about this tuning process, see the section “Tuning Random Walk Metropolis Algorithms” on page 19 in Chapter 2, “Introduction to Bayesian Analysis Procedures.”

Alternatively, PROC CQLIM can use the No-U-Turn Sampler (NUTS) algorithm, a variant of the Hamiltonian Monte Carlo (HMC) algorithm, to obtain the posterior sample. For an overview of HMC and NUTS, see the sections “Hamiltonian Monte Carlo Sampler” on page 21 and “No-U-Turn Sampler” on page 22 in Chapter 2, “Introduction to Bayesian Analysis Procedures.” NUTS requires an integrator step size parameter and a mass matrix parameter to be tuned to match certain features of the posterior distribution. For more information about this tuning process, see the section “Tuning for NUTS” on page 23 in Chapter 2, “Introduction to Bayesian Analysis Procedures.”

Additional details specific to PROC CQLIM follow.

Tuning the Random Walk Metropolis Sampler

PROC CQLIM tunes the random walk Metropolis proposal distribution during a tuning phase that consists of n_{stage} tuning stages, each n_{tune} iterations long. For more information, see the section “Tuning Random Walk Metropolis Algorithms” on page 19 in Chapter 2, “Introduction to Bayesian Analysis Procedures.” The random walk Metropolis proposal covariance matrix that is defined in that section, COV^{cur} , should approximate the posterior covariance matrix and must be initialized prior to tuning. PROC CQLIM always initializes it to the covariance matrix of the maximum likelihood estimates of the model parameters.

Tuning the No-U-Turn Sampler

PROC CQLIM tunes both the integrator step size and the mass matrix in a tuning phase that consists of n_{tune} iterations, divided into a sequence of tuning windows. For more information, see the section “[Tuning for NUTS](#)” on page 23 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” The step size is tuned throughout the entire tuning phase in order to achieve a desired Metropolis acceptance rate. The mass matrix is diagonal, and the ideal mass matrix is the diagonal matrix of inverse posterior variances. These posterior variances are learned in the tuning phase. PROC CQLIM always initializes the mass matrix to the diagonal matrix of the inverse square standard errors of the maximum likelihood parameter estimates.

Initial Values of the Markov Chain

PROC CQLIM starts each Markov chain at the maximum likelihood estimates (MLEs) of the parameters. If the MLE for a particular parameter is outside the support of its prior distribution, then PROC CQLIM instead initializes that parameter near the boundary of its prior support.

Bayesian Analysis with Weighted Likelihoods

In PROC CQLIM, you can account for sample weights in Bayesian analysis by using the [WEIGHT](#) statement, provided that the weights are handled properly. The weighted log-likelihood functions in PROC CQLIM have the form

$$\ell(\theta) = \sum_{i=1}^n w_i \ell_i(\theta)$$

where $\ell_i(\theta)$ is the log likelihood for observation i , w_i is the corresponding sample weight, and n is the sample size. When $w_i = 1$ for all i , this is equivalent to a nonweighted log likelihood. In this case the weights sum to the sample size.

In the general weighted case, the sum of the weights should still equal the sample size—that is, $\sum_{i=1}^n w_i = n$ —and w_i/n should represent the proportion of the sample size that corresponds to observation i . This ensures that the prior density and likelihood function are weighted correctly in the posterior density. For more information, see Savitsky and Toth (2016).

PROC CQLIM normalizes the weights to sum to the sample size by default, so weighted Bayesian analyses are valid. The [NONNORMALIZE](#) option in the [WEIGHT](#) statement tells PROC CQLIM not to normalize the weights, so it can result in misleading weighted Bayesian analyses and is generally not recommended.

The [FREQ](#) statement is similar to the [WEIGHT](#) statement, except that each frequency represents the sample size associated with the corresponding observation. By definition, the sum of the frequencies is equal to the sample size. So you can combine the [BAYES](#) statement with the [FREQ](#) statement without issue.

Distributed Data and MCMC

When you use a CAS server that is running in massively parallel processing (MPP) mode, the data are distributed across the workers of the server. Iterative methods such as MCMC can be significantly slower in this context because of communication costs. Additionally, when you request multiple Markov chains by using the [NCHAIN=](#) option, those chains are independent of each other and can be run in parallel instead of serially. When you perform Bayesian analysis by using the [BAYES](#) statement on a server in MPP mode, PROC CQLIM automatically tries to take advantage of the distributed environment to make MCMC run as fast as possible. To do this, it automatically copies the data onto each worker machine and divides the

number of requested Markov chains among the workers as evenly as possible. Then those workers run their chains in parallel and send their results back to the controller. MCMC is therefore always as slow as the slowest worker—typically one of the workers that have the most Markov chains if the chains cannot be evenly divided among the workers. The fastest possible CAS server configuration has at least as many workers as the number of Markov chains that you request, though there is no benefit to having more workers than chains.

Although copying the data to each of the workers and letting them run their own Markov chains in parallel is fast, it requires the data set to be small enough to fit on each worker machine. You can use the **DISTRIBUTED** option to instead keep the data distributed at the cost of a much slower analysis. The **DISTRIBUTED** option also enables you to obtain results that are numerically identical to those of PROC CQLIM in release 2021.2.3 and previous releases, which do not automatically copy the data set onto the workers for Bayesian analysis. If the CAS server is running in symmetric multiprocessing (SMP) mode, then the server has only a single machine and the **DISTRIBUTED** option does nothing.

Prior Distributions

The **PRIOR** statement is used to specify the prior distribution of the model parameters. You must specify a list of parameters, a tilde (~), and then a distribution with its parameters. You can specify multiple **PRIOR** statements to define independent prior distributions. Parameters that are associated with a regressor variable are referred to by the name of the corresponding regressor variable. For information about how parameters are named so that you can refer to them when specifying prior distributions, see the section “[Naming of Parameters](#)” on page 614. Alternatively, the **SHOWNAMES** suboption of the **PRIORSUMMARY** option displays the names that are used in all parameters in the prior summary table.

If a parameter is not associated with a **PRIOR** statement, then the default choices shown in [Table 12.4](#) are used.

Table 12.4 Default Values for Prior Distributions

Parameter Type	Default Prior Name	Default Prior Specification
Regression, location	Normal	NORMAL(MEAN=0, VAR=1E6)
Scale, SD	Truncated normal	NORMAL(MEAN=0, VAR=1E6, LOWER=0, UPPER=INF)

For more information about each prior density, see the section “[Standard Prior Distributions](#)” on page 12 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” For information about truncated densities, see the section “[Truncated Prior Distributions](#)” on page 609. For information about why the default prior on the standard deviation parameter is a truncated normal distribution, see the section “[Prior Distributions for Scale Parameters](#)” on page 608.

Prior Distributions for Scale Parameters

A common choice for a prior distribution on variance parameters and other scale parameters is the inverse gamma distribution. This prior can often be problematic in practice, and as a consequence it can even cause computational problems. This is partially because the density puts essentially no probability mass near zero. For more information, see Gelman (2006). The main advantage of the inverse gamma prior is that it is the conditionally conjugate prior for a variance parameter in a normal model; this makes certain sampling algorithms easier to use. For more information, see the section “[Conjugate Priors](#)” on page 9 in Chapter 2,

“[Introduction to Bayesian Analysis Procedures](#).” However, the posterior sampler in PROC CQLIM does not take advantage of this conjugacy, so there is no computational benefit to using the inverse gamma prior.

Instead, PROC CQLIM uses the half-normal distribution for the default prior on standard deviation parameters—that is, a mean zero normal distribution truncated from below at zero. This density allows for the standard deviation parameter to be estimated arbitrarily close to zero and otherwise removes the problems associated with the inverse gamma prior. In general, if you want to specify prior information on the standard deviation parameter or some other scale parameter, it is recommended that you encode that information into a truncated normal distribution or perhaps a truncated t distribution.

If you want to specify the classic inverse gamma prior for a variance parameter, note that PROC CQLIM is parameterized in terms of the standard deviation parameter. So instead, the square root inverse gamma prior should be used on `_Sigma`, the standard deviation parameter in most models in PROC CQLIM.

Truncated Prior Distributions

All prior distributions except the uniform distribution can be turned into truncated prior distributions by using the LOWER and UPPER parameters. By default, LOWER is set to the maximum of the parameter’s lower bound and the specified prior distribution’s lower bound, and UPPER is set to the minimum of the parameter’s upper bound and the specified prior distribution’s upper bound. For example, for `_Sigma`, the standard deviation parameter in a regression model, the following statement specifies a truncated normal prior on `_Sigma` with LOWER=0 and UPPER=INFINITY:

```
prior _Sigma ~ normal(mean = 0, var = 1);
```

The density for of truncated prior can be computed from the corresponding untruncated density given in the section “[Standard Prior Distributions](#)” on page 12 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#),” as follows. Suppose $\pi(\theta)$ is the prior density for the parameter θ . Then let $\Pi(\theta)$ denote the corresponding CDF, defined by the integral

$$\Pi(\theta) = \int_{-\infty}^{\theta} \pi(x) dx$$

where $\pi(x)$ is taken to be 0 outside its range (for each prior density’s range, see the section “[Standard Prior Distributions](#)” on page 12 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#)”). Then the truncated density is constructed from the untruncated density by using the formula

$$\pi_{L,U}(\theta) = \frac{\pi(\theta)}{\Pi(U) - \Pi(L)}$$

where $L < U$ are the lower and upper truncation bounds, respectively. If the lower truncation bound is the lower bound of π ’s range, then $\Pi(L) = 0$, and similarly if the upper truncation bound is the upper bound of π ’s range, then $\Pi(U) = 1$.

Output to SAS Data Table

XBeta, Predicted, and Residual

XBeta is the structural part on the right-hand side of the model. The predicted value is the predicted dependent variable value. For censored variables, if the predicted value is outside the boundaries, it is reported as the closest boundary. For discrete variables, it is the level whose boundaries Xbeta falls between. The residual is defined only for continuous variables and is defined as

$$\text{Residual} = \text{Observed} - \text{Predicted}$$

Error Standard Deviation

The error standard deviation is σ_i in the model. It varies only when the HETERO statement is used.

Marginal Effects

A marginal effect is defined as a contribution of one control variable to the response variable. For a binary choice model with two response categories, $\mu_0 = -\infty$, $\mu_1 = 0$, and $\mu_2 = \infty$. For an ordinal response model with M response categories ($\mu_0, \dots, \mu_M$), define

$$R_{i,j} = \mu_j - \mathbf{x}_i' \boldsymbol{\beta}$$

The probability that the unobserved dependent variable is contained in the j th category can be written as

$$P[\mu_{j-1} < y_i^* \leq \mu_j] = F(R_{i,j}) - F(R_{i,j-1})$$

The marginal effect of changes in the regressors on the probability of $y_i = j$ is then

$$\frac{\partial \text{Prob}[y_i = j]}{\partial \mathbf{x}} = [f(\mu_{j-1} - \mathbf{x}_i' \boldsymbol{\beta}) - f(\mu_j - \mathbf{x}_i' \boldsymbol{\beta})] \boldsymbol{\beta}$$

where $f(x) = \frac{dF(x)}{dx}$. In particular,

$$f(x) = \frac{dF(x)}{dx} = \begin{cases} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} & (\text{probit}) \\ \frac{e^{-x}}{[1+e^{(-x)}]^2} & (\text{logit}) \end{cases}$$

The marginal effects in the truncated regression model are

$$\frac{\partial E[y_i | L_i < y_i^* < R_i]}{\partial \mathbf{x}} = \boldsymbol{\beta} \left[1 - \frac{(\phi(a_i) - \phi(b_i))^2}{(\Phi(b_i) - \Phi(a_i))^2} + \frac{a_i \phi(a_i) - b_i \phi(b_i)}{\Phi(b_i) - \Phi(a_i)} \right]$$

where $a_i = \frac{L_i - \mathbf{x}_i' \boldsymbol{\beta}}{\sigma_i}$ and $b_i = \frac{R_i - \mathbf{x}_i' \boldsymbol{\beta}}{\sigma_i}$.

The marginal effects in the censored regression model are

$$\frac{\partial E[y | \mathbf{x}_i]}{\partial \mathbf{x}} = \boldsymbol{\beta} \times \text{Prob}[L_i < y_i^* < R_i]$$

For models that involve higher-order or interaction variables, the marginal effects are calculated by treating these variables as independent explanatory variables. Therefore, the marginal effects of the higher-order or interaction variables ignore the polynomial nature of these variables.

Inverse Mills Ratio, Expected and Conditionally Expected Values

Expected and conditionally expected values are computed only for continuous variables. The inverse Mills ratio is computed for censored or truncated continuous, binary discrete, and selection endogenous variables.

Let L_i and R_i be the lower boundary and the upper boundary of y_i . Define $a_i = \frac{L_i - \mathbf{x}_i' \boldsymbol{\beta}}{\sigma_i}$ and $b_i = \frac{R_i - \mathbf{x}_i' \boldsymbol{\beta}}{\sigma_i}$. Then the inverse Mills ratio is defined as

$$\lambda = \frac{(\phi(a_i) - \phi(b_i))}{(\Phi(b_i) - \Phi(a_i))}$$

for a continuous variable and defined as

$$\lambda = \frac{\phi(\mathbf{x}_i' \boldsymbol{\beta})}{\Phi(\mathbf{x}_i' \boldsymbol{\beta})}$$

for a binary discrete variable.

The expected value is the unconditional expectation of the dependent variable. For a censored variable, it is

$$E[y_i] = \Phi(a_i)L_i + (\mathbf{x}_i' \boldsymbol{\beta} + \lambda\sigma_i)(\Phi(b_i) - \Phi(a_i)) + (1 - \Phi(b_i))R_i$$

For a left-censored variable ($R_i = \infty$), this formula is

$$E[y_i] = \Phi(a_i)L_i + (\mathbf{x}_i' \boldsymbol{\beta} + \lambda\sigma_i)(1 - \Phi(a_i))$$

where $\lambda = \frac{\phi(a_i)}{1 - \Phi(a_i)}$.

For a right-censored variable ($L_i = -\infty$), this formula is

$$E[y_i] = (\mathbf{x}_i' \boldsymbol{\beta} + \lambda\sigma_i)\Phi(b_i) + (1 - \Phi(b_i))R_i$$

where $\lambda = -\frac{\phi(b_i)}{\Phi(b_i)}$.

For a noncensored variable, this formula is

$$E[y_i] = \mathbf{x}_i' \boldsymbol{\beta}$$

The conditional expected value is the expectation given that the variable is inside the boundaries:

$$E[y_i | L_i < y_i < R_i] = \mathbf{x}_i' \boldsymbol{\beta} + \lambda\sigma_i$$

Probability

Probability applies only to discrete responses. It is the marginal probability that the discrete response is taking the value of the observation. If you specify the PROBALL option, then the probability for all possible responses of the discrete variables is computed.

Technical Efficiency

Technical efficiency for each producer is computed only for stochastic frontier models.

In general, the stochastic production frontier can be written as

$$y_i = f(x_i; \beta) \exp\{v_i\} TE_i$$

where y_i denotes producer i 's actual output, $f(\cdot)$ is the deterministic part of the production frontier, $\exp\{v_i\}$ is a producer-specific error term, and TE_i is the technical efficiency coefficient, which can be written as

$$TE_i = \frac{y_i}{f(x_i; \beta) \exp\{v_i\}}$$

For a Cobb-Douglas production function, $TE_i = \exp\{-u_i\}$. For more information, see the section “[Stochastic Frontier Production and Cost Models](#)” on page 600.

The cost frontier can be written in general as

$$E_i = c(y_i, w_i; \beta) \exp\{v_i\} / CE_i$$

where w_i denotes producer i 's input prices, $c(\cdot)$ is the deterministic part of the cost frontier, $\exp\{v_i\}$ is a producer-specific error term, and CE_i is the cost efficiency coefficient, which can be written as

$$CE_i = \frac{c(x_i, w_i; \beta) \exp\{v_i\}}{E_i}$$

For a Cobb-Douglas cost function, $CE_i = \exp\{-u_i\}$. For more information, see the section “[Stochastic Frontier Production and Cost Models](#)” on page 600. Hence, both technical and cost efficiency coefficients are the same. The estimates of technical efficiency are provided in the following subsections.

Normal-Half-Normal Model

Define $\mu_* = -\epsilon\sigma_u^2/\sigma^2$ and $\sigma_*^2 = \sigma_u^2\sigma_v^2/\sigma^2$. Then, as shown by Jondrow et al. (1982), conditional density is as follows:

$$f(u|\epsilon) = \frac{f(u, \epsilon)}{f(\epsilon)} = \frac{1}{\sqrt{2\pi}\sigma_*} \exp\left\{-\frac{(u - \mu_*)^2}{2\sigma_*^2}\right\} \Bigg/ \left[1 - \Phi\left(-\frac{\mu_*}{\sigma_*}\right)\right]$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\mu_*, \sigma_*^2)$.

From this result, it follows that the estimate of technical efficiency (Battese and Coelli 1988) is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \left[\frac{1 - \Phi(\sigma_* - \mu_{*i}/\sigma_*)}{1 - \Phi(-\mu_{*i}/\sigma_*)} \right] \exp\left\{-\mu_{*i} + \frac{1}{2}\sigma_*^2\right\}$$

The second version of the estimate (Jondrow et al. 1982) is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \mu_{*i} + \sigma_* \left[\frac{\phi(-\mu_{*i}/\sigma_*)}{1 - \Phi(-\mu_{*i}/\sigma_*)} \right] = \sigma_* \left[\frac{\phi(\epsilon_i \lambda / \sigma)}{1 - \Phi(\epsilon_i \lambda / \sigma)} - \left(\frac{\epsilon_i \lambda}{\sigma} \right) \right]$$

Normal-Exponential Model

Define $A = -\tilde{\mu}/\sigma_v$ and $\tilde{\mu} = -\epsilon - \sigma_v^2/\sigma_u$. Then, as shown by Kumbhakar and Lovell (2000), conditional density is as follows:

$$f(u|\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_v\Phi(-\tilde{\mu}/\sigma_v)} \exp\left\{-\frac{(u - \tilde{\mu})^2}{2\sigma^2}\right\}$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\tilde{\mu}, \sigma_v^2)$.

From this result, it follows that the estimate of technical efficiency is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \left[\frac{1 - \Phi(\sigma_v - \tilde{\mu}_i/\sigma_v)}{1 - \Phi(-\tilde{\mu}_i/\sigma_v)} \right] \exp\left\{-\tilde{\mu}_i + \frac{1}{2}\sigma_v^2\right\}$$

The second version of the estimate is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \tilde{\mu}_i + \sigma_v \left[\frac{\phi(-\tilde{\mu}_i/\sigma_v)}{1 - \Phi(-\tilde{\mu}_i/\sigma_v)} \right] = \sigma_v \left[\frac{\phi(A)}{\Phi(-A)} - A \right]$$

Normal-Truncated Normal Model

Define $\tilde{\mu} = (-\sigma_u^2\epsilon_i + \mu\sigma_v^2)/\sigma^2$ and $\sigma_*^2 = \sigma_u^2\sigma_v^2/\sigma^2$. Then, as shown by Kumbhakar and Lovell (2000), conditional density is as follows:

$$f(u|\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_*[1 - \Phi(-\tilde{\mu}/\sigma_*)]} \exp\left\{-\frac{(u - \tilde{\mu})^2}{2\sigma_*^2}\right\}$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\tilde{\mu}, \sigma_*^2)$.

From this result, it follows that the estimate of technical efficiency is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \frac{1 - \Phi(\sigma_* - \tilde{\mu}_i/\sigma_*)}{1 - \Phi(-\tilde{\mu}_i/\sigma_*)} \exp\left\{-\tilde{\mu}_i + \frac{1}{2}\sigma_*^2\right\}$$

The second version of the estimate is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \tilde{\mu}_i + \sigma_* \left[\frac{\phi(\tilde{\mu}_i/\sigma_*)}{1 - \Phi(-\tilde{\mu}_i/\sigma_*)} \right]$$

Naming

Naming of Parameters

When the estimation contains only one equation, the parameters are named in the same way as in other SAS procedures, such as the REG and PROBIT procedures. The constant in the regression equation is named Intercept. The coefficients of independent variables are named by the independent variables. The standard deviation of the errors is called `_Sigma`. The limits of the discrete dependent variable are named `_Limiti`. `_Limit1` is set to 0 and the limit parameters start from $i = 2$. If you also specify the HETERO statement, the coefficients of the independent variables in the HETERO statement are called `_H.x`, where x is the name of the independent variable.

When the estimation contains multiple equations, the parameters in the main equation are named in the format of `y.x`, where y is the name of the dependent variable and x is the name of the independent variable. The standard deviation of the errors is named `_Sigma.y`. The correlation of the errors is named `_Rho` for the bivariate model. For the model that contains three variables, it is `_Rho.y1.y2`, `_Rho.y1.y3`, and `_Rho.y2.y3`. The construction of correlation names for models that contain four or more variables is analogous.

Naming of Output Variables

Table 12.5 shows the *options* in the OUTPUT statement, along with the corresponding variable names and their explanations.

Table 12.5 OUTPUT Statement Options

<i>output-option</i>	Variable Name	Explanation
CONDITIONAL	CEXPCT_y	Conditional expected value of y , conditioned on the truncation
ERRSTD	ERRSTD_y	Standard deviation of error term
EXPECTED	EXPCT_y	Unconditional expected value of y
MARGINAL	MEFF_x	Marginal effect of x on y ($\frac{\partial y}{\partial x}$) with single equation
	MEFF_y_x	Marginal effect of x on y ($\frac{\partial y}{\partial x}$) with multiple equations
	MEFF_Pi_x	Marginal effect of x on y ($\frac{\partial \text{Prob}(y=i)}{\partial x}$) with single equation and discrete y
	MEFF_Pi_y_x	Marginal effect of x on y ($\frac{\partial \text{Prob}(y=i)}{\partial x}$) with multiple equations and discrete y
MILLS	MILLS_y	Inverse Mills ratio for y
PREDICTED	P_y	Predicted value of y
PROB	PROB_y	Probability that y is taking the observed value in this observation (discrete y only)
PROBALL	PROB_i_y	Probability that y is taking the i th value (discrete y only)
RESIDUAL	RESID_y	Residual of y , ($y - \text{PredictedY}$)
TE1	TE1	Technical efficiency estimate for each producer proposed by Battese and Coelli (1988)

Table 12.5 *continued*

<i>output-option</i>	Variable Name	Explanation
TE2	TE2	Technical efficiency estimate for each producer proposed by Jondrow et al. (1982)
XBETA	XBETA_y	Structure part ($\mathbf{x}'\boldsymbol{\beta}$) of y equation

ODS Table Names

PROC CQLIM assigns a name to each table that it creates. You can use these names to refer to the tables when you use the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in Table 12.6.

Table 12.6 ODS Tables Produced by PROC CQLIM

Table Name	Description	Statement	Option
ClassInfo	Level information for classification variables	CLASS	Default
Correlation	Correlation of parameter estimates	PROC CQLIM	CORRB
Covariance	Covariance of parameter estimates	PROC CQLIM	COVB
DiscreteRespProfile	Response profile	ENDOGENOUS or MODEL	DISCRETE
FitModelSummary	Summary of nonlinear estimation	MODEL	Default
GoodnessOfFit	Pseudo-R-square measures	MODEL	Default
InitialParameterEstimates	Optimization start	PROC CQLIM	ITPRINT
IterationHistory	Iteration history of nonlinear estimation	PROC CQLIM	ITPRINT
OptimizationResults	Optimization results	PROC CQLIM	ITPRINT
ParameterEstimates	Parameter estimates	MODEL	Default
SummaryContResponse	Summary of continuous response	MODEL	Default
TestResults	Hypothesis test statistics	TEST	Default
ODS Tables Created by the BAYES Statement			
AccRates	Metropolis acceptance rates	BAYES	SAMSUM
AutoCorr	Autocorrelation statistics for each parameter	BAYES	DIAG=AUTOCORR
AutoCorrByChain	Autocorrelation statistics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)=AUTOCORR
DiagSettings	Settings used for MCMC diagnostics	BAYES	DIAG(SETTINGS)=
DiagSummaries	ESS and MCSE diagnostics for each parameter	BAYES	Default

Table 12.6 *continued*

Table Name	Description	Statement	Option
DiagSummariesByChain	ESS and MCSE diagnostics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)=SUMMARIES
Geweke	Geweke diagnostics for each parameter	BAYES	DIAG=GEWEKE
GewekeByChain	Geweke diagnostics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)=GEWEKE
Heidelberger	Heidelberger-Welch diagnostics for each parameter	BAYES	DIAG=HEIDEL
HeidelbergerByChain	Heidelberger-Welch diagnostics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)=HEIDEL
NUTSDiagnostics	Diagnostics for the No-U-Turn Sampler	BAYES	SAMSUM
NUTSMassMatrixDiag	Mass matrix for the No-U-Turn Sampler	BAYES	SAMSUM(VERBOSE)
PostCorr	Posterior correlation matrix	BAYES	STATS=CORR
PostCorrByChain	Posterior correlation matrix for each Markov chain	BAYES	STATS(BYCHAIN)=CORR
PostCov	Posterior covariance matrix	BAYES	STATS=COV
PostCovByChain	Posterior covariance matrix for each Markov chain	BAYES	STATS(BYCHAIN)=COV
PostIntervals	Equal-tail and highest posterior density intervals for each parameter	BAYES	STATS=INTERVALS
PostIntervalsByChain	Equal-tail and highest posterior density intervals for each parameter in each Markov chain	BAYES	STATS(BYCHAIN)=INTERVALS
PostSummaries	Posterior mean, standard deviation, and percentiles for each parameter	BAYES	Default
PostSummariesByChain	Posterior mean, standard deviation, and percentiles for each parameter in each Markov chain	BAYES	STATS(BYCHAIN)=SUMMARIES
PriorSummary	Prior distribution for each parameter	BAYES	PRIORSUMMARY
Raftery	Raftery-Lewis diagnostics for each parameter	BAYES	DIAG=RAFTERY
RafteryByChain	Raftery-Lewis diagnostics for each parameter in each Markov chain	BAYES	DIAG(BYCHAIN)=RAFTERY
SamplerSettings	Full MCMC sampler specification	BAYES	SAMSET

Examples: The CQLIM Procedure

Example 12.1: Model with Censoring

This example uses the CQLIM procedure to process a large data table.

The following DATA step generates a data set that contains 5 million observations from a censored model. The model contains seven variables.

```
data simulate;
  call streaminit(12345);
  array vars x1-x7;
  array parms{7} (3 4 2 4 -3 -5 -3);
  intercept = 2;
  do i = 1 to 5000000;
    sum_xb = 0;
    do j = 1 to 7;
      vars[j] = rand('NORMAL', 0, 1);
      sum_xb = sum_xb + parms[j] * vars[j];
    end;
    y = intercept + sum_xb + 100 * rand('NORMAL', 0, 1);
    if y > 400 then y = 400;
    if y < 0 then y = 0;
    output;
  end;
  keep y x1-x7;
run;
```

You can load the data set simulate into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.simulate;
  set simulate;
run;
```

The following statements estimate a censored model:

```
proc cqlim data = mylib.simulate;
  model y = x1-x7 / censored(lb = 0 ub = 400);
run;
```

Output 12.1.1 shows the estimation results for the censored model. The “Model Fit Summary” table shows detailed information about the model. All parameter estimates in the “Parameter Estimates” table are highly significant and correspond to their theoretical values that were set during the data generating process.

Output 12.1.1 Censored Model: Summary

The CQLIM Procedure

Summary Statistics of Continuous Responses								
Variable	Mean	Standard Error	Type	Lower Bound	Upper Bound	N Obs		
						Lower Bound	Upper Bound	
y	41.11168	59.338269	Censored	0	400	246E4		189

Output 12.1.1 *continued*

Model Fit Summary	
Dependent Variable	y
Number of Observations	5000000
Data Set	SIMULATE
Log Likelihood	-1.702E7
Maximum Absolute Gradient	1.806E-9
Number of Iterations	7
Optimization Method	Newton-Raphson
AIC	34043697
SBC	34043817
Covariance Estimation	Hessian

Convergence criterion (FCONV=1E-11) satisfied.

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	2.045243	0.054689	37.40	<.0001
x1	1	3.021825	0.049348	61.23	<.0001
x2	1	4.006987	0.049345	81.20	<.0001
x3	1	1.955760	0.049327	39.65	<.0001
x4	1	4.045403	0.049338	81.99	<.0001
x5	1	-3.020339	0.049310	-61.25	<.0001
x6	1	-5.041650	0.049351	-102.16	<.0001
x7	1	-2.950510	0.049362	-59.77	<.0001
_Sigma	1	100.015057	0.049335	2027.25	<.0001

Example 12.2: Bivariate Probit Analysis

This example shows how to estimate a bivariate probit model by using a large data table.

The following DATA step generates a data set that contains 1 million observations from a bivariate probit model:

```
data bivariate_probit;
  call streaminit(19283);
  keep y1 y2 x1 x2;
  do i = 1 to 1000000;
    x1 = rand('NORMAL', 0, 1);
    x2 = rand('NORMAL', 0, 1);
    u1 = rand('NORMAL', 0, 1);
    u2 = rand('NORMAL', 0, 1);
    y1l = 1 + 2 * x1 + 3 * x2 + u1;
    y2l = 3 + 4 * x1 - 2 * x2 + u1 * 0.2 + u2;
    if y1l > 0 then y1 = 1;
    else y1 = 0;
    if y2l > 0 then y2 = 1;
    else y2 = 0;
    output;
  end;
run;
```

You can load the data set `bivariate_probit` into a data table in your session that is associated with the `mylib` libref. The DATA step assumes that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
data mylib.bivariate_probit;
    set bivariate_probit;
run;
```

The following statements estimate a bivariate probit model. Note that the INIT statement in the SAS program sets the initial values for some parameters in the optimization. The METHOD=QN option specifies the quasi-Newton optimization algorithm.

```
proc cqlim data = mylib.bivariate_probit method = qn;
    init y1.x1 = 2.8, y1.x2 = 2.1, _rho = 0.1;
    model y1 = x1 x2 / discrete;
    model y2 = x1 x2 / discrete;
run;
```

Output 12.2.1 shows the estimation results for the bivariate probit model. The “Discrete Response Profile” tables show the response profile for the variables y_1 and y_2 .

Output 12.2.1 Bivariate Probit Analysis Results

The CQLIM Procedure

Discrete Response Profile of y1		
Index	Value	Total Frequency
1	0	394734
2	1	605266

The CQLIM Procedure

Discrete Response Profile of y2		
Index	Value	Total Frequency
1	0	256241
2	1	743759

Model Fit Summary	
Dependent Variable	y1 y2
Number of Observations	1000000
Data Set	BIVARIATE_PROBIT
Log Likelihood	-313821
Maximum Absolute Gradient	0.058932
Number of Iterations	17
Optimization Method	Quasi-Newton
AIC	627656.7
SBC	627739.4
Covariance Estimation	Hessian

Convergence criterion (FCONV=1E-11) satisfied.

Output 12.2.1 *continued*

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
y1.Intercept	1	1.003233	0.003077	326.02	<.0001
y1.x1	1	2.004488	0.004805	417.20	<.0001
y1.x2	1	3.001878	0.006719	446.77	<.0001
y2.Intercept	1	2.944550	0.007710	381.92	<.0001
y2.x1	1	3.924242	0.010270	382.09	<.0001
y2.x2	1	-1.962626	0.005678	-345.65	<.0001
_Rho	1	0.194474	0.006278	30.98	<.0001

Example 12.3: Bayesian Analysis

This example uses the CQLIM procedure to process a small data table.

The following DATA step generates a data set that contains 1,000 observations from a censored model. The model contains eight variables.

```
data bayes_ex;
  call streaminit(12345);
  array vars x1-x7;
  array parms{7} (3 4 2 4 -3 -5 -3);
  intercept1 = 0;
  intercept2 = 4;
  intercept3 = 10;
  do i = 1 to 1000;
    sum_xb = 0;
    do j = 1 to 7;
      vars[j] = rand('NORMAL', 0, 1);
      sum_xb = sum_xb + parms[j] * vars[j];
    end;
    g = rand('NORMAL', 0, 1);
    if g < -0.5 then intercept = intercept1;
    if g >= -0.5 then intercept = intercept2;
    if g > 0.5 then intercept = intercept3;
    y = intercept + sum_xb + 10 * rand('NORMAL', 0, 1);
    if y > 400 then y = 400;
    if y < 0 then y = 0;
    if g < -0.5 then group = 0;
    if g >= -0.5 then group = 1;
    if g > 0.5 then group = 2;
    output;
  end;
  keep y x1-x7 group;
run;
```

You can load the data set bayes_ex into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.bayes_ex;
  set bayes_ex;
run;
```

The following statements estimate a censored model by using Bayesian methods. Note that the BAYES statement controls the settings for the MCMC sampler and determines which Bayesian output is produced. The NCHAIN= option controls the number of Markov chains, and the NSAMPLE= option controls the size of the posterior sample per chain. The PRIOR statements set the prior distributions for the parameters.

```
proc cqlim data = mylib.bayes_ex;
  class group;
  model y = x1-x7 group / censored(lb = 0 ub = 400);
  bayes priorsummary(shownames) seed = 72342 nsample = 10000 nchain = 4
        sampler = rwm(ntune = 100 ntunestage = 20);
  prior intercept ~ normal(mean = 0, sd = 100);
  prior x1-x5 ~ normal(mean = 0, sd = 10);
  prior x6 ~ t(loc = 0, scale = 10, df = 3, lower = 0);
  prior group_0 ~ normal(mean = -4, var = 100);
  prior _sigma ~ normal(mean = 0, sd = 100);
run;
```

Output 12.3.1 shows the Bayesian estimation results for the censored model. The “Prior Summary” table shows detailed information about prior distributions for the parameters in the model. The “Posterior Summaries” table contains two point estimates of the parameters from the posterior samples: the posterior mean and the posterior median (50th percentile). The “MCMC Diagnostics Summaries” table contains basic convergence diagnostics to check whether the Markov chain has converged and whether the sample size is sufficient.

Output 12.3.1 Bayesian Estimation of Censored Model

The CQLIM Procedure

Prior Summary									
			Bounds			Hyperparameters			
Parameter	Name	Prior	Lower	Upper	Name	Value	Name	Value	Name
Intercept	Intercept	Normal	-Infy	Infy	Mean	0	Variance	10000	
x1	x1	Normal	-Infy	Infy	Mean	0	Variance	100	
x2	x2	Normal	-Infy	Infy	Mean	0	Variance	100	
x3	x3	Normal	-Infy	Infy	Mean	0	Variance	100	
x4	x4	Normal	-Infy	Infy	Mean	0	Variance	100	
x5	x5	Normal	-Infy	Infy	Mean	0	Variance	100	
x6	x6	Truncated t	0	Infy	Location	0	Scale	10	DF 3
x7	x7	Normal	-Infy	Infy	Mean	0	Variance	1000000	
group 0	group_0	Normal	-Infy	Infy	Mean	-4	Variance	100	
group 1	group_1	Normal	-Infy	Infy	Mean	0	Variance	1000000	
_Sigma	_Sigma	Truncated Normal	0	Infy	Mean	0	Variance	10000	

Output 12.3.1 *continued*

The CQLIM Procedure

Posterior Summaries								
Parameter	N	Standard		Percentiles				
		Mean	Deviation	2.5%	25%	50%	75%	97.5%
Intercept	40000	10.2101	0.6881	8.8046	9.7512	10.2263	10.6837	11.5101
x1	40000	2.9478	0.3737	2.2276	2.7001	2.9466	3.1891	3.7152
x2	40000	3.6635	0.3549	2.9727	3.4211	3.6657	3.9017	4.3697
x3	40000	2.3640	0.3546	1.7168	2.1229	2.3415	2.5945	3.1004
x4	40000	3.7139	0.3440	3.0879	3.4800	3.6988	3.9391	4.4343
x5	40000	-3.1058	0.3693	-3.8217	-3.3589	-3.1012	-2.8553	-2.3860
x6	40000	0.0282	0.0281	0.000812	0.00822	0.0197	0.0390	0.1032
x7	40000	-2.4930	0.3636	-3.1820	-2.7399	-2.4991	-2.2469	-1.7793
group 0	40000	-9.4894	0.9844	-11.3458	-10.1500	-9.5118	-8.8705	-7.4648
group 1	40000	-7.4007	0.8598	-9.0624	-7.9876	-7.3987	-6.8169	-5.6735
_Sigma	40000	10.4933	0.2834	9.9736	10.2973	10.4818	10.6785	11.0791

The CQLIM Procedure

MCMC Diagnostic Summaries						
Parameter		MCSE	MCSE/SD	ESS	Autocorrelation	
					Time	ESS/N
Intercept	*	0.0539	0.0783	163.1	245.3	0.00408
x1	*	0.0240	0.0643	242.1	165.2	0.00605
x2	*	0.0206	0.0580	297.1	134.6	0.00743
x3	*	0.0190	0.0536	348.6	114.7	0.00872
x4	*	0.0215	0.0625	255.7	156.4	0.00639
x5	*	0.0211	0.0571	306.3	130.6	0.00766
x6	*	0.00128	0.0454	485.4	82.4062	0.0121
x7	*	0.0236	0.0650	236.3	169.2	0.00591
group 0	*	0.0660	0.0670	222.4	179.8	0.00556
group 1	*	0.0606	0.0705	201.4	198.6	0.00503
_Sigma	*	0.0162	0.0571	306.4	130.6	0.00766

*Autocorrelation Remains

Example 12.4: Marginal Effects Using Bayesian Analysis with NUTS

This example uses the CQLIM procedure to process a small data table.

The following DATA step generates a data set that contains 5,000 observations from a logistic regression model. The model contains seven variables.

```
data nuts_ex;
  call streaminit(784923);
  array vars x1-x7;
  array slopes{8} (0.03 0.04 0.02 0.04 -0.03 -0.05 -0.03 0.01);
  intercept = 2;

  do i = 1 to 5000;
    sum_xb = intercept;
    do j=1 to 7;
      vars[j] = rand('NORMAL',0,1);
      sum_xb = sum_xb + slopes[j] * vars[j];
    end;
    sum_xb = sum_xb + vars[1] * vars[1] * slopes[8];
    u = rand('UNIFORM');
    z = sum_xb + log(u) - log(1-u);
    if z > 0 then y = 1;
    if z <= 0 then y = 0;
    output;
  end;
  keep y x1-x7;
run;
```

You can load the data set nuts_ex into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.nuts_ex;
  set nuts_ex;
run;
```

The following statements estimate a logistic regression model in the Bayesian framework by using the No-U-Turn Sampler (NUTS). The model includes a second-order term for the covariate x1. The **NCHAIN=** option controls the number of Markov chains, and the **NSAMPLE=** option controls the size of the posterior sample per chain. NUTS is specified in the **SAMPLER=** option, and several values controlling the sampler can be set as arguments to **SAMPLER=NUTS**. The **OUTPOST=** option creates a CAS table that contains the posterior sample, which in this case is named nuts_post.

```
proc cqlim data = mylib.nuts_ex;
  model y = x1-x7 x1*x1 / discrete(distribution = logistic);
  bayes seed = 72342 nsample = 2000 nchain = 4 sampler = nuts(ntune = 2000)
    outpost = mylib.nuts_post;
  prior intercept ~ normal(mean = 0, sd = 1000);
  prior x1-x7 ~ normal(mean = 0, sd = 100);
  prior x1__x1 ~ normal(mean = 0, sd = 10);
run;
```

Output 12.4.1 shows the Bayesian estimation results for the logistic regression model. The “Posterior Summaries” table contains two point estimates of the parameters from the posterior samples: the posterior mean and the posterior median (50th percentile). The “MCMC Diagnostics Summaries” table contains basic convergence diagnostics to check whether the Markov chain has converged and whether the sample size is sufficient.

Output 12.4.1 Bayesian Estimation of Logistic Regression Model
The CQLIM Procedure

Posterior Summaries								
Parameter	N	Standard		Percentiles				
		Mean	Deviation	2.5%	25%	50%	75%	97.5%
Intercept	8000	2.0093	0.0558	1.8998	1.9722	2.0094	2.0467	2.1201
x1	8000	0.0142	0.0459	-0.0753	-0.0171	0.0141	0.0451	0.1047
x2	8000	0.00846	0.0442	-0.0793	-0.0210	0.00828	0.0381	0.0953
x3	8000	0.0258	0.0437	-0.0583	-0.00339	0.0259	0.0547	0.1118
x4	8000	0.0914	0.0444	0.00394	0.0611	0.0909	0.1216	0.1796
x5	8000	-0.1322	0.0449	-0.2211	-0.1629	-0.1325	-0.1011	-0.0452
x6	8000	-0.0451	0.0434	-0.1311	-0.0744	-0.0452	-0.0158	0.0395
x7	8000	-0.0684	0.0447	-0.1560	-0.0986	-0.0676	-0.0388	0.0191
x1 * x1	8000	0.0320	0.0338	-0.0329	0.00949	0.0312	0.0547	0.1004

The CQLIM Procedure

MCMC Diagnostic Summaries					
Parameter	MCSE	MCSE/SD	ESS	Autocorrelation	
				Time	ESS/N
Intercept	0.000643	0.0115	7525.1	1.0631	0.9406
x1	0.000449	0.00980	10422.5	0.7676	1.3028
x2	0.000432	0.00977	10474.5	0.7638	1.3093
x3	0.000444	0.0101	9719.2	0.8231	1.2149
x4	0.000447	0.0101	9866.8	0.8108	1.2333
x5	0.000434	0.00968	10681.7	0.7489	1.3352
x6	0.000436	0.0100	9904.7	0.8077	1.2381
x7	0.000436	0.00976	10495.2	0.7623	1.3119
x1 * x1	0.000389	0.0115	7560.8	1.0581	0.9451

The following sequence of statements computes the posterior sample of the sample mean marginal effects for each of the covariates in the model. In binary choice models, the marginal effect of the j th covariate on the i th observation is given by the derivative of i 's choice probability with respect to the j th covariate, evaluated at i 's value of that covariate. For logistic regression, this is given by

$$\frac{\partial}{\partial x_{ij}} P(y_i = 1) = \frac{\partial}{\partial x_{ij}} \frac{\exp(\mathbf{x}'_i \boldsymbol{\beta})}{1 + \exp(\mathbf{x}'_i \boldsymbol{\beta})} = \frac{\exp(\mathbf{x}'_i \boldsymbol{\beta})}{[1 + \exp(\mathbf{x}'_i \boldsymbol{\beta})]^2} \beta_j$$

The model includes a squared term for the covariate x1. Rewrite $\mathbf{x}'_i \boldsymbol{\beta}$ as $\mathbf{x}'_i \boldsymbol{\beta} + x_{i1}^2 \alpha$ to make the squared term explicit. Then the marginal effect of the covariate x1 is given by

$$\frac{\partial}{\partial x_{i1}} P(y_i = 1) = \frac{\partial}{\partial x_{i1}} \frac{\exp(\mathbf{x}'_i \boldsymbol{\beta} + x_{i1}^2 \alpha)}{1 + \exp(\mathbf{x}'_i \boldsymbol{\beta} + x_{i1}^2 \alpha)} = \frac{\exp(\mathbf{x}'_i \boldsymbol{\beta} + x_{i1}^2 \alpha)}{[1 + \exp(\mathbf{x}'_i \boldsymbol{\beta} + x_{i1}^2 \alpha)]^2} (\beta_j + 2x_{i1} \alpha)$$

The sample mean marginal effect is the mean of these individual marginal effects over all observations in the data set. These mean marginal effects must be computed for each iteration in the posterior sample to obtain the posterior distribution of the mean marginal effects.

The first set of statements computes the sample mean marginal effects for each iteration in the posterior distribution:

```

proc cas;
  fedSql.execDirect / query =
    "create table nuts_marginaleffects {options replace=true} as
    select avg(gradprob * (b_x1 + 2 * x1 * b_x1__x1)) as me_x1,
          avg(gradprob * b_x2) as me_x2,
          avg(gradprob * b_x3) as me_x3,
          avg(gradprob * b_x4) as me_x4,
          avg(gradprob * b_x5) as me_x5,
          avg(gradprob * b_x6) as me_x6,
          avg(gradprob * b_x7) as me_x7 from
    (
      select exp(xbeta) / (1 + exp(xbeta))**2 as gradprob, * from
      (
        select b.Intercept + a.x1 * b.x1 + a.x2 * b.x2 + a.x3 * b.x3 +
              a.x4 * b.x4 + a.x5 * b.x5 + a.x6 * b.x6 + a.x7 * b.x7 +
              a.x1 * a.x1 * b.x1__x1 as xbeta,
              a.*, b.Chain, b.Iteration, b.Intercept, b.x1__x1 as b_x1__x1,
              b.x1 as b_x1, b.x2 as b_x2, b.x3 as b_x3, b.x4 as b_x4,
              b.x5 as b_x5, b.x6 as b_x6, b.x7 as b_x7
        from nuts_ex as a, nuts_post as b
      ) as c
    ) as d
    group by Chain, Iteration;";
run; quit;

```

The last set of statements uses several actions to compute summary statistics of the posterior distribution of the mean marginal effects, including posterior means, standard deviations, and percentiles:

```

proc cas;
  simple.summary / table = {name = "nuts_marginaleffects"},
    inputs = {"me_x1", "me_x2", "me_x3", "me_x4", "me_x5", "me_x6", "me_x7"},
    subSet = {"N", "mean", "std"};
  casout = {name = "nuts_me_summary", replace = True};
  percentile.percentile / table = {name = "nuts_marginaleffects"},
    inputs = {"me_x1", "me_x2", "me_x3", "me_x4", "me_x5", "me_x6", "me_x7"},
    values = {2.5, 25, 50, 75, 97.5};
  casout = {name = "nuts_me_percentiles", replace = True};
  transpose.transpose / table = {name = "nuts_me_percentiles",
    groupby = {"_Column_"}}, transpose = {"_Value_"}, prefix = "Pct_",
    id = {"_Pctl_"}, casout = {name = "nuts_me_percentiles", replace = True};
  table.alterTable / name = "nuts_me_percentiles",
    drop = "_NAME_",
    columns = {{name = "_Column_", rename = "Parameter"},
      {name = "Pct_2.5", rename = "Pct_2_5",
        label = "2.5%", format = "D8."},
      {name = "Pct_25", label = "25%", format = "D8."},
      {name = "Pct_50", label = "50%", format = "D8."},
      {name = "Pct_75", label = "75%", format = "D8."},
      {name = "Pct_97.5", rename = "Pct_97_5",
        label = "97.5%", format = "D8."}};
  table.alterTable / name = "nuts_me_summary",
    columns = {{name = "_Column_", rename = "Parameter"},
      {name = "_NObs_", rename = "N"}},

```

```

        {name = "_Mean_", rename = "Mean", format = "D8."},
        {name = "_Std_", rename = "Std",
          label = "Standard Deviation", format = "D8."}};
fedSql.execDirect / query =
  "create table nuts_me_post_summary(options replace = true) as
  select a.*, b.Pct_2_5, b.Pct_25, b.Pct_50, b.Pct_75, b.Pct_97_5 from
  nuts_me_summary as a, nuts_me_percentiles as b
  where a.Parameter = b.Parameter";
table.fetch / table = {name = "nuts_me_post_summary"},
  index = False, sortby = "Parameter";
run;
quit;

```

Output 12.4.2 shows the posterior summary of the mean marginal effects.

Output 12.4.2 Posterior Summary of Mean Marginal Effects

Results from table.fetch

Selected Rows from Table NUTS_ME_POST_SUMMARY								
Parameter	N	Mean	Standard Deviation	2.5%	25%	50%	75%	97.5%
ME_X1	8000	0.00127	0.00450	-0.00749	-0.00179	0.00128	0.00428	0.0100
ME_X2	8000	0.000868	0.00455	-0.00817	-0.00217	0.000851	0.00391	0.00983
ME_X3	8000	0.00265	0.00449	-0.00603	-0.00035	0.00264	0.00559	0.0115
ME_X4	8000	0.00939	0.00457	0.000405	0.00630	0.00934	0.0124	0.0183
ME_X5	8000	-0.0136	0.00462	-0.0227	-0.0168	-0.0136	-0.0104	-0.00464
ME_X6	8000	-0.00464	0.00446	-0.0134	-0.00764	-0.00463	-0.00162	0.00401
ME_X7	8000	-0.00703	0.00459	-0.0162	-0.0101	-0.00696	-0.00399	0.00200

An advantage of the Bayesian framework is its ability to compute the posterior probability that a particular hypothesis is true. The following statements compute the posterior probability that each mean marginal effect is positive.

```

data mylib.nuts_me_probs;
  set mylib.nuts_marginaleffects;
  if me_x1 > 0 then x1 = 1; else x1 = 0;
  if me_x2 > 0 then x2 = 1; else x2 = 0;
  if me_x3 > 0 then x3 = 1; else x3 = 0;
  if me_x4 > 0 then x4 = 1; else x4 = 0;
  if me_x5 > 0 then x5 = 1; else x5 = 0;
  if me_x6 > 0 then x6 = 1; else x6 = 0;
  if me_x7 > 0 then x7 = 1; else x7 = 0;
  keep x1 x2 x3 x4 x5 x6 x7;
run;
quit;

proc cas;
  simple.summary /
    table = {name = "nuts_me_probs"},
    inputs = {"x1", "x2", "x3", "x4", "x5", "x6", "x7"},
    subSet = {"mean"},
    casout = {name = "nuts_me_post_probs", replace = True};
  table.alterTable / name = "nuts_me_post_probs",

```

```

columns = {{name = "_Mean_", rename = "Prob",
               label = "P (ME > 0)", format = "7.4"},
           {name = "_Column_", rename = "Parameter"}};
table.fetch / table = {name = "nuts_me_post_probs"},
index = False, sortby = "Parameter";
run;
quit;

```

Output 12.4.3 shows the posterior probability that each mean marginal effect is greater than zero.

Output 12.4.3 Posterior Probability of Positive Mean Marginal Effects

Results from table.fetch

Selected Rows from Table NUTS_ME_POST_PROBS	
Parameter	P(ME > 0)
x1	0.6124
x2	0.5763
x3	0.7240
x4	0.9801
x5	0.0009
x6	0.1523
x7	0.0621

Example 12.5: Bayesian Analysis with Multiple Chains

This example shows how to perform Bayesian analysis with multiple chains in the CQLIM procedure by using a small data table.

The following DATA step creates the Christenson Associates airline data, a frequently cited data set (Greene 2000). The data measure the costs, prices of inputs, and utilization rates for six airlines from 1970 to 1984. This example analyzes the log transformations of cost (C), quantity (Q), and price (PF) and the untransformed load factor (LF). These statements assume that your libref is named mylib, but you can substitute any appropriately defined libref.

```

data mylib.airline;
  input Obs AirlineID T C Q PF LF;
  Year = T + 1969;
  lC   = log(C);
  lQ   = log(Q);
  lPF  = log(PF);
  label lC   = "Log Transformation of Costs";
  label lQ   = "Log Transformation of Quantity";
  label lPF  = "Log Transformation of Price of Fuel";
  label LF   = "Load Factor (utilization index)";
datalines;
1   1   1   1140640   0.95276   106650   0.53449
2   1   2   1215690   0.98676   110307   0.53233
3   1   3   1309570   1.09198   110574   0.54774
4   1   4   1511530   1.17578   121974   0.54085
5   1   5   1676730   1.16017   196606   0.59117
... more lines ...

```

The following statements estimate a simple linear regression model in the Bayesian framework by using the No-U-Turn Sampler (NUTS). NUTS is specified in the **SAMPLER=** option, and several values controlling the sampler can be set as arguments to **SAMPLER=NUTS**. The **NCHAIN=** option controls the number of Markov chains, and the **NSAMPLE=** option controls the size of the posterior sample per chain. PROC CQLIM automatically aggregates the results of each of the chains to create the posterior statistics and MCMC diagnostics tables that you request. The **BYCHAIN** suboption of the **STATISTICS=** and **DIAGNOSTICS=** options additionally produces those tables separately for each chain. This example produces the “Posterior Summaries” table, the “MCMC Diagnostic Summaries” table, and the “Heidelberger-Welch Diagnostics” table for each chain as well as for all chains combined.

```
proc cqlim data = mylib.airline;
  model lC = lQ lPF lF;
  bayes seed = 2358 nsample = 2000 nchain = 4
    sampler = nuts(ntune = 2000)
    statistics(bychain) = summary
    diagnostics(bychain) = (summary heidel);
run;
```

Output 12.5.1 shows the Bayesian estimation results. The “Posterior Summaries” table contains the estimates of the posterior mean and posterior quantiles computed using all four chains. The other posterior summary statistics tables that are produced by the **STATISTICS=** option would similarly be computed using all four chains. The “MCMC Diagnostic Summaries” table contains basic convergence diagnostics aggregated across all four chains. The “Autocorrelation Diagnostics” table would also be aggregated for multiple chains. For more information about how this aggregation is performed, see the section “Using Multiple Chains” on page 37 in Chapter 2, “Introduction to Bayesian Analysis Procedures.” The “Heidelberger-Welch Diagnostics” table displays a summary of all the Heidelberger-Welch tests performed for each chain. Specifically, it shows the number of chains for which each test passed for each parameter, as well as the largest number of iterations that were discarded for the stationarity test. The “Geweke Diagnostics” table and the “Raftery-Lewis Diagnostics” table both would be summarized similarly.

Output 12.5.1 Posterior Summaries and MCMC Diagnostics

The CQLIM Procedure

Posterior Summaries								
Parameter	N	Mean	Standard Deviation	Percentiles				
				2.5%	25%	50%	75%	97.5%
Intercept	8000	9.5189	0.2369	9.0576	9.3602	9.5200	9.6780	9.9897
lQ	8000	0.8834	0.0136	0.8565	0.8743	0.8835	0.8924	0.9110
lPF	8000	0.4544	0.0206	0.4146	0.4397	0.4543	0.4685	0.4936
lF	8000	-1.6387	0.3551	-2.3521	-1.8720	-1.6356	-1.3933	-0.9782
_Sigma	8000	0.1262	0.00994	0.1085	0.1192	0.1255	0.1325	0.1466

Output 12.5.1 *continued***The CQLIM Procedure**

MCMC Diagnostic Summaries					
Parameter	MCSE	MCSE/SD	ESS	Autocorrelation	
				Time	ESS/N
Intercept	0.00634	0.0268	1394.9	5.7350	0.1744
IQ	0.000358	0.0263	1449.6	5.5187	0.1812
IPF	0.000536	0.0261	1469.3	5.4449	0.1837
LF	0.00865	0.0244	1686.1	4.7448	0.2108
_Sigma	0.000216	0.0218	2111.8	3.7882	0.2640

Heidelberger-Welch Diagnostics					
Parameter	Stationarity Test			Half-Width Test	
	Chains Tested	Tests Passed	Largest Iterations Discarded	Chains Tested	Tests Passed
Intercept	4	4	200	4	4
IQ	4	4	0	4	4
IPF	4	4	200	4	4
LF	4	4	0	4	4
_Sigma	4	4	0	4	4

Detailed results for each chain are available in [Output 12.5.2](#) as a result of the BYCHAIN options. The “Posterior Summaries by Chain” table contains the estimates of the posterior mean and posterior quantiles computed separately for each chain. Similarly, the “MCMC Diagnostic Summaries by Chain” table contains basic convergence diagnostics computed for each chain, and the “Heidelberger-Welch Diagnostics by Chain” table contains the Heidelberger-Welch diagnostics computed for each chain. For more information about the differences between the tables in [Output 12.5.1](#) and [Output 12.5.2](#), see the section “[Using Multiple Chains](#)” on page 37 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).”

Output 12.5.2 Posterior Summaries and MCMC Diagnostics by Chain
The CQLIM Procedure

Posterior Summaries by Chain									
Chain	Parameter	N	Standard		Percentiles				
			Mean	Deviation	2.5%	25%	50%	75%	97.5%
1	Intercept	2000	9.5273	0.2426	9.0619	9.3645	9.5192	9.6952	10.0305
	IQ	2000	0.8832	0.0132	0.8576	0.8753	0.8830	0.8919	0.9118
	IPF	2000	0.4524	0.0209	0.4128	0.4383	0.4525	0.4667	0.4925
	LF	2000	-1.6099	0.3489	-2.3108	-1.8336	-1.6070	-1.3769	-0.9698
	_Sigma	2000	0.1263	0.0107	0.1086	0.1185	0.1254	0.1334	0.1471
2	Intercept	2000	9.5156	0.2507	9.0271	9.3435	9.5209	9.6835	10.0036
	IQ	2000	0.8837	0.0147	0.8538	0.8735	0.8839	0.8932	0.9138
	IPF	2000	0.4545	0.0204	0.4128	0.4401	0.4543	0.4688	0.4917
	LF	2000	-1.6347	0.3751	-2.4184	-1.8903	-1.6307	-1.3722	-0.9120
	_Sigma	2000	0.1266	0.00978	0.1091	0.1195	0.1257	0.1324	0.1476
3	Intercept	2000	9.5163	0.2183	9.0936	9.3677	9.5193	9.6629	9.9285
	IQ	2000	0.8829	0.0136	0.8565	0.8731	0.8830	0.8927	0.9092
	IPF	2000	0.4552	0.0204	0.4168	0.4404	0.4553	0.4691	0.4944
	LF	2000	-1.6543	0.3499	-2.3549	-1.8776	-1.6564	-1.4140	-0.9975
	_Sigma	2000	0.1259	0.00976	0.1077	0.1191	0.1256	0.1325	0.1456
4	Intercept	2000	9.5165	0.2348	9.0438	9.3565	9.5211	9.6735	9.9813
	IQ	2000	0.8837	0.0129	0.8594	0.8753	0.8839	0.8917	0.9105
	IPF	2000	0.4553	0.0204	0.4163	0.4403	0.4550	0.4688	0.4949
	LF	2000	-1.6559	0.3440	-2.3441	-1.8875	-1.6461	-1.4118	-1.0155
	_Sigma	2000	0.1259	0.00951	0.1085	0.1195	0.1254	0.1320	0.1464

Output 12.5.2 *continued***The CQLIM Procedure**

MCMC Diagnostic Summaries by Chain						
Chain	Parameter	MCSE	MCSE/SD	ESS	Autocorrelation	
					Time	ESS/N
1	Intercept	0.0147	0.0604	274.1	7.2954	0.1371
	IQ	0.000784	0.0591	285.9	6.9958	0.1429
	IPF	0.00125	0.0598	279.8	7.1470	0.1399
	LF	0.0175	0.0502	397.1	5.0365	0.1986
	_Sigma	0.000527	0.0495	408.5	4.8960	0.2042
2	Intercept	0.0132	0.0528	359.1	5.5694	0.1796
	IQ	0.000730	0.0498	403.5	4.9562	0.2018
	IPF	0.00103	0.0505	392.8	5.0921	0.1964
	LF	0.0186	0.0495	408.5	4.8964	0.2042
	_Sigma	0.000434	0.0444	506.5	3.9486	0.2533
3	Intercept	0.0117	0.0535	349.4	5.7249	0.1747
	IQ	0.000697	0.0512	381.4	5.2442	0.1907
	IPF	0.000990	0.0486	423.4	4.7240	0.2117
	LF	0.0174	0.0498	403.6	4.9559	0.2018
	_Sigma	0.000380	0.0389	659.9	3.0308	0.3300
4	Intercept	0.0110	0.0466	459.7	4.3503	0.2299
	IQ	0.000638	0.0494	409.9	4.8787	0.2050
	IPF	0.00100	0.0491	415.2	4.8167	0.2076
	LF	0.0156	0.0452	488.9	4.0905	0.2445
	_Sigma	0.000385	0.0405	610.2	3.2776	0.3051

Output 12.5.2 continued

Heidelberger-Welch Diagnostics by Chain								
Chain	Parameter	Stationarity Test			Half-Width Test			
		Cramer-von Mises Stat	Test p-Value	Test Outcome	Iterations Discarded	Half-Width	Mean	Relative Half-Width Test Outcome
1	Intercept	0.2967	0.1381	Passed	200	0.0315	9.5184	0.00331 Passed
	IQ	0.0679	0.7645	Passed	0	0.00154	0.8832	0.00174 Passed
	IPF	0.3602	0.0923	Passed	200	0.00252	0.4533	0.00555 Passed
	LF	0.0853	0.6617	Passed	0	0.0362	-1.6099	-0.0225 Passed
	_Sigma	0.1054	0.5592	Passed	0	0.00103	0.1263	0.00818 Passed
2	Intercept	0.0629	0.7955	Passed	0	0.0284	9.5156	0.00299 Passed
	IQ	0.1841	0.3005	Passed	0	0.00149	0.8837	0.00168 Passed
	IPF	0.0904	0.6339	Passed	0	0.00215	0.4545	0.00473 Passed
	LF	0.0904	0.6339	Passed	0	0.0395	-1.6347	-0.0241 Passed
	_Sigma	0.1082	0.5459	Passed	0	0.000934	0.1266	0.00738 Passed
3	Intercept	0.0896	0.6384	Passed	0	0.0226	9.5163	0.00238 Passed
	IQ	0.1032	0.5692	Passed	0	0.00143	0.8829	0.00162 Passed
	IPF	0.1125	0.5272	Passed	0	0.00193	0.4552	0.00425 Passed
	LF	0.0255	0.9886	Passed	0	0.0357	-1.6543	-0.0216 Passed
	_Sigma	0.0666	0.7726	Passed	0	0.000817	0.1259	0.00649 Passed
4	Intercept	0.0374	0.9461	Passed	0	0.0228	9.5165	0.00239 Passed
	IQ	0.0968	0.6006	Passed	0	0.00132	0.8837	0.00150 Passed
	IPF	0.0896	0.6382	Passed	0	0.00210	0.4553	0.00461 Passed
	LF	0.2274	0.2201	Passed	0	0.0310	-1.6559	-0.0187 Passed
	_Sigma	0.0565	0.8357	Passed	0	0.000657	0.1259	0.00522 Passed

References

- Aigner, C., Lovell, C. A. K., and Schmidt, P. (1977). "Formulation and Estimation of Stochastic Frontier Production Function Models." *Journal of Econometrics* 6:21–37.
- Battese, G. E., and Coelli, T. J. (1988). "Prediction of Firm-Level Technical Efficiencies with a Generalized Frontier Production Function and Panel Data." *Journal of Econometrics* 38:387–399.
- Christensen, L. R., and Greene, W. H. (1976). "Economies of Scale in U.S. Electric Power Generation." *Journal of Political Economy* 84:655–676.
- Coelli, T. J., Prasada Rao, D. S., and Battese, G. E. (1998). *An Introduction to Efficiency and Productivity Analysis*. London: Kluwer Academic.
- Gallant, A. R. (1987). *Nonlinear Statistical Models*. New York: John Wiley & Sons.
- Gelman, A. (2006). "Prior Distributions for Variance Parameters in Hierarchical Models." *Bayesian Analysis* 1:515–533.
- Genz, A. (1992). "Numerical Computation of Multivariate Normal Probabilities." *Journal of Computational and Graphical Statistics* 1:141–150.
- Greene, W. H. (2000). *Econometric Analysis*. 4th ed. Upper Saddle River, NJ: Prentice-Hall.

- Gregory, A. W., and Veall, M. R. (1985). "On Formulating Wald Tests for Nonlinear Restrictions." *Econometrica* 53:1465–1468.
- Hajivassiliou, V. A., and McFadden, D. L. (1998). "The Method of Simulated Scores for the Estimation of LDV Models." *Econometrica* 66:863–896.
- Jondrow, J., Lovell, C. A. K., Materov, I. S., and Schmidt, P. (1982). "On the Estimation of Technical Efficiency in the Stochastic Frontier Production Function Model." *Journal of Econometrics* 19:233–238.
- Kumbhakar, S. C., and Lovell, C. A. K. (2000). *Stochastic Frontier Analysis*. New York: Cambridge University Press.
- McKelvey, R. D., and Zavoina, W. (1975). "A Statistical Model for the Analysis of Ordinal Level Dependent Variables." *Journal of Mathematical Sociology* 4:103–120.
- Meeusen, W., and van den Broeck, J. (1977). "Efficiency Estimation from Cobb-Douglas Production Functions with Composed Error." *International Economic Review* 18:435–444.
- Mroz, T. A. (1987). "The Sensitivity of an Empirical Model of Married Women's Work to Economic and Statistical Assumptions." *Econometrica* 55:765–799.
- Phillips, C. B., and Park, J. Y. (1988). "On Formulating Wald Tests of Nonlinear Restrictions." *Econometrica* 56:1065–1083.
- Savitsky, T. D., and Toth, D. (2016). "Bayesian Estimation under Informative Sampling." *Electronic Journal of Statistics* 10:1677–1708.
- Wooldridge, J. M. (2002). *Econometric Analysis of Cross Section and Panel Data*. Cambridge, MA: MIT Press.

Chapter 13

CSPATIALREG Procedure

Contents

Overview: CSPATIALREG Procedure	636
PROC CSPATIALREG Features	637
PROC CSPATIALREG Compared with the SPATIALREG Procedure	638
Using CAS Sessions and CAS Engine Librefs	638
Getting Started: CSPATIALREG Procedure	639
Syntax: CSPATIALREG Procedure	643
Functional Summary	644
PROC CSPATIALREG Statement	645
BOUNDS Statement	650
BY Statement	651
CLASS Statement	651
DISPLAY Statement	651
DISPLAYOUT Statement	652
INIT Statement	653
MODEL Statement	654
OUTPUT Statement	656
RESTRICT Statement	657
SPATIALEFFECTS Statement	658
SPATIALID Statement	658
TEST Statement	658
Details: CSPATIALREG Procedure	660
Missing Values	660
Spatial Autoregressive Models	660
Spatial Durbin Models	661
Spatial Error Models	662
Spatial Durbin Error Models	662
Spatial Moving Average Models	663
Spatial Durbin Moving Average Models	664
Spatial Autoregressive Moving Average Models	665
Spatial Durbin Autoregressive Moving Average Models	666
Spatial Autoregressive Confused Models	667
Spatial Durbin Autoregressive Confused Models	667
Linear Regression Models	668
Spatial Lag of X Models	669
Conditional Autoregressive Models	670
Specifying the Spatial Weights Matrix	671

Compact Representation of the Spatial Weights Matrix	672
Spatial ID Matching	674
Parameter Space of Spatial Coefficients	675
Approximations to the Jacobian	676
Impact Estimation	677
Multiple Model Comparison	679
Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements	680
Computational Resources	683
Covariance Matrix Types	684
Displayed Output	684
OUTPUT OUT= Data Table	686
ODS Table Names	686
Examples: CSPATIALREG Procedure	687
Example 13.1: Columbus Crime Data	687
Example 13.2: Simulated Data Example	696
Example 13.3: Compact Representation of a Spatial Weights Matrix	699
Example 13.4: Taylor and Chebyshev Approximations	702
Example 13.5: Impact Estimation	709
Example 13.6: Comparison of Multiple Models	712
References	715

Overview: CSPATIALREG Procedure

The CSPATIALREG (spatial regression) procedure, in SAS Viya, analyzes a class of linear spatial econometric models for cross-sectional data whose observations are spatially referenced or georeferenced. For example, housing price data that are collected from the 48 contiguous US states fall into the category of spatially referenced data. Compared to nonspatial regression models, spatial econometric models are capable of handling spatial interaction and spatial heterogeneity in a regression setting (Anselin 2001).

Spatial econometric models have been widely used in economics, political science, sociology, and other fields. For example, LeSage and Pace (2009) provide a detailed introduction to commonly used spatial econometric models from both frequentist and Bayesian perspectives. A brief introduction to spatial econometric models is also provided by Elhorst (2013).

PROC CSPATIALREG Features

The CSPATIALREG procedure fits spatial data regression models by maximum likelihood techniques.

PROC CSPATIALREG supports the following models:

- linear model
- linear model with spatial lag of X (SLX) effects
- spatial autoregressive (SAR) model
- spatial Durbin model (SDM)
- spatial error model (SEM)
- spatial Durbin error model (SDEM)
- spatial moving average (SMA) model
- spatial Durbin moving average (SDMA) model
- spatial autoregressive moving average (SARMA) model
- spatial Durbin autoregressive moving average (SDARMA) model
- spatial autoregressive confused (SAC) model
- spatial Durbin autoregressive confused (SDAC) model
- conditional autoregressive (CAR) model
- conditional autoregressive model with spatially lagged covariates (CAR2)

The following list summarizes some basic features of the CSPATIALREG procedure:

- uses model-building syntax that includes CLASS and effect-based MODEL statements that are familiar from other SAS Econometrics analytic procedures
- enables you to fit various models by using maximum likelihood estimation
- provides a **TEST** statement for performing linear tests of hypotheses
- provides a **RESTRICT** statement for imposing linear parameter constraints
- provides an **OUTPUT** statement for producing a data table that contains predictions and residuals
- enables you to fit multiple models, compare their fit statistics, and select the best fit by using criteria such as AIC and SBC

Because the CSPATIALREG procedure runs on SAS Cloud Analytic Services (CAS), it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode
- exploits all the available cores and concurrent threads. For information about how PROC CSPATIALREG uses threads, see the section “[Multithreading](#)” on page 108 in Chapter 4, “[Shared Concepts](#).”

PROC CSPATIALREG Compared with the SPATIALREG Procedure

The CSPATIALREG procedure is specifically designed to operate on SAS Viya and performs computations in multiple threads on multiple machines. In comparison, the SPATIALREG procedure executes in multiple threads on a single machine. The CSPATIALREG procedure models cross-sectional data in a way that is similar to how the SPATIALREG procedure in SAS/ETS software models them.

The CSPATIALREG procedure now supports all the models that are supported by PROC SPATIALREG and new conditional autoregressive (CAR) models that are not supported by PROC SPATIALREG. In addition, PROC CSPATIALREG supports impact estimation for the following models: CAR, linear, SAC, SAR, SARMA, SDAC, SDARMA, SDEM, SDM, SDMA, SEM, SMA, and SLX models. Furthermore, PROC CSPATIALREG supports the fitting and comparison of multiple models. PROC SPATIALREG does not provide any of this support.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the `CASHOST` and `CASPORT` of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the `TERMINATE` option in the CAS statement as follows:


```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: CSPATIALREG Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “[Using CAS Sessions and CAS Engine Librefs](#)” on page 54 and “[Loading a SAS Data Set onto a CAS Server](#)” on page 55 in Chapter 4, “[Shared Concepts](#).”

The CSPATIALREG procedure is similar to other SAS regression model procedures for nonspatial data, except that in PROC CSPATIALREG you usually need to provide a secondary data table (in the WMAT= option), which provides information about spatial weights matrix. The spatial weights matrix defines all pairwise spatial relationships and is a vital component of a spatial econometric model. For more information about how to create spatial weights matrix, see the section “[Specifying the Spatial Weights Matrix](#)” on page 671.

The following statements fit a SAR model:

```
proc cspatialreg data=mylib.One Wmat=mylib.W;
  model y = x1 x2 / type=SAR;
  spatialid sid;
run;
```

The response variable *y* (a continuous variable) and the explanatory variables *x1* and *x2* are contained in the data table *mylib.One*. The spatial weights matrix that you specify using the WMAT= option defines neighbor relationships among all spatial units in the data. The SPATIALID statement specifies a spatial ID variable that identifies spatial units in two data tables that you provide in the DATA= and WMAT= options. You specify the TYPE=SAR option in the MODEL statement to request a SAR model.

The following example illustrates PROC CSPATIALREG by using a real-world data set. The data set *CrimeOH* is taken from Anselin (1988). This data set contains variables such as *Income* (household income, measured in \$1,000), *HValue* (housing value by \$1,000), and *Crime* (number of crimes, including residential burglaries and vehicle thefts, measured per 1,000 households) in 49 neighborhoods in Columbus, Ohio. You want to examine how household income and housing value affect the number of crimes in these neighborhoods.

The first 10 observations in the *mylib.CrimeOH* data set are shown in [Figure 13.1](#).

Figure 13.1 Columbus Crime Data

Obs	crime	income	hvalue	lat	lon	sid
1	18.802	21.232	44.567	35.62	42.38	s1
2	0.178	8.438	75.000	33.36	38.41	s4
3	50.732	11.252	23.225	40.01	38.00	s7
4	34.001	13.598	96.400	47.61	36.42	s10
5	19.146	18.942	40.300	50.11	29.91	s13
6	16.241	29.833	61.950	48.44	27.93	s16
7	33.705	11.709	30.450	43.37	33.46	s19
8	41.968	9.918	23.600	44.10	30.40	s22
9	25.962	16.961	33.500	43.23	27.31	s25
10	29.028	14.135	27.733	39.32	25.85	s28

The following SAS statements fit a linear regression model to the CrimeOH data set:

```

/*-- Linear Model --*/
proc cspatialreg data=mylib.CrimeOH;
    model crime = income hvalue / type=LINEAR;
run;

```

The “Model Fit Summary” table, shown in [Figure 13.2](#), lists several fit summary statistics for the model. By default, the CSPATIALREG procedure uses the Newton-Raphson optimization technique. The maximum log-likelihood value is shown, in addition to two information measures, Akaike’s information criterion (AIC) and Schwarz’s Bayesian information criterion (SBC). AIC or SBC can be used for model selection. For a set of candidate models, the model that has the smallest AIC or SBC is often preferred. In addition, the method that is used for covariance estimation is shown.

Figure 13.2 Fit Summary Statistics for a Linear Model

The CSPATIALREG Procedure

Model Fit Summary	
Dependent Variable	crime
Number of Observations	49
Data Set	CRIMEOH
Model	LINEAR
Log Likelihood	-187.377
Maximum Absolute Gradient	7.599E-7
Number of Iterations	16
Optimization Method	Newton-Raphson
AIC	382.7542
SBC	390.3215
Covariance Estimation	Hessian

The parameter estimates of the model and their standard errors are shown in [Figure 13.3](#). Based on the p -values, both Income and HValue are significant at the 0.05 level.

Figure 13.3 Parameter Estimates of the Linear Model

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	68.618863	4.588210	14.96	<.0001
income	1	-1.597304	0.323739	-4.93	<.0001
hvalue	1	-0.273931	0.099989	-2.74	0.0062
_sigma2	1	122.751696	24.799493	4.95	<.0001

The following statements fit a SAR model to the CrimeOH data set:

```

/*-- SAR Model --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat NONORMALIZE;
    model crime = income hvalue / type=SAR;
    spatialid sid;
run;

```

The NONORMALIZE option requests that the spatial weights matrix that is specified in the mylib.CrimeWmat data table be used “as is” rather than be row-standardized. The “Model Fit Summary” table, shown in [Figure 13.4](#), lists several fit summary statistics for the SAR model. For this model, the value of AIC is about 374.78—smaller than 382.75, which is the AIC value for the preceding linear model. Based on AIC, the SAR model is preferred.

Figure 13.4 Fit Summary Statistics for the SAR Model

The CSPATIALREG Procedure

Model Fit Summary	
Dependent Variable	crime
Number of Observations	49
Data Set	CRIMEOH
Spatial Weights	CRIMEWMAT
Model	SAR
Spatial ID Variable	sid
Log Likelihood	-182.36
Maximum Absolute Gradient	0.000192
Number of Iterations	16
Optimization Method	Newton-Raphson
AIC	374.7207
SBC	384.1798
Covariance Estimation	Hessian

The parameter estimates of the SAR model and their standard errors are shown in [Figure 13.5](#). According to the p -values, both Income and HValue are significant at the 0.05 level. In addition, the spatial autoregressive coefficient ρ is estimated to be about 0.431, with a p -value of 0.0005.

Figure 13.5 Parameter Estimates of the SAR Model

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	45.076368	7.870002	5.73	<.0001
income	1	-1.031514	0.328375	-3.14	0.0017
hvalue	1	-0.265923	0.088211	-3.01	0.0026
_rho	1	0.431032	0.123586	3.49	0.0005
_sigma2	1	95.470316	19.499458	4.90	<.0001

Unlike the previous SAR model, a spatial Durbin model (SDM) accounts for exogenous interaction effects by introducing additional spatially lagged regressors into the model. To fit an SDM, you need to combine the MODEL statement (with the TYPE=SAR option) and the SPATIALEFFECTS statement. The following statements fit an SDM :

```

/*-- SDM --*/
proc cspatialreg data=mylib.CrimeOH wmat=mylib.CrimeWmat NONORMALIZE;
    model crime = income hvalue / type=SAR;
    spatialeffects income hvalue;
    spatialid sid;
run;

```

The fit summary statistics for the SDM are shown in [Figure 13.6](#). Parameter estimates are provided in [Figure 13.7](#).

Figure 13.6 Fit Summary Statistics for the SDM

The CSPATIALREG Procedure	
Model Fit Summary	
Dependent Variable	crime
Number of Observations	49
Data Set	CRIMEOH
Spatial Weights	CRIMEWMAT
Model	SDM
Spatial ID Variable	sid
Log Likelihood	-181.363
Maximum Absolute Gradient	0.000211
Number of Iterations	16
Optimization Method	Newton-Raphson
AIC	376.7269
SBC	389.9696
Covariance Estimation	Hessian

Figure 13.7 Parameter Estimates for the SDM

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	42.802410	13.923873	3.07	0.0021
income	1	-0.914200	0.336418	-2.72	0.0066
hvalue	1	-0.293746	0.088851	-3.31	0.0009
W_income	1	-0.519612	0.594739	-0.87	0.3823
W_hvalue	1	0.245718	0.176843	1.39	0.1647
_rho	1	0.426506	0.167485	2.55	0.0109
_sigma2	1	91.768262	18.904590	4.85	<.0001

The spatial autoregressive coefficient ρ is estimated to be 0.426 with a p -value of 0.0109 based on an asymptotic t test. This result seems to suggest that there is a significantly positive spatial dependence in the number of crimes.

In the CSPATIALREG procedure, the null hypothesis $H_0: \rho = 0$ can also be tested against the alternative $H_a: \rho \neq 0$ by using the likelihood ratio (LR) test, Lagrange multiplier (LM) test, and Wald test. For the LR test, the test statistic is equal to $-2(\mathcal{L}_{\text{linear}} - \mathcal{L}_{\text{SAR}}) = -2(-187.38 + 182.39) = 9.98$, where $\mathcal{L}_{\text{linear}}$ and $\mathcal{L}_{\text{SAR}}$ are the log likelihoods for the linear regression model and SAR model, respectively. The likelihood ratio test is significant at the 0.05 level, providing strong evidence of spatial dependence in the data.

Syntax: CSPATIALREG Procedure

The following statements are available in the CSPATIALREG procedure. Items within angle brackets (<>) or square brackets ([]) are optional.

```

PROC CSPATIALREG <options>;
  BOUNDS bound1 [ , bound2 ... ] ;
  BY variables ;
  CLASS variable <(options)> ... <variable <(options)>> </global-options>;
  DISPLAY <table-list> </options>;
  DISPLAYOUT table-spec-list </options>;
  INIT initialization1 < , initialization2 ... >;
  MODEL dependent-variable = regressors </options>;
  OUTPUT OUT=libref.data-table <options> <keyword <=name>> ... <keyword <=name>>>;
  RESTRICT restriction1 [ , restriction2 ... ] ;
  TEST equation1 < , equation2 ... > </test-options>;
  SPATIALEFFECTS <spatial-effect-regressors>;
  SPATIALID variable ;

```

You can specify only one MODEL statement. If you specify a BOUNDS, INIT, OUTPUT, RESTRICT, SPATIALEFFECTS, or TEST statement, it must appear after the MODEL statement. You can specify no more than one SPATIALEFFECTS statement. However, multiple BOUNDS, INIT, RESTRICT, and TEST statements are allowed.

Functional Summary

Table 13.1 summarizes the statements and options available in the CSPATIALREG procedure. The sections that follow provide detailed syntax information about each of these statements, beginning with the PROC CSPATIALREG statement. The remaining statements are covered in alphabetical order.

Table 13.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CSPATIALREG	DATA=
Specifies the spatial weights input data table	PROC CSPATIALREG	WMAT=
Suppresses normalization of the spatial weights	PROC CSPATIALREG	NONNORMALIZE
Writes estimates of $\mathbf{x}_i'\boldsymbol{\beta}$, predicted values, and residuals to an output data table	OUTPUT	OUT=
Approximation Control Options		
Specifies the approximation-related options	PROC CSPATIALREG	APPROXIMATION=
Declaring the Role of Variables		
Specifies BY-group processing	BY	
Specifies classification variables	CLASS	
Specifies a spatial ID variable	SPATIALID	
Impact Estimation Control Option		
Specifies options that are related to impact estimation	PROC CSPATIALREG	IMPACT
Printing Control Options		
Prints the correlation matrix of the estimates	MODEL PROC CSPATIALREG	CORRB CORRB
Prints the covariance matrix of the estimates	MODEL PROC CSPATIALREG	COVB COVB
Suppresses the normal printed output	MODEL PROC CSPATIALREG	NOPRINT NOPRINT
Prints all available output	MODEL PROC CSPATIALREG	PRINTALL PRINTALL
Prints timing information	PROC CSPATIALREG	PRINTTIMING
Prints the names used internally for the parameters	PROC CSPATIALREG	PRINTINTERNALNAMES
Options to Control the Optimization Process		
Specifies the maximum number of iterations	MODEL PROC CSPATIALREG	MAXITER= MAXITER=
Selects the iterative minimization method to use	MODEL PROC CSPATIALREG	METHOD= METHOD=

Table 13.1 *continued*

Description	Statement	Option
Specifies the maximum number of function calls	MODEL PROC CSPATIALREG	MAXFUNC=
Specifies the upper limit of CPU time in seconds	MODEL PROC CSPATIALREG	MAXTIME=
Sets boundary restrictions on parameters	BOUNDS	
Sets initial values of parameters	INIT	
Sets linear restrictions on parameters	RESTRICT	
Model Estimation Options		
Specifies the spatial lag of covariate effect	SPATIALEFFECTS	
Specifies the type of model	MODEL PROC CSPATIALREG	TYPE=
Specifies the type of covariance matrix	MODEL	COVEST=
Suppresses the intercept parameter	MODEL	NOINT
Output Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	
Outputs SAS variables to the output data table	OUTPUT	COPYVAR=
Outputs the residual	OUTPUT	RESID=
Outputs the expected value of the response variable	OUTPUT	PRED=
Outputs estimates of $\mathbf{x}'_i \boldsymbol{\beta}$	OUTPUT	XBETA=

PROC CSPATIALREG Statement

PROC CSPATIALREG *options* ;

You can specify the following *options* in the PROC CSPATIALREG statement.

Data Table Options

You must specify the following *option*:

DATA=*libref.data-table*

names the input data table for PROC CSPATIALREG to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 638.

data-table specifies the name of the input data table.

This is the primary input data table, which contains dependent variables, explanatory variables, and so on.

For all models except a purely linear model, you must also specify the following *option*:

WMAT=libref.data-table

specifies the secondary spatial weights data table, which you can use to construct the spatial weights matrix **W**. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 638.

Loosely speaking, the entries of **W**, $w(s_i, s_j)$, define the amount of influence that a unit s_j has over a unit s_i . The entries $w(s_i, s_j)$ must be nonnegative and have zeros on the diagonal; that is, $w(s_i, s_j) \geq 0$ and $w(s_i, s_i) = 0$, where i and $j = 1, 2, \dots, n$ and n is the total number of spatial units in the data. Any nonzero diagonal elements $w(s_i, s_i)$ are replaced with 0. The spatial weights matrix can be asymmetric; that is, it is not necessary that $w(s_i, s_j) = w(s_j, s_i)$. For information about missing spatial weights in **W**, see the **NONORMALIZE** option.

The **W** matrix can take two different forms:

- You can provide a full spatial weights matrix. In this case, the data table that you specify in the **WMAT=** option has n rows and $n + 1$ columns and must have a column for spatial ID variable.
- You can specify the spatial weights matrix by using a compact form when appropriate. In this form, the number of observations in the data table that you specify in the **WMAT=** option should match the number of nonzero elements in the spatial weights matrix. Moreover, the number of columns in this data table should be three. The first two columns contain the row and column indices for nonzero entries in the spatial weights matrix, and the third column contains the nonzero entries in the spatial weights matrix. If you use the compact form for the spatial weights matrix, you must include a **SPATIALID** statement to match observations in the data tables that you specify in the **DATA=** and **WMAT=** options. For more information about the **SPATIALID** statement, see the section “**SPATIALID Statement**” on page 658. For more information about the compact representation of the spatial weights matrix, see the section “**Compact Representation of the Spatial Weights Matrix**” on page 672.

For all models except a purely linear model, you can also specify the following *option*:

NONORMALIZE

suppresses the row standardization of the spatial weights matrix that you specify in the **WMAT=** option. By default, the spatial weights matrix is row-standardized; that is, the spatial weights matrix has unit row sum. If you specify the **NONORMALIZE** option, spatial weights are used “as is” except for $w(s_i, s_i)$, which is always treated as 0. This implies that an entry $w(s_i, s_j)$ in the **W** matrix cannot be missing for $i \neq j$ if you specify this option. If you do not specify this option, missing spatial weights are replaced with zeros.

Approximation Control Option

For the SAR, SDM, SEM, SDEM, and CAR models, you can specify the following *options*:

APPROXIMATION=(*approx-options*)

specifies options that are related to approximating the Jacobian, as described in the section “[Approximations to the Jacobian](#)” on page 676. To invoke approximation, you must specify one or more of the following *approx-options*:

CHEBYSHEV | TAYLOR

specifies the approximation method. By default, Chebyshev approximation is used. The Taylor approximation is used only if you specify the TAYLOR option.

NMC=*number*

specifies a positive integer as the number of standard random normal draws for Monte Carlo simulation. By default, NMC=100.

ORDER=*number*

specifies a positive integer as the order of series in Taylor or Chebyshev approximation. If you specify Taylor approximation, ORDER=50 by default. If you specify Chebyshev approximation, ORDER=5 by default.

SEED=*number*

specifies an integer seed in the range 1 to $2^{31} - 1$ for the random number generator that is used for Monte Carlo simulation. Specifying a seed enables you to reproduce your analysis. By default, SEED=1.

Impact Estimation Control Option

For impact estimation, you can specify the following *options*:

IMPACT(*impacttest-option*)

specifies options that are related to impact estimation, as described in the section “[Impact Estimation](#)” on page 677. To invoke impact estimation, you must specify one or more of the following *impacttest-options*:

NMC=*number*

specifies a positive integer as the number of iterations for Monte Carlo simulation. By default, NMC=1000.

ORDER=*number*

specifies a positive integer as the highest order of the Neumann series for approximation. By default, ORDER=50.

SEED=*number*

specifies an integer seed in the range 1 to $2^{31} - 1$ for the random number generator that is used for Monte Carlo simulation. Specifying a seed enables you to reproduce your analysis. By default, SEED=1.

You can use the IMPACT option to summarize the average direct impacts, the average indirect impacts, and the average total impacts of explanatory variables in the model.

Printing Options

You can specify the following options in either the PROC CSPATIALREG statement or the MODEL statement:

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

NOPRINT

suppresses all printed output.

PRINTALL

requests all printing options.

PRINTINTERNALNAMES

prints internal names that are assigned to parameters.

PRINTTIMING

prints a timing report.

Estimation Control Options

You can specify the following options in either the PROC CSPATIALREG statement or the MODEL statement:

COVEST=HESSIAN | OP | QML

specifies the type of covariance matrix for the parameter estimates. You can specify the following types:

HESSIAN

specifies the covariance from the Hessian matrix.

OP

specifies the covariance from the outer product matrix.

QML

specifies the covariance from the outer product and Hessian matrices.

By default, COVEST=HESSIAN. The quasi-maximum-likelihood estimates are computed using COVEST=QML. For all models except the linear and SLX models, only COVEST=HESSIAN is supported.

TYPE=ALL | AUTO | CAR | LINEAR | SAC | SAR | SARMA | SEM | SMA

specifies the type of model to be fit. You can specify the following values:

ALL

fits all models.

AUTO

fits a subset of selected models.

CAR

fits a conditional autoregressive model.

LINEAR

fits a linear model.

SAC

fits a spatial autoregressive confused model.

SAR

fits a spatial autoregressive model.

SARMA	fits a spatial autoregressive moving average model.
SEM	fits a spatial error model.
SMA	fits a spatial moving average model.

By default, TYPE=SAR. For more information about the TYPE=ALL or TYPE=AUTO option, see the section “[Multiple Model Comparison](#)” on page 679.

If you specify covariates in the SPATIALEFFECT statement, then their spatial lag is added to the model. In that case, you can specify the following TYPE= option values to get the model that you want:

TYPE=CAR	fits the CAR2 model.
TYPE=LINEAR	fits the SLX model.
TYPE=SAC	fits the SDAC model.
TYPE=SAR	fits the SDM.
TYPE=SARMA	fits the SDARMA model.
TYPE=SEM	fits the SDEM.
TYPE=SMA	fits the SDMA model.

Optimization Control Options

PROC CSPATIALREG uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. You can specify the following *options* in either the PROC CSPATIALREG statement or the MODEL statement:

MAXFUNC=*i*

MAXFU=*i*

specifies the maximum number of function calls in the optimization process. By default, MAXFUNC=1000.

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually used can exceed the number of calls that you specify in this option.

MAXITER=*i*

MAXIT=*i*

specifies the maximum number of iterations in the optimization process. By default, MAXITER=200.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. The default value is the largest floating-point double representation available on your computer. The time that you specify in this option is checked only once at the end of each iteration. Therefore, the actual run time can be much longer than *r*. The actual run time includes the remaining time that is needed to finish the iteration and the time that is needed to generate the output of the results.

METHOD=CONGRA | DBLDOG | NEWRAP | NONE | NRRIDG | QUANEW | TRUREG

specifies the iterative minimization method to use. You can specify the following methods:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double-dogleg method.
NEWRAP	specifies the Newton-Raphson method.
NONE	specifies that optimization not be performed.
NRRIDG	specifies the Newton-Raphson ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

By default, METHOD=NEWRAP.

BOUNDS Statement

BOUNDS *bound1* [, *bound2* ...] ;

The BOUNDS statement imposes simple boundary constraints on the parameter estimates. You can specify any number of BOUNDS statements.

Each *bound* is composed of parameter names, constants, and inequality operators as follows:

item operator item [*operator item* [*operator item* ...]]

Each *item* is a constant, a parameter name, or a list of parameter names. Each *operator* is <, >, <=, or >=. Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the BOUNDS statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

You can use both the BOUNDS statement and the RESTRICT statement to impose boundary constraints. However, the BOUNDS statement provides a simpler syntax for specifying these kinds of constraints. For more information about the RESTRICT statement, see the section “[RESTRICT Statement](#)” on page 657.

The following BOUNDS statement constrains the estimates of the parameter for *z* to be negative, the parameters for *x1* through *x10* to be between 0 and 1, and the parameter for the spatial lag of *x1* to be less than 1:

```
bounds z < 0,
       0 < x1-x10 < 1,
       W_x1 < 1;
```

BY Statement

BY *variables* ;

You can use a BY statement in PROC CSPATIALREG to obtain separate analyses of observations in groups that are defined by the BY variables. When you use a BY statement, the primary input data table (specified in the DATA= option) should be sorted by the BY variables.

BY-statement processing is not supported when the CSPATIALREG procedure runs alongside the database or alongside the Hadoop Distributed File System (HDFS). These modes are used if the input data are stored in a database or HDFS and the grid host is the appliance that houses the data.

CLASS Statement

CLASS *variable* < (*options*) > . . . < *variable* < (*options*) > > < / *global-options* > ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. You can list the response variable for binary models in the CLASS statement, but this is not required. [Table 13.2](#) summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “[CLASS Statement](#)” on page 56 in Chapter 4, “[Shared Concepts](#).”

Table 13.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable
SPLIT	Splits levels of CLASS variables into independent effects

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE

statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 686. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

- | | |
|------------------|---|
| <i>key=value</i> | specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the output table name. |
| <i>key</i> | specifies <i>key</i> as the ODS table name and also as the output table name. |

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 686. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

INIT Statement

INIT *initialization1* < , *initialization2* ... > ;

The INIT statement sets initial values for parameters in the optimization.

Each *initialization* is written as a parameter or parameter list, followed by an optional equal sign (=), followed by a number:

parameter <=> *number*

Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

For more information about how parameters are named in the INIT statement, see the section “[Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 680.

MODEL Statement

MODEL *dependent-variable* = *regressors* </ options> ;

The MODEL statement specifies the *dependent-variable* and independent covariates (*regressors*) for the regression model. If you specify no *regressors*, PROC CSPATIALREG fits a model that contains only an intercept. The *dependent-variable* is treated as a continuous variable in the primary input data table (specified in the DATA= option). Models in PROC CSPATIALREG do not allow missing values. If there are missing values, an error message is returned.

You can specify only one MODEL statement. You can specify the following *options* after a slash (/):

NOINT

suppresses the intercept parameter.

TYPE=ALL | AUTO | CAR | LINEAR | SAC | SAR | SARMA | SEM | SMA

specifies the type of model to be fit. If you specify this option in both the MODEL statement and the PROC CSPATIALREG statement, the MODEL statement overrides the PROC CSPATIALREG statement. You can specify the following values:

ALL	fits all models.
AUTO	fits a subset of selected models.
CAR	fits a conditional autoregressive model.
LINEAR	fits a linear model.
SAC	fits a spatial autoregressive confused model.
SAR	fits a spatial autoregressive model.
SARMA	fits a spatial autoregressive moving average model.
SEM	fits a spatial error model.
SMA	fits a spatial moving average model.

By default, TYPE=SAR. For more information about the TYPE=ALL or TYPE=AUTO option, see the section “[Multiple Model Comparison](#)” on page 679.

If you specify covariates in the SPATIALEFFECT statement, then their spatial lag is added to the model. In that case, you can specify the following TYPE= option values to get the model that you want:

TYPE=CAR	fits the CAR2 model.
TYPE=LINEAR	fits the SLX model.
TYPE=SAC	fits the SDAC model.
TYPE=SAR	fits the SDM.
TYPE=SARMA	fits the SDARMA model.
TYPE=SEM	fits the SDEM.
TYPE=SMA	fits the SDMA model.

Printing Options

You can specify the following options in either the PROC CSPATIALREG statement or the MODEL statement. If you specify these options in both the PROC CSPATIALREG and MODEL statements, the MODEL statement overrides the PROC CSPATIALREG statement.

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

NOPRINT

suppresses all printed output.

PRINTALL

prints all available output.

Optimization Control Options

You can specify the following options in either the PROC CSPATIALREG statement or the MODEL statement. If you specify these options in both the PROC CSPATIALREG and MODEL statements, the MODEL statement overrides the PROC CSPATIALREG statement.

MAXFUNC=*i*

MAXFU=*i*

specifies the maximum number of function calls in the optimization process. By default, MAXFUNC=1000.

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually used can exceed the number of calls that you specify in this option.

MAXITER=*i*

MAXIT=*i*

specifies the maximum number of iterations in the optimization process. By default, MAXITER=200.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. The default value is the largest floating-point double representation available on your computer. The time that you specify in this option is checked only once at the end of each iteration. Therefore, the actual run time can be much longer than *r*. The actual run time includes the remaining time that is needed to finish the iteration and the time that is needed to generate the output of the results.

METHOD=CONGRA | DBLDOG | NEWRAP | NONE | NRRIDG | QUANEW | TRUREG

specifies the iterative minimization method to use. You can specify the following methods:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double-dogleg method.
NEWRAP	specifies the Newton-Raphson method.
NONE	specifies that optimization not be performed.

NRRIDG	specifies the Newton-Raphson ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

By default, METHOD=NEWRAP.

OUTPUT Statement

```
OUTPUT OUT=libref.data-table
           < options >
           < keyword < =name > > . . . < keyword < =name > > ;
```

The OUTPUT statement creates a data table that contains observationwise statistics that PROC CSPATIALREG computes after fitting the model. These observationwise statistics include the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the residual. In order to avoid data duplication for large data tables, the variables in the input data table are *not* included in the output data table unless you specify them in the COPYVAR= option.

You can specify only one OUTPUT statement. You must specify the following option:

OUT=*libref.data-table*
 names the output data table for PROC CSPATIALREG to use. You must specify this option before any other options. *libref.data-table* is a two-level name, where

<i>libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>libref</i> , see the section “Using CAS Sessions and CAS Engine Librefs” on page 638.
<i>data-table</i>	specifies the name of the output data table.

You can also specify the following *options*:

COPYVAR=*variable*

COPYVARS=*variables*

transfers one or more variables from the primary input data table to the output data table.

PRED=*name*

MEAN=*name*

assigns a name to the variable that contains the predicted value of the response variable.

RESID=*name*

RESIDUAL=*name*

assigns a name to the variable that contains the residuals (that is, the difference between the observed and predicted values of the response variable).

XBETA=*name*

names the variable that contains estimates of $\mathbf{x}_i' \boldsymbol{\beta}$.

RESTRICT Statement

RESTRICT *restriction1* [, *restriction2* ...] ;

The RESTRICT statement imposes linear restrictions on the parameter estimates. You can specify any number of RESTRICT statements.

Each *restriction* is written as an expression, followed by an equality operator (=) or an inequality operator (<, >, <=, >=), followed by a second expression:

expression operator expression

The *operator* can be =, <, >, <=, or >=.

Restriction expressions can be composed of parameter names; constants; and the operators times (*), plus (+), and minus (–). The restriction expressions must be a linear function of the parameters.

Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the RESTRICT statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

For more information about how parameters are named in the RESTRICT statement, see the section “[Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 680.

Lagrange multipliers are reported in the “Parameter Estimates” table for all the active linear constraints. They are identified by the names Restrict1, Restrict2, and so on. The *p*-values of these Lagrange multipliers are computed using a beta distribution (LaMotte 1994). Nonactive (nonbinding) restrictions have no effect on the estimation results and are not noted in the output.

For example, the following RESTRICT statement constrains the spatial autoregressive coefficient ρ to 0, which removes endogenous interaction effects:

```
restrict _rho = 0;
```

SPATIALEFFECTS Statement

SPATIALEFFECTS < *spatial-effect-regressors* > < /*options* > ;

The SPATIALEFFECTS statement enables you to specify covariates (such as *Z*) whose spatial lag, *WZ*, is to be added to the model.

PROC CSPATIALREG adds the spatially weighted *spatial-effect-regressors* to regressors that you specify in the MODEL statement. For example, if you specify q variables $z_1, \dots, z_q$ in the SPATIALEFFECTS statement, then each of q spatially weighted variables, as represented by each column of *WZ*, has a parameter to be included in the regression. Here, *WZ* denotes the matrix product of *W* and *Z*. In addition, *Z* is the design matrix that is formed by the q variables $z_1, \dots, z_q$. The spatial weights matrix *W* comes from the data table that you specify in the WMAT= option. The “Parameter Estimates” table in the displayed output shows the estimates for spatially weighted *spatial-effect-regressors*; they are labeled with the prefix “W_”. For example, if you specify *z* (a variable in your primary data table) as a *model-spatial-effect-regressor*, then the “Parameter Estimates” table labels the corresponding parameter estimate “W_*z*”.

SPATIALID Statement

SPATIALID *variable* ;

The SPATIALID statement specifies a variable that identifies a spatial unit for each observation in the two data tables that you specify in the DATA= and WMAT= options in the PROC CSPATIALREG statement. The variable that you specify in the SPATIALID statement is also used to match the rows and columns in the spatial weights matrix. Only one SPATIALID statement and one spatial ID *variable* are allowed. For all models except a purely linear regression model without an SLX effect, you need a SPATIALID statement in order to specify a spatial ID *variable* for the purpose of matching observations in two data tables that you specify in the DATA= and WMAT= options. The values of the spatial ID *variable* in either the DATA= or WMAT= data table cannot be missing.

The *variable* in the SPATIALID statement can be either numeric or character. However, the type of spatial ID *variable* in the two data tables that you specify in the DATA= and WMAT= options must be the same. When the spatial ID *variable* is numeric, it needs to be integer-valued. If you specify a number that is not an integer, PROC CSPATIALREG uses the integer part of that number for matching.

TEST Statement

< *label* : > **TEST** < '*string*' > *equation1* < , *equation2* . . . > / < *test-options* > ;

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the parameters that you specify in the preceding MODEL statement.

You can add a label (which is printed in the output) to a TEST statement in two ways: add an unquoted *label* followed by a colon before the TEST keyword, or add a quoted *string* after the TEST keyword. The unquoted *label* cannot contain spaces. If you include both an unquoted *label* and a quoted *string*, PROC CSPATIALREG uses the unquoted *label*. If you specify neither an unquoted *label* nor a quoted *string*, PROC CSPATIALREG automatically labels the tests.

Each *equation* specifies a linear hypothesis to be tested and consists of regression parameter names and relational operators. The regression parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the TEST statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

For more information about how parameters are named in the TEST statement, see the section “[Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 680. Only linear equality restrictions and tests are permitted in PROC CSPATIALREG. Test *equations* can consist only of algebraic operations that involve the addition symbol (+), subtraction symbol (-), and multiplication symbol (*).

All hypotheses in a TEST statement are tested jointly.

You can specify the following options after the slash (/):

ALL

requests the Wald, Lagrange multiplier, and likelihood ratio tests.

LM

requests the Lagrange multiplier test.

LR

requests the likelihood ratio test.

WALD

requests the Wald test.

The following statements illustrate the use of the TEST statement:

```
proc cspatialreg data=mylib.dat;
  model y = x1 x2 x3/type=LINEAR;
  test x1 = 0, x2 * .5 + 2 * x3 = 0/ALL;
  test_int: test intercept = 0, x3 = 0/LR;
run;
```

The first test investigates the joint hypothesis that $\beta_1 = 0$ and $0.5\beta_2 + 2\beta_3 = 0$.

The TEST statement accepts labels that are reproduced in the printed output. TEST statements can be labeled in two ways: a TEST statement can be preceded by a label followed by a colon, or the keyword TEST can be followed by a quoted string. If both are present, PROC CSPATIALREG uses the label preceding the colon. If no label is specified, PROC CSPATIALREG automatically labels the tests.

Details: CSPATIALREG Procedure

Missing Values

Missing values can occur in both the primary data table (which you specify in the `DATA=` option) and the secondary spatial weights data table (which you specify in the `WMAT=` option). The CSPATIALREG procedure does not allow missing values in the primary input data table. If any observation in the primary input data table has a missing value for one or more of the regressors or the dependent variable, PROC CSPATIALREG will not fit the model. If any observation in the primary data table has a missing value for the spatial ID variable when you specify the `SPATIALID` statement, PROC CSPATIALREG will not fit the model. In such cases, you get an error message.

For the secondary spatial weights data table, a missing value for the spatial ID variable is not allowed when you specify the `SPATIALID` statement. Moreover, missing spatial weights are not allowed if you specify the `NONNORMALIZE` option. In these cases, PROC CSPATIALREG issues an error message and skips the model fitting.

Spatial Autoregressive Models

The spatial autoregressive (SAR) model is useful for incorporating spatial dependence in the dependent variable—that is, the endogenous interaction effect. Let y_i denote the observation that is associated with a spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let an $n \times n$ matrix $\mathbf{W}$ with nonnegative elements be a spatial weights matrix and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded for the spatial unit s_i . You can formulate the SAR model as

$$y_i = \rho \sum_{j=1}^n W_{ij} y_j + \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

where $i = 1, 2, \dots, n$, ρ is the spatial autoregressive coefficient, $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector, W_{ij} is the (i, j) th element of the matrix $\mathbf{W}$ subject to $W_{ii} = 0$, and the error term ϵ_i that is related to the spatial unit s_i is assumed to follow $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$ for $i = 1, 2, \dots, n$.

The SAR model is often described in vector form as

$$\mathbf{y} = \rho \mathbf{W} \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, and $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$.

The standard estimator for the SAR model is the maximum likelihood estimator (MLE). For the SAR model, the log-likelihood function is (Anselin 2001)

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^2} + \ln |\mathbf{A}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}$, $\mathbf{I}_n$ is an $n \times n$ identity matrix, and $|\mathbf{A}|$ denotes the determinant of $\mathbf{A}$.

You can derive the gradients as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} &= \frac{\mathbf{X}'(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^2} \\ \frac{\partial \mathcal{L}}{\partial \rho} &= \frac{1}{\sigma^2} \mathbf{y}'\mathbf{W}'(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) - \text{tr}(\mathbf{A}^{-1}\mathbf{W}) \\ \frac{\partial \mathcal{L}}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} + \frac{(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^4}\end{aligned}$$

For the $n \times n$ matrix $\mathbf{A}$, $\text{tr}(\mathbf{A}) = \sum_{i=1}^n a_{ii}$, where a_{ii} is the i th diagonal element of $\mathbf{A}$.

A SAR model does not account for exogenous interaction effects. In practice, the value of the dependent variable y for a spatial unit might also be affected by some independent explanatory variables of other spatial units. In such a case, you can use the spatial Durbin model instead.

Spatial Durbin Models

Unlike a SAR model, a spatial Durbin model (SDM) can account for exogenous interaction effects in addition to the endogenous interaction effects. Let y_i denote the observation that is associated with a spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix of your choice. Further assume that $\mathbf{x}_i$ is a $p \times 1$ vector that denotes values of p regressors that are recorded for the spatial unit s_i and assume that $\mathbf{z}_i$ is a $q \times 1$ vector that denotes values of q regressors that are measured at unit s_i .

The SDM can be described in vector form as (LeSage and Pace 2009)

$$\mathbf{y} = \rho \mathbf{W}\mathbf{y} + \mathbf{X}\boldsymbol{\beta} + \mathbf{WZ}\boldsymbol{\theta} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, $\mathbf{Z}$ is an $n \times q$ matrix where each row consists of $\mathbf{z}_i'$, and $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are $p \times 1$ and $q \times 1$ parameter vectors, respectively.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{WZ}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, you can rewrite the SDM as

$$\mathbf{y} = \rho \mathbf{W}\mathbf{y} + \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}} + \boldsymbol{\epsilon}$$

The log-likelihood function for the SDM is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})'(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{2\sigma^2} + \ln |\mathbf{A}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}$.

For the SDM, the gradients are

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} &= \frac{\widetilde{\mathbf{X}}'(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{\sigma^2} \\ \frac{\partial \mathcal{L}}{\partial \rho} &= \frac{1}{\sigma^2} \mathbf{y}'\mathbf{W}'(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}}) - \text{tr}(\mathbf{A}^{-1}\mathbf{W}) \\ \frac{\partial \mathcal{L}}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} + \frac{(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})'(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{2\sigma^4}\end{aligned}$$

Both the SDM and the SAR model account for endogenous interaction effects. However, in some cases there might be an interaction among error terms. In such cases, you might consider a spatial error model, which addresses spatial interaction among error terms.

Spatial Error Models

The spatial error model (SEM) accounts for spatial dependence in the error terms rather than in the dependent variable. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i .

The SEM can be described in vector form by using the following two-stage formulation (LeSage and Pace 2009),

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{u}$$

$$\mathbf{u} = \lambda \mathbf{W}\mathbf{u} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

The log-likelihood function for the SEM is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^2} + \ln |\mathbf{B}|$$

where $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}$.

For the SEM, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{(\mathbf{B}\mathbf{X})' [\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{\sigma^2} [\mathbf{W}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})] - \text{tr}(\mathbf{B}^{-1}\mathbf{W})$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^4}$$

In addition to the interaction effects among error terms, you might also want to include exogenous interaction effects in the model. In such cases, you need to consider a spatial Durbin error model.

Spatial Durbin Error Models

The spatial Durbin error model (SDEM) accounts for spatial dependence among the error terms and the exogenous interaction effect. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i , and let $\mathbf{z}_i$ be a $q \times 1$ vector that denotes values of q regressors that are measured at unit s_i .

The SDEM can be described in vector form by using the following two-stage formulation (LeSage and Pace 2009),

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{W}\mathbf{Z}\boldsymbol{\theta} + \mathbf{u}$$

$$\mathbf{u} = \lambda \mathbf{W}\mathbf{u} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, $\mathbf{Z}$ is an $n \times q$ matrix where each row consists of $\mathbf{z}_i'$, and $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are $p \times 1$ and $q \times 1$ parameter vectors, respectively.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{WZ}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, you can rewrite the SDEM as

$$\mathbf{y} = \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}} + \mathbf{B}^{-1}\boldsymbol{\epsilon}$$

where $\mathbf{B} = \mathbf{I}_n - \lambda\mathbf{W}$.

The log-likelihood function for the SDEM is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^2} + \ln |\mathbf{B}|$$

For the SDEM, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} = \frac{(\mathbf{B}\widetilde{\mathbf{X}})' [\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{\sigma^2} [\mathbf{W}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})] - \text{tr}(\mathbf{B}^{-1}\mathbf{W})$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^4}$$

Spatial Moving Average Models

The spatial moving average (SMA) model accounts for spatial dependence among the error terms; thus it is similar to the SEM model but with a different autocorrelation structure. The SMA model is used for modeling local autocorrelation. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix, and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i .

The SMA model can be described in vector form by using the following two-stage formulation,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{u}$$

$$\mathbf{u} = (\mathbf{I}_n - \lambda\mathbf{W})\boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that has $\mathbf{x}_i'$ in each row, $\mathbf{Z}$ is an $n \times q$ matrix that has $\mathbf{z}_i'$ in each row, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

The log-likelihood function for the SMA model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^2} - \ln |\mathbf{B}|$$

where $\mathbf{B} = \mathbf{I}_n - \lambda\mathbf{W}$.

For the SMA model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{(\mathbf{B}^{-1}\mathbf{X})' [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}\mathbf{W}] [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})] + \text{tr}(\mathbf{B}^{-1}\mathbf{W})$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^4}$$

Spatial Durbin Moving Average Models

The term spatial Durbin moving average (SDMA) model is used to refer to the SMA model that accommodates exogenous interaction effects. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i , and let $\mathbf{z}_i$ be a $q \times 1$ vector that denotes values of q covariates that are measured at unit s_i .

The SDMA model can be described in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{WZ}\boldsymbol{\theta} + (\mathbf{I}_n - \lambda\mathbf{W})\boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$; $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$; $\mathbf{X}$ and $\mathbf{Z}$ are $n \times p$ and $n \times q$ matrices, where each row consists of $\mathbf{x}_i'$ and $\mathbf{z}_i'$, respectively; $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector; and $\boldsymbol{\theta}$ is a $q \times 1$ parameter vector.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{WZ}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, the SDMA model can be written as

$$\mathbf{y} = \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}} + \mathbf{B}\boldsymbol{\epsilon}$$

The log-likelihood function for the SDMA model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}^{-1}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^2} - \ln |\mathbf{B}|$$

where $\mathbf{B} = \mathbf{I}_n - \lambda\mathbf{W}$ and $|\mathbf{B}|$ denotes the determinant of matrix $\mathbf{B}$.

For the SDMA model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} = \frac{(\mathbf{B}^{-1}\widetilde{\mathbf{X}})' [\mathbf{B}^{-1}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} [\mathbf{B}^{-1}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1}\mathbf{W}] [\mathbf{B}^{-1}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})] + \text{tr}(\mathbf{B}^{-1}\mathbf{W})$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}^{-1}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^4}$$

Spatial Autoregressive Moving Average Models

The spatial autoregressive moving average (SARMA) model, like the SMA model, can account for spatial dependence among the error terms. In addition, the SARMA model enables you to account for spatial dependence in the dependent variable, as the SAR model does. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let the $n \times n$ matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ be two spatial weights matrices that contain nonnegative elements. In practice, $\mathbf{W}_1$ and $\mathbf{W}_2$ can be identical. Further, it is assumed that $\mathbf{x}_i$ is a $p \times 1$ vector that denotes values of p covariates that are recorded at unit s_i .

The SARMA model can be described in vector form by using the following two-stage formulation (LeSage and Pace 2009),

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{u}$$

$$\mathbf{u} = (\mathbf{I}_n - \lambda \mathbf{W}_2) \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that consists of $\mathbf{x}_i'$ in each row, $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector, and $\mathbf{I}_n$ is an $n \times n$ identity matrix.

The log-likelihood function for the SARMA model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^2} + \ln |\mathbf{A}| - \ln |\mathbf{B}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}_1$, $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}_2$, and $|\cdot|$ denotes the matrix determinant operator.

For the SARMA model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{(\mathbf{B}^{-1} \mathbf{X})' [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = \frac{(\mathbf{B}^{-1} \mathbf{W}_1 \mathbf{y})' \mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^2} - \text{tr}(\mathbf{A}^{-1} \mathbf{W}_1)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1} \mathbf{W}_2] [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})] + \text{tr}(\mathbf{B}^{-1} \mathbf{W}_2)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^4}$$

Spatial Durbin Autoregressive Moving Average Models

You can also accommodate exogenous interaction effects in the SARMA model. The term spatial Durbin autoregressive moving average (SDARMA) model is used to refer to such an extension of the SARMA model. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}_1$ and $\mathbf{W}_2$ be two spatial weights matrices. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i , and let $\mathbf{z}_i$ be a $q \times 1$ vector that denotes values of q regressors that are measured at unit s_i .

The SDARMA model can be described in vector form by using the following two-stage formulation,

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{W}_1 \mathbf{Z} \boldsymbol{\theta} + \mathbf{u}$$

$$\mathbf{u} = (\mathbf{I}_n - \lambda \mathbf{W}_2) \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that has $\mathbf{x}_i'$ in each row, $\mathbf{Z}$ is an $n \times q$ matrix that has $\mathbf{z}_i'$ in each row, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{W}_1 \mathbf{Z}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, the SDARMA model can be written as

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \widetilde{\mathbf{X}} \widetilde{\boldsymbol{\beta}} + (\mathbf{I}_n - \lambda \mathbf{W}_2) \boldsymbol{\epsilon}$$

The log-likelihood function for the SDARMA model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^2} + \ln |\mathbf{A}| - \ln |\mathbf{B}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}_1$ and $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}_2$.

For the SDARMA model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} = \frac{(\mathbf{B}^{-1} \widetilde{\mathbf{X}})' [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = \frac{(\mathbf{B}^{-1} \mathbf{W}_1 \mathbf{y})' \mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{\sigma^2} - \text{tr}(\mathbf{A}^{-1} \mathbf{W}_1)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1} \mathbf{W}_2] [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})] + \text{tr}(\mathbf{B}^{-1} \mathbf{W}_2)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^4}$$

Spatial Autoregressive Confused Models

The spatial autoregressive confused (SAC) model, like the SARMA model, can accommodate spatial dependence in both the dependent variable and error terms. However, the covariance structure for the error terms in a SAC model is different from that of the SARMA model. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}_1$ and $\mathbf{W}_2$ be two spatial weights matrices, and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i .

The SAC model can be described in vector form by using the following two-stage formulation (LeSage and Pace 2009),

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{u}$$

$$\mathbf{u} = \lambda \mathbf{W}_2 \mathbf{u} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that has $\mathbf{x}_i'$ in each row, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

The log-likelihood function for the SAC model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^2} + \ln |\mathbf{A}| + \ln |\mathbf{B}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}_1$ and $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}_2$.

For the SAC model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{(\mathbf{B}\mathbf{X})' [\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = \frac{(\mathbf{B}\mathbf{W}_1 \mathbf{y})' \mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^2} - \text{tr}(\mathbf{A}^{-1} \mathbf{W}_1)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{\sigma^2} [\mathbf{W}_2(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})] - \text{tr}(\mathbf{B}^{-1} \mathbf{W}_2)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^4}$$

Spatial Durbin Autoregressive Confused Models

The SAC model can be extended to account for exogenous interaction effects. The term spatial Durbin autoregressive confused (SDAC) model is used to refer to such an extension of the SAC model. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}_1$ and $\mathbf{W}_2$ be two spatial weights matrices. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i , and assume that $\mathbf{z}_i$ is a $q \times 1$ vector that denotes values of q regressors that are measured at unit s_i .

The SDAC model can be described in vector form by using the following two-stage formulation,

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{W}_1 \mathbf{Z} \boldsymbol{\theta} + \mathbf{u}$$

$$\mathbf{u} = (\mathbf{I}_n - \lambda \mathbf{W}_2)^{-1} \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that has $\mathbf{x}_i'$ in each row, $\mathbf{Z}$ is an $n \times q$ matrix that has $\mathbf{z}_i'$ in each row, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{W}_1 \mathbf{Z}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, the SDAC model can be rewritten as

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \widetilde{\mathbf{X}} \widetilde{\boldsymbol{\beta}} + (\mathbf{I}_n - \lambda \mathbf{W}_2)^{-1} \boldsymbol{\epsilon}$$

The log-likelihood function for the SDAC model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^2} + \ln |\mathbf{A}| + \ln |\mathbf{B}|$$

For the SDAC model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} = \frac{(\mathbf{B}\widetilde{\mathbf{X}})' [\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = \frac{(\mathbf{B}\mathbf{W}_1 \mathbf{y})' \mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{\sigma^2} - \text{tr}(\mathbf{A}^{-1} \mathbf{W}_1)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{\sigma^2} [\mathbf{W}_2(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})] - \text{tr}(\mathbf{B}^{-1} \mathbf{W}_2)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^4}$$

Linear Regression Models

You can also fit a linear regression model in PROC CSPATIALREG. In this case, let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$, and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors recorded at unit s_i .

You can describe the linear regression model in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, and $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$.

The log-likelihood function for the linear regression model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^2}$$

For the linear regression model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{\mathbf{X}'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^4}$$

The Hessians take the following forms:

$$\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} = -\frac{\mathbf{X}'\mathbf{X}}{\sigma^2}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\beta} \partial \sigma^2} = -\frac{\mathbf{X}'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^4}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \sigma^4} = \frac{n}{2\sigma^4} - \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^6}$$

Spatial Lag of X Models

The spatial lag of X (SLX) model assumes no endogenous interaction effects or spatial dependence in the error terms. Instead, it incorporates only exogenous interaction effects into the linear regression model. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be a spatial weights matrix. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors recorded at unit s_i , and let $\mathbf{z}_i$ be a $q \times 1$ vector that denotes values of q regressors measured at unit s_i .

The SLX model can be described in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{WZ}\boldsymbol{\theta} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, $\mathbf{Z}$ is an $n \times q$ matrix where each row consists of $\mathbf{z}_i'$, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

By letting $\tilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{WZ}]$ and $\tilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, you can rewrite the SLX model as

$$\mathbf{y} = \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}} + \boldsymbol{\epsilon}$$

The log-likelihood function for the SLX model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})'(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})}{2\sigma^2}$$

For the SLX model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \tilde{\boldsymbol{\beta}}} = \frac{(\tilde{\mathbf{X}})'(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})'(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})}{2\sigma^4}$$

The Hessians take the following forms:

$$\frac{\partial^2 \mathcal{L}}{\partial \tilde{\boldsymbol{\beta}} \partial \tilde{\boldsymbol{\beta}}'} = -\frac{\tilde{\mathbf{X}}'\tilde{\mathbf{X}}}{\sigma^2}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \tilde{\boldsymbol{\beta}} \partial \sigma^2} = -\frac{\tilde{\mathbf{X}}'(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})}{\sigma^4}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \sigma^4} = \frac{n}{2\sigma^4} - \frac{(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})'(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})}{\sigma^6}$$

Conditional Autoregressive Models

Let y_i denote the observation that is associated with a spatial unit s_i for $i = 1, 2, \dots, n$, and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors recorded at unit s_i .

The conditional autoregressive (CAR) model is defined by the set of full conditional distributions as

$$y_i | \mathbf{Y}_{(i)} \sim N \left(\mathbf{x}_i' \boldsymbol{\beta} + \sum_{j=1}^n C_{ij} (y_j - \mathbf{x}_j' \boldsymbol{\beta}), \sigma_i^2 \right)$$

where $\mathbf{Y}_{(i)} = (y_1, y_2, \dots, y_{i-1}, y_{i+1}, \dots, y_n)'$ and $\mathbf{C}$ is an $n \times n$ matrix whose (i, j) th element is C_{ij} .

Denote $\mathbf{M} = \text{diagonal}(\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2)$. Under the conditions that $\mathbf{M}^{-1}\mathbf{C}$ is symmetric and $\mathbf{M}^{-1}(\mathbf{I}_n - \mathbf{C})$ is positive definite, the joint distribution of $\mathbf{y}$ is well defined as

$$\mathbf{y} \sim N(\mathbf{X}\boldsymbol{\beta}, \boldsymbol{\Sigma})$$

where $\mathbf{X}$ is an $n \times p$ matrix in which each row consists of $\mathbf{x}_i'$ and $\boldsymbol{\Sigma} = (\mathbf{I}_n - \mathbf{C})^{-1}\mathbf{M}$.

The log-likelihood function for the CAR model takes the form

$$\mathcal{L} = -\frac{n}{2} \ln 2\pi - \frac{1}{2} \ln |\boldsymbol{\Sigma}| - \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})' \boldsymbol{\Sigma}^{-1} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2}$$

Often, it is considered that $\mathbf{C} = \rho \mathbf{W}$, where $\mathbf{W}$ is a spatial weights matrix, and

$$\boldsymbol{\Sigma} = \begin{cases} \sigma^2 (\mathbf{I}_n - \rho \mathbf{W})^{-1} & \text{if } \mathbf{W} \text{ is symmetric} \\ \sigma^2 (\mathbf{D} - \rho \mathbf{W})^{-1} & \text{if a symmetric } \mathbf{W} \text{ is row-standardized} \end{cases}$$

where $\mathbf{D} = \text{diagonal}(\mathbf{W}_{1+}, \mathbf{W}_{2+}, \dots, \mathbf{W}_{n+})$, in which $\mathbf{W}_{i+}$ is the sum of entries in the i th row of the $\mathbf{W}$ matrix for $i = 1, 2, \dots, n$.

Let $\mathbf{D}^\dagger$ be defined as

$$\mathbf{D}^\dagger = \begin{cases} \mathbf{I}_n & \text{if a symmetric } \mathbf{W} \text{ is used} \\ \mathbf{D} & \text{if the row-standardized version of a symmetric } \mathbf{W} \text{ is used} \end{cases}$$

The gradients for the CAR model can be derived as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{\mathbf{X}' \mathbf{D}^\dagger (\mathbf{I} - \rho \mathbf{W}) (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = -\frac{1}{2} \text{tr}(\mathbf{A}^{-1} \mathbf{W}) + \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})' \mathbf{D}^\dagger \mathbf{W} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})' \mathbf{D}^\dagger (\mathbf{I}_n - \rho \mathbf{W}) (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2(\sigma^2)^2}$$

where $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}$.

Specifying the Spatial Weights Matrix

The spatial weights matrix $\mathbf{W}$ plays a vital role in spatial econometric modeling. If you fit a purely linear model without SLX effects, you do not need a $\mathbf{W}$ matrix. For other types of models in PROC CSPATIALREG, you need to provide a spatial weights matrix to fit the model. Although how you create the $\mathbf{W}$ matrix is often problem-specific, there are some general guidelines to consider. Two common ways to create the $\mathbf{W}$ matrix are k -order binary contiguity matrices and k -nearest-neighbor matrices (Elhorst 2013).

k -Order Binary Contiguity Matrices

To create the $\mathbf{W}$ matrix by using k -order binary contiguity matrices, you start with the spatial contiguity matrix $\mathbf{C}$. In the case of the first-order neighbors ($k = 1$), a value of 1 for the (i, j) th entry in $\mathbf{C}$ indicates that the two units i and j are neighbors to each other, and 0 indicates otherwise. The neighbor relationship is often defined based on sharing of a common boundary. To generalize this, a k -order neighbor ($k \geq 2$) of a unit i can be any units whose neighbors are $(k - 1)$ -order neighbors of unit i . In this sense, the two units i and j that are not first-order neighbors can still be second-order neighbors if unit j is the neighbor to a first-order neighbor of unit i .

As an example, a first-order binary contiguity matrix might look like the following:

$$\mathbf{C} = \begin{pmatrix} \text{SID} & \text{L1} & \text{L2} & \text{L3} & \text{L4} \\ \text{L1} & 0 & 1 & 0 & 1 \\ \text{L2} & 1 & 0 & 0 & 0 \\ \text{L3} & 0 & 0 & 0 & 1 \\ \text{L4} & 1 & 0 & 1 & 0 \end{pmatrix}$$

The diagonal elements of $\mathbf{C}$ are zeros because, in general, a unit is not considered to be a neighbor of itself. Moreover, the two units L2 and L4 are neighbors of L1; L2 has L1 as its only neighbor; L3 has L4 as its only neighbor; and L4 has L1 and L3 as its neighbors. You can create the spatial weights matrix $\mathbf{W}$ by row-standardizing the contiguity matrix $\mathbf{C}$. To do so, you divide entries in each row of $\mathbf{C}$ by the sum of that row. The spatial weights matrix $\mathbf{W}$, which is the row-standardized version of $\mathbf{C}$, is as follows:

$$\mathbf{W} = \begin{pmatrix} \text{SID} & \text{L1} & \text{L2} & \text{L3} & \text{L4} \\ \text{L1} & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \text{L2} & 1 & 0 & 0 & 0 \\ \text{L3} & 0 & 0 & 0 & 1 \\ \text{L4} & \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

k -Nearest-Neighbor Matrices

To create the $\mathbf{W}$ matrix by using k -nearest-neighbor matrices, you can start by creating a spatial contiguity matrix based on a distance metric. Let d_{ij} denote the distance between the two units i and j , which might be the Euclidean distance between centroids of the two spatial units. Let $(\text{lon}_i, \text{lat}_i)$ and $(\text{lon}_j, \text{lat}_j)$ be the centroids of units i and j , where $1 \leq i, j \leq n$, and lon and lat denote the longitude and latitude, respectively. Under the Euclidean distance metric, the distance d_{ij} between units i and j is

$$d_{ij} = \sqrt{(\text{lat}_i - \text{lat}_j)^2 + (\text{lon}_i - \text{lon}_j)^2}$$

After computing the distance between the unit i and other units under a certain metric, you sort d_{ij} in ascending order; for example, $d_{ij_1} \leq d_{ij_2} \leq \dots \leq d_{ij_k} \leq \dots \leq d_{ij_{n-1}}$. For a given k , let $N_k(i) = \{j_1, j_2, \dots, j_k\}$ be the set that contains the indices of k -nearest neighbors of unit i ; then the (i, j) th entries of the contiguity matrix $\mathbf{C}$ are defined as

$$C_{ij} = \begin{cases} 1 & \text{if } j \in N_k(i) \\ 0 & \text{otherwise} \end{cases}$$

The (i, j) th entry of the corresponding row-standardized matrix $\mathbf{W}$ is $W_{ij} = C_{ij} \left\{ \sum_{j \in N_k(i)} C_{ij} \right\}^{-1}$.

Unlike the k -order binary contiguity matrix, which is often symmetric by construction, k -nearest-neighbor matrices can be asymmetric. To obtain symmetric k -nearest neighbor matrices, you can define the (i, j) th entries of the contiguity matrix $\mathbf{C}$ as follows:

$$C_{ij} = \begin{cases} 1 & \text{if } j \in N_k(i) \text{ or } i \in N_k(j) \\ 0 & \text{otherwise} \end{cases}$$

In addition to the Euclidean distance measure, you can use other distance metrics as appropriate. A variant of k -nearest-neighbor matrices $\mathbf{C}^*$ that is used in some empirical studies defines its (i, j) th entries as

$$C_{ij}^* = \begin{cases} 1 & \text{if } d_{ij} \leq d_{\text{cutoff}} \\ 0 & \text{otherwise} \end{cases}$$

where d_{cutoff} is a prespecified threshold distance.

In addition to the two constructions of spatial weights matrices that are presented earlier, see Elhorst (2013) and the references therein for more information about other ways to create a spatial weights matrix. In practice, you can define the neighbor relation that is problem-specific. For example, you can define two spatial units that are far apart to be neighbors because they share some attributes (such as population sizes larger than 500,000).

The data table that you specify in the `WMAT=` option is row-standardized by default to create a spatial weights matrix. This means that if you specify `WMAT=C`, PROC CSPATIALREG row-standardizes the spatial contiguity matrix to create a spatial weights matrix. If you want to suppress row standardization, you must specify the `NONNORMALIZE` option in the PROC CSPATIALREG statement.

Compact Representation of the Spatial Weights Matrix

When the number of spatial units n increases, the amount of memory that it takes to store n^2 entries of the spatial contiguity matrix $\mathbf{C}$ or the spatial weights matrix $\mathbf{W}$ increases dramatically. To circumvent the storage issue, PROC CSPATIALREG enables you to provide a compact representation of $\mathbf{W}$ (or $\mathbf{C}$) when appropriate. For the compact matrix representation, you provide a data table that contains three variables by using the `WMAT=` option. The first two variables identify the row r and column c of $\mathbf{W}$ (or $\mathbf{C}$), and (r, c) can be expressed either as numerical indices or as values of the variable that you specify in the `SPATIALID` statement. The third variable contains the nonzero value of $\mathbf{W}$ (or $\mathbf{C}$) for row r and column c . In this compact representation, the number of observations in the data table that you specify in the `WMAT=` option equals the total number of nonzero entries in $\mathbf{W}$ (or $\mathbf{C}$).

You must use a `SPATIALID` statement if you want to use the compact representation of the spatial contiguity or spatial weights matrix. In the compact representation, the first two variables of the data table that you specify in the `WMAT=` option must be of the same type. First, the first two columns in that data table can

be the row and column index for each nonzero entry in **W** (or **C**). In this case, the SPATIALID variable is numeric type. Alternatively, the first two columns in the WMAT= data table can be characters that are the names of two neighboring spatial units in **W** (or **C**). In this second case, the SPATIALID variable is character type.

For example, the compact representation of the spatial weights matrix **W**,

$$\mathbf{W} = \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix}$$

would look like the following:

```
data Ws;
  input SID cSID Weight;
  datalines;
  1 2 0.5
  1 4 0.5
  2 1 1.0
  3 4 1.0
  4 1 0.5
  4 3 0.5
  ;
run;
```

For the spatial contiguity matrix **C**,

$$\mathbf{C} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

the compact representation would look like the following:

```
data Cs;
  input SID cSID Weight;
  datalines;
  1 2 1.0
  1 4 1.0
  2 1 1.0
  3 4 1.0
  4 1 1.0
  4 3 1.0
  ;
run;
```

If the spatial weights matrix is the same as matrix **W** in the section “*k*-Order Binary Contiguity Matrices” on page 671, its compact representation would be as follows:

```
data Ws2;
  input SID $ cSID $ Weight;
  datalines;
  L1 L2 0.5
```

```

L1 L4 0.5
L2 L1 1.0
L3 L4 1.0
L4 L1 0.5
L4 L3 0.5
;
run;

```

If the spatial contiguity matrix is the same as matrix **C** in the section “*k*-Order Binary Contiguity Matrices” on page 671, its compact representation can be given in the data set **Cs2** as follows:

```

data Cs2;
  input SID $ cSID $ Weight;
  datalines;
L1 L2 1.0
L1 L4 1.0
L2 L1 1.0
L3 L4 1.0
L4 L1 1.0
L4 L3 1.0
;
run;

```

Spatial ID Matching

Depending on the type of model that you use in PROC CSPATIALREG, you might need to specify two data tables: one in the **DATA=** option and the other in the **WMAT=** option. In such cases, you must use a **SPATIALID** statement to specify a spatial ID variable in order to match observations in these two data tables.

As an example, assume that the data table that you specify in the **DATA=** option looks like the following:

```

data mylib.Example;
  input SID $ x1 x2 y;
  datalines;
L1 0.3 0.5 0.9
L3 -0.7 0.8 -0.4
L2 0.4 -1.2 0.6
L8 -1.7 1.2 -0.5
L4 1.4 0.9 0.3
L5 2.3 1.5 1.9
L7 -0.9 -0.8 -1.3
L6 1.4 -1.6 -2.0
;
run;

```

Suppose that the spatial contiguity matrix that you specify in the **WMAT=** option looks like the following:

```

data mylib.Cmat;
  input SID $ L1 L8 L3 L4 L7 L6 L5 L2;
  datalines;
L1 0 1 0 1 0 1 0 1
L2 1 0 0 0 1 0 0 0
L6 1 0 1 0 0 0 0 0

```

```

L4  1 0 0 0 1 0 0 0
L3  0 0 0 0 1 1 0 0
L7  0 0 1 1 0 0 0 1
L5  0 1 0 0 0 0 0 0
L8  1 0 0 0 0 0 1 0
;
run;

```

As you can see, rows in the two data tables `mylib.Example` and `mylib.Cmat` do not share identically sorted SID values. The second row in the `Example` data table contains the observation for a spatial unit L3, and its neighbor information is given in the fifth row of the `mylib.Cmat` data table. Moreover, the rows and columns of the spatial weights data table `mylib.Cmat` are not in the same order. The following SAS statements fit a SAR model to these data:

```

proc cspatialreg data=mylib.Example Wmat=mylib.Cmat;
  model y=x1 x2/type=SAR;
  spatialid SID;
run;

```

The `SPATIALID` statement enables you to match rows and columns of the `mylib.Cmat` data table in addition to rows of the `mylib.Example` and `mylib.Cmat` data tables.

Parameter Space of Spatial Coefficients

All models except linear regression models require that the parameters ρ and/or λ satisfy some assumptions to ensure consistency of the maximum likelihood estimator (Elhorst 2013). For SDMs and SAR models, the Jacobian term involves the log-determinant of $\mathbf{I} - \rho \mathbf{W}_1$, and the parameter space of ρ is specified such that $\mathbf{I} - \rho \mathbf{W}_1$ is nonsingular. For the SEM, SDEM, SMA, and SDMA models, the Jacobian term involves the log-determinant of $\mathbf{I} - \lambda \mathbf{W}_1$, and the parameter space of λ is specified such that $\mathbf{I} - \lambda \mathbf{W}_1$ is nonsingular. For the SAC, SDAC, SARMA, and SDARMA models, both $\mathbf{I} - \rho \mathbf{W}_1$ and $\mathbf{I} - \lambda \mathbf{W}_2$ must be nonsingular. As a result, the parameter space of ρ and λ is often specified such that both $\mathbf{I} - \rho \mathbf{W}_1$ and/or $\mathbf{I} - \lambda \mathbf{W}_1$ are nonsingular.

In the `CSPATIALREG` procedure, the parameter space of the spatial coefficients ρ and λ depends on the spatial weights matrix $\mathbf{W}$ that you choose. For a given $\mathbf{W}$, the parameter space of the parameters ρ and λ in `PROC CSPATIALREG` is determined as follows:

1. For a symmetric $\mathbf{W}$, the nonsingularity condition requires $\rho \in (\omega_{\min}^{-1}, \omega_{\max}^{-1})$ and $\lambda \in (\omega_{\min}^{-1}, \omega_{\max}^{-1})$. Here $\omega_{\min}$ and $\omega_{\max}$ denote the smallest (that is, most negative) and largest real eigenvalues of $\mathbf{W}$, respectively.
2. If $\mathbf{W}$ is symmetric and subsequently row-standardized, the nonsingularity condition requires $\rho \in (\omega_{\min}^{*-1}, 1)$ and $\lambda \in (\omega_{\min}^{*-1}, 1)$. Here $\omega_{\min}^*$ denotes the smallest purely real eigenvalue of the row-standardized $\mathbf{W}$.
3. If $\mathbf{W}$ is asymmetric and subsequently row-standardized, the nonsingularity condition requires $\rho \in (r_{\min}^{*-1}, 1)$ and $\lambda \in (r_{\min}^{*-1}, 1)$. Here $r_{\min}^*$ denotes the smallest purely real eigenvalue of the row-standardized $\mathbf{W}$.

4. When Taylor or Chebyshev approximation is used for SDMs and SAR models, $\mathbf{W}$ is required to be row-standardized. In these cases, the restriction on the spatial autoregressive coefficient ρ is $\rho \in (-1, 1)$.
5. When Taylor or Chebyshev approximation is used for SEMs and SDEMs, $\mathbf{W}$ is required to be row-standardized. In these cases, the restriction on the spatial autocorrelation coefficient λ is $\lambda \in (-1, 1)$.

For CAR models, only symmetric spatial weights matrices are allowed. As a result, the parameter space of ρ is $\rho \in (\omega_{\min}^{-1}, \omega_{\max}^{-1})$.

For the interpretation of the spatial coefficient ρ , a significant and positive (or negative) value of ρ in the CAR, SAR, SDM, SAC, SDAC, SARMA, and SDARMA models would suggest positive (or negative) spatial lag dependence. Similarly, a significant and positive (or negative) value of λ in the SEM, SDEM, SAC, and SDAC models would suggest positive (or negative) spatial error dependence, which is contrary to the SMA, SDMA, SARMA, and SDARMA models. For the SMA, SDMA, SARMA, and SDARMA models, a significant and positive (or negative) value of λ would suggest negative (or positive) spatial error dependence.

Approximations to the Jacobian

To obtain maximum likelihood estimates for all models except linear regression models, the Jacobian term must be computed, because it appears in the log-likelihood function. The Jacobian term is $\ln |\mathbf{I}_n - \rho \mathbf{W}|$ for SDMs and SAR models and $\ln |\mathbf{I}_n - \lambda \mathbf{W}|$ for SEMs and SDEMs, where n is the number of observations and $\mathbf{W}$ is the spatial weights matrix. When n is not large, the Jacobian is computed as follows,

$$\ln |\mathbf{I}_n - \rho \mathbf{W}| = \sum_{i=1}^n \ln |1 - \rho \omega_i|$$

where ω_i are the eigenvalues of $\mathbf{W}$ for $i = 1, 2, \dots, n$. Such a method requires you to precompute all eigenvalues of $\mathbf{W}$. This works fine for small data sets, but when n is large, computing the Jacobian term by using the eigenvalue method can be computationally infeasible. Instead, you can use approximations to the Jacobian.

The CSPATIALREG procedure supports two different approximations to the Jacobian. For SDMs, SEMs, SDEMs, and SAR models, you can approximate the Jacobian by using either Taylor or Chebyshev approximation. Using SDMs and SAR models as an example, the two approximations for the Jacobian term can be described as follows (for more information, see LeSage and Pace 2009, and the references therein):

- Taylor approximation uses finite, lower-order series to approximate the log-determinant as

$$\ln |\mathbf{I}_n - \rho \mathbf{W}| \approx - \sum_{k=1}^q \frac{\rho^k \text{tr}(\mathbf{W}^k)}{k}$$

- Chebyshev approximation uses finite, lower-order Chebyshev polynomials of the first kind to approximate the log-determinant as

$$\ln |\mathbf{I}_n - \rho \mathbf{W}| \approx \sum_{k=0}^q c_k(\rho) \text{tr}(T_k(\mathbf{W}))$$

where $T_0(\mathbf{W}) = \mathbf{I}_n$, $T_1(\mathbf{W}) = \mathbf{W}$, and $T_{k+1}(\mathbf{W}) = 2\mathbf{W}T_k(\mathbf{W}) - T_{k-1}(\mathbf{W})$ for $k = 1, 2, \dots, q$. The coefficients $c_k(\rho)$ are defined as

$$c_k(\rho) = \begin{cases} \frac{1}{q+1} \sum_{j=0}^q \ln(1 - \rho \cos \theta_j) \cos(k\theta_j) & \text{if } k = 0 \\ \frac{2}{q+1} \sum_{j=0}^q \ln(1 - \rho \cos \theta_j) \cos(k\theta_j) & \text{if } k > 0 \end{cases}$$

with $\theta_j = (j+1)\pi/(q+1)$ for $j = 0, 1, \dots, q$.

The traces of powers of $\mathbf{W}$ can be computed exactly or approximated using Monte Carlo simulation. The Monte Carlo simulation is done as follows,

$$\text{tr}(\mathbf{W}^k) \approx \frac{1}{M} \sum_{l=1}^M \frac{n}{\mathbf{u}_l' \mathbf{u}_l} \mathbf{u}_l' \mathbf{W}^k \mathbf{u}_l$$

where $\mathbf{u}_l \stackrel{\text{iid}}{\sim} N(\mathbf{0}, \mathbf{I}_n)$ and M is the total number of Monte Carlo samples.

When you apply these two approximations, it is often assumed that the maximum eigenvalue of $\mathbf{W}$ equals 1 and the minimum eigenvalue of $\mathbf{W}$ is greater than or equal to -1 (see LeSage and Pace 2009, and the references therein). One way to satisfy this assumption is to use a row-standardized spatial weights matrix that is similar to a symmetric matrix. If the spatial weights matrix is not symmetric or similar to a symmetric matrix, it becomes more difficult to apply Chebyshev approximation and thus requires extra care (LeSage and Pace 2009).

When you request an approximation to the Jacobian, the choices that you need to make might include the approximation method to use (that is, Taylor or Chebyshev approximation), the order of series q , and the number of Monte Carlo samples (that is, M). Your choice can be accommodated through the **APPROXIMATION=** option in the PROC CSPATIALREG statement. For the approximation method, you can use the keyword TAYLOR in the APPROXIMATION= option to request Taylor approximation. Otherwise, the approximation method defaults to Chebyshev approximation. You specify ORDER= q in the APPROXIMATION= option to request a series of order q when approximating the log-determinant. In addition, you specify NMC= M in the APPROXIMATION= option to request M Monte Carlo samples to be drawn when approximating the traces of powers of $\mathbf{W}$. In addition, you can use the SEED= suboption of the APPROXIMATION= option to specify an integer seed for a random number generator to replicate your analysis.

Impact Estimation

For models in PROC CSPATIALREG that include endogenous interactions or exogenous interactions, the interpretation of marginal effects is not as straightforward as it is in the classical linear regression model. For example, consider the following SDM model:

$$\mathbf{y} = \rho \mathbf{W} \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{W} \mathbf{X} \boldsymbol{\theta} + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma^2 \mathbf{I})$.

For an explanatory variable x_p in the model,

$$\frac{\partial y_i}{\partial x_{jp}} = [\mathbf{A}^{-1}(\beta_p \mathbf{I}_n + \theta_p \mathbf{W})]_{ij}$$

where $\mathbf{A} = \mathbf{I} - \rho\mathbf{W}$ and $[\mathbf{B}]_{ij}$ denotes the (i, j) th element of the matrix $\mathbf{B}$ for $i, j = 1, 2, \dots, n$.

According to $\frac{\partial y_i}{\partial x_{jp}}$, you can see that the impact of changes in x_p at a location j on the dependent variable differ from one location to another. To quantify the marginal effects of an explanatory variable x_p in the model, LeSage and Pace (2009) propose the average direct impacts ($\bar{M}(p)_{\text{direct}}$), the average total impacts ($\bar{M}(p)_{\text{total}}$), and the average indirect impacts ($\bar{M}(p)_{\text{indirect}}$) as follows:

$$\begin{aligned}\bar{M}(p)_{\text{direct}} &= \frac{1}{n} \sum_{i=1}^n [\mathbf{A}^{-1}(\beta_p \mathbf{I}_n + \theta_p \mathbf{W})]_{ii} \\ \bar{M}(p)_{\text{total}} &= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^n [\mathbf{A}^{-1}(\beta_p \mathbf{I}_n + \theta_p \mathbf{W})]_{ij} \\ \bar{M}(p)_{\text{indirect}} &= \bar{M}(p)_{\text{total}} - \bar{M}(p)_{\text{direct}}\end{aligned}$$

In addition, LeSage and Pace (2009) describe how to use Monte Carlo simulation to quantify uncertainty that is associated with the three impact estimates. First, the empirical distribution for parameters (the regression coefficients and, if applicable, the spatial dependence parameter ρ) is assumed to be a multivariate normal distribution where the mean vector contains the point estimates and the covariance matrix is estimated based on the maximum likelihood estimation. Second, M samples are drawn from the empirical distribution and for each sample being drawn, $\bar{M}(p)_{\text{direct}}$, $\bar{M}(p)_{\text{total}}$, and $\bar{M}(p)_{\text{indirect}}$ are computed. Based on these random samples, the sample mean and standard deviation for each of three impact estimates can be computed.

When $\bar{M}(p)_{\text{direct}}$, $\bar{M}(p)_{\text{total}}$, and $\bar{M}(p)_{\text{indirect}}$ are computed, the inverse of $\mathbf{A} = \mathbf{I}_n - \rho\mathbf{W}$ needs to be computed for each random sample drawn. Since the exact inversion of $\mathbf{A}$ can be computationally intensive, approximation can be helpful when it is applicable. When the spectral radius of $\rho\mathbf{W}$ is less than 1, $\mathbf{A}^{-1}$ can be approximated by the partial sums of Neumann series as

$$(\mathbf{I}_n - \rho\mathbf{W})^{-1} = \sum_{o=0}^{+\infty} (\rho\mathbf{W})^o \approx \mathbf{I}_n + \rho\mathbf{W} + \dots + \rho^o \mathbf{W}^o$$

The IMPACT option in the PROC CSPATIALREG statement computes $\bar{M}(p)_{\text{direct}}$, $\bar{M}(p)_{\text{total}}$, and $\bar{M}(p)_{\text{indirect}}$ by using the Monte Carlo simulation that is proposed by LeSage and Pace (2009). In particular, you specify NMC= M in the IMPACT option to request that M samples be drawn from the empirical distribution of the relevant model parameters. You use the ORDER= O suboption of the IMPACT option to specify the highest order of Neumann series to approximate $\mathbf{A}^{-1}$. You use the SEED= suboption of the IMPACT option to specify an integer seed for a random number generator to replicate your results.

Impact estimation is not supported for models that include any of the following: a classification variable, interactions between regressions, a BOUNDS statement, or a RESTRICT statement. In addition, Monte Carlo simulation is not needed to compute summary statistics for $\bar{M}(p)_{\text{direct}}$, $\bar{M}(p)_{\text{total}}$, and $\bar{M}(p)_{\text{indirect}}$ for linear, SLX, SEM, SDEM, SMA, and SDMA models because a closed-form solution is available ($\bar{M}(p)_{\text{direct}} = \beta_p$, $\bar{M}(p)_{\text{total}} = \beta_p + \theta_p$, and $\bar{M}(p)_{\text{indirect}} = \theta_p$). In these models, any NMC=, ORDER=, and SEED= suboptions that you specify in the IMPACT option are ignored. When you specify the IMPACT option for SAR, SDM, SAC, SDAC, SARMA, and SDARMA models, the spatial weights matrix $\mathbf{W}$ must be row-standardized. Moreover, the restriction $-1 < \rho < 1$ is imposed for SAR and SDM models and the restrictions $-1 < \rho < 1$ and $-1 < \lambda < 1$ are imposed for SAC, SDAC, SARMA, and SDARMA models. With the restriction $-1 < \rho < 1$, the average total impacts $\bar{M}(p)_{\text{total}}$ can be simplified as

$$\bar{M}(p)_{\text{total}} = \frac{1}{1 - \rho}(\beta_p + \theta_p)$$

Multiple Model Comparison

This section describes how you can fit multiple models, compare their corresponding fit statistic, and identify the model that provides the best fit to the data according to some criteria, such as AIC and SBC.

To fit multiple models, you can specify the `TYPE=ALL` or `TYPE=AUTO` option in the `MODEL` statement or `PROC CSPATIALREG` statement. The `TYPE=AUTO` option enables you to fit a selected subset of models among all models that `PROC CSPATIALREG` supports.

- If you specify the `TYPE=AUTO` option without the `SPATIALEFFECTS` statement, each of the seven available models—`CAR`, `LINEAR`, `SAC`, `SAR`, `SARMA`, `SEM`, and `SMA`—is fit using the regressors that you specify in the `MODEL` statement.
- If you specify both the `TYPE=AUTO` option and `SPATIALEFFECTS` statement, each of the available seven models—`CAR2`, `SDAC`, `SDARMA`, `SDM`, `SDMA`, and `SLX`—is fit using both the regressors that you specify in the `MODEL` statement and the spatial lagged regressors that you specify in the `SPATIALEFFECTS` statement, respectively.

The `TYPE=ALL` option enables you to fit all fourteen models that `PROC CSPATIALREG` supports.

- If you specify the `TYPE=ALL` option without the `SPATIALEFFECTS` statement, `PROC CSPATIALREG` fits seven models. In this case, each of the seven models—`CAR`, `LINEAR`, `SAC`, `SAR`, `SARMA`, `SEM`, and `SMA`—is fit using the regressors that you specify in the `MODEL` statement. This is equivalent to specifying the `TYPE=AUTO` option without the `SPATIALEFFECTS` statement.
- If you specify both the `TYPE=ALL` option and the `SPATIALEFFECTS` statement, `PROC CSPATIALREG` fits all fourteen models. On the one hand, each of the seven models—`CAR`, `LINEAR`, `SAC`, `SAR`, `SARMA`, `SEM`, and `SMA`—is fit using only the regressors that you specify in the `MODEL` statement. On the other hand, another seven models—`CAR2`, `SDAC`, `SDARMA`, `SDM`, `SDMA`, and `SLX`—are fit using both the regressors and the spatial lagged regressors that you specify in the `MODEL` statement and `SPATIALEFFECTS` statement, respectively.

When you fit multiple models by specifying the `TYPE=ALL` or `TYPE=AUTO` option, the following syntax is not supported:

- `BOUNDS`, `INIT`, `RESTRICT`, and `TEST` statements
- `APPROXIMATION=` option in the `PROC CSPATIALREG` statement

Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements

This section describes how you can refer to the parameters in the MODEL and SPATIALEFFECTS statements when you use the RESTRICT, TEST, BOUNDS, or INIT statement. The following examples use the RESTRICT statement, but the same remarks apply to naming parameters when you use the TEST, BOUNDS, or INIT statement. The names of the parameters are displayed in the “Parameter Estimates” table if you specify the option. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)'), then you must use the internal name of the parameter to refer to that parameter in the RESTRICT, TEST, BOUNDS, or INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

To impose a restriction on a parameter that is related to a regressor in the MODEL statement, you simply use the name of the regressor itself. Suppose that your model is

```
model y = x1-x3 / type=SAR;
```

where x1–x3 are continuous variables. Suppose you want to restrict the parameter that is associated with the regressor x3 to be greater than 1.7. You should provide the following statement:

```
RESTRICT x3 > 1.7;
```

To impose a restriction on a parameter that is associated with a regressor in the SPATIALEFFECTS statement, you can form the name of the parameter by adding the prefix W_ to the name of the regressor. Suppose that your MODEL and SPATIALEFFECTS statements are as follows:

```
model y = x1-x3 / type=SAR;  
spatialeffects x1 x2 x3;
```

Suppose you want to restrict the parameter that is related to the x3 regressor in the SPATIALEFFECTS statement to be less than 1.0. You should refer to the parameter as W_x3 and provide the following statement:

```
RESTRICT W_x3 < 1.0;
```

Even though the regressor x3 appears in both the MODEL and SPATIALEFFECTS statements, the parameter that is associated with x3 in the MODEL statement is, of course, different from the parameter that is associated with x3 in the SPATIALEFFECTS statement. Thus, when you use the name of a regressor in a RESTRICT statement without any prefix, it refers to the parameter that is associated with that regressor in the MODEL statement. Meanwhile, when you use the name of a regressor in a RESTRICT statement with the prefix W_, it refers to the parameter that is associated with that regressor in the SPATIALEFFECTS statement. Note that the intercept is not included in the SPATIALEFFECTS statement.

Referring to Class Level Parameters

When your MODEL includes a classification variable, you can impose restrictions on the parameters associated with each of the levels that are related to the classification variable as follows.

Suppose that your classification variable is named *C* and has three levels: 0, 1, and 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 C;
```

Adding a classification variable as a regressor to your model introduces additional parameters to your model, each of which is associated with one of the levels of the classification variable. You can form the name of the parameter that is associated with a particular level of your classification variable by inserting the underscore character between the name of the classification variable and the value of the level. Thus, to restrict the parameter that is associated with level 0 of the classification variable *C* to always be greater than 0.7, you refer to the parameter as *C_0* and provide the following statement:

```
RESTRICT C_0 > 0.7;
```

Referring to Parameters Associated with Interactions between Regressors

When a regressor in your model involves an interaction between other regressors, you can impose restrictions on the parameters that are associated with the interaction as described in this section.

Suppose you have the following model:

```
model y = x1 x2 x3*x4;
```

You can form the name of the parameter that is associated with the interaction regressor *x3*x4* by replacing the multiplication sign with an underscore. Thus, *x3_x4* refers to the parameter that is associated with the interaction regressor *x3*x4*.

You refer to interactions between regressors and CLASS variables in exactly the same way. Suppose you have a CLASS variable named *C* that has three levels (0, 1, and 2), and your model is the following:

```
class C;
model y = x1 x2 C*x3;
```

The interaction between the continuous variable *x3* and the CLASS variable *C* introduces three additional parameters, which are named *x3_C_0*, *x3_C_1*, and *x3_C_2*. Note that, although the order of the terms in the interaction is *C* followed by *x3*, the name of the parameter that is associated with the interaction is formed by placing the name of the continuous variable *x3* first, followed by an underscore, followed by the name of the CLASS variable *C*, then another underscore, and then the level value. Once again, depending on the parameterization that you specify in your CLASS statement, for each interaction in your model that involves a CLASS variable, one of the parameters that are associated with that interaction can be dropped from your model before optimization.

The name of a parameter that is associated with a nested interaction is formed in a slightly different way. Suppose you have a CLASS variable named *C* that has three levels (0, 1, and 2) and your model is the following:

```
class C;
model y = x1 x2 x3 (C);
```

The nested interaction between the continuous variable `x3` and the CLASS variable `C` introduces three additional parameters, which are named `x3_C__0`, `x3_C__1`, and `x3_C__2`. Note how the name in each case was formed from the name of the regressor by replacing the left and right parentheses with underscores and then appending another underscore, followed by the level value.

Referring to Class Level Parameters with Negative Values

When the value of a level is a negative number, you must replace the minus sign with an underscore when you form the name of the parameter that is associated with that particular level of the classification variable. For example, suppose your classification variable is named `D` and has four levels: `-1`, `0`, `1`, and `2`. Suppose that your model is the following:

```
class D;
model y = x1 x2 D;
```

To restrict the parameter that is associated with level `-1` of the classification variable `D` to always be less than `0.4`, you refer to the parameter as `D__1` (note that there are two underscores in this parameter name: one to connect the name of the classification variable to its value and the other to replace the minus sign in the value itself) and provide the following statement:

```
RESTRICT D__1 < 0.4;
```

Dropping a Class Level Parameter to Avoid Collinearity

Depending on the parameterization that you impose on your classification variable, one of the parameters that are associated with its levels might be dropped from your model before optimization in order to avoid collinearity. For example, when the default parameterization GLM is imposed, the parameter that is associated with the last level of your classification variable is dropped before optimization. If you attempt to impose a restriction on a dropped parameter by using the `RESTRICT` statement, PROC CSPATIALREG issues an error message in the log.

For example, suppose again that your classification variable is named `C` and has three levels: `0`, `1`, and `2`. Suppose that your model is the following:

```
class C;
model y = x1 x2 C;
```

Because no additional options are specified in the CLASS statement, GLM parameterization is assumed. This means that the parameter named `C_2` (which is the parameter associated with the last level of your classification variable) is dropped from your model before the optimizer is invoked. Therefore, you get an error if you attempt to restrict the `C_2` parameter in any way by referring to it in a `RESTRICT` statement. For example, the following `RESTRICT` statement will generate an error:

```
RESTRICT C_2 < 0.3;
```

Referring to Implicit Parameters

For all models in PROC CSPATIALREG, one or more implicit parameters are added to your model before optimization. You can impose restrictions on these implicit parameters as follows.

If you have a linear model or an SLX model, the `_sigma2` parameter is added to your model. For the SDM or SAR model, the `_rho` and `_sigma2` parameters are added to your model. For the SEM or SDEM, the `_lambda` and `_sigma2` parameters are added to your model.

Whenever your model type dictates the addition of one or more of these implicit parameters, you can impose restrictions on the implicit parameters by referring to them by name. For example, assuming that your model type implies the existence of the `_rho` parameter, you can restrict `_rho` to be greater than 0 as follows:

```
RESTRICT _rho > 0.0;
```

Computational Resources

The time and memory that PROC CSPATIALREG requires are proportional to the number of parameters in the model and the number of observations in the data that you are analyzing. Less time and memory are required for smaller models and fewer observations. When you run PROC CSPATIALREG in the high-performance distributed environment, the amount of time that is required is also affected by the number of nodes and the number of threads per node when the optimization task is distributed.

The method that you use to calculate the variance-covariance matrix and the optimization method also affect the time and memory resources. All optimization methods available through the `METHOD=` option have similar memory use requirements. The processing time might differ among methods, depending on the number of iterations and functional calls that are needed. The data is read into memory to save processing time. If not enough memory is available to hold the data, the CSPATIALREG procedure stores the data in a utility file on disk and rereads the data as needed from this file, substantially increasing the execution time of the procedure. The gradient and the variance-covariance matrix must be held in memory. If the model has p parameters, including the intercept, then you need at least $8 * (p + p * (p + 1)/2)$ bytes of memory. The processing time is also a function of the number of iterations that are needed to converge to a solution for the model parameters. The number of iterations that are needed cannot be known in advance. You can use the `MAXITER=` option to limit the number of iterations that PROC CSPATIALREG performs. You can alter the convergence criteria by using the nonlinear optimization options available in the PROC CSPATIALREG statement. For a list of all the nonlinear optimization options, see the section “[Optimization Control Options](#)” on page 649.

Covariance Matrix Types

The CSPATIALREG procedure enables you to specify the estimation method for the covariance matrix. You have three choices, which you can specify in either the PROC CSPATIALREG or MODEL statement: the COVEST=HESSIAN option estimates the covariance matrix based on the inverse of the Hessian matrix, the COVEST=OP option uses the outer product of gradients, and the COVEST=QML option produces the covariance matrix based on both the Hessian and outer product matrices. By default, COVEST=HESSIAN.

Although all three methods produce asymptotically equivalent results, they differ in computational intensity and produce results that might differ in finite samples. The COVEST=OP option provides the covariance matrix that is usually the easiest to compute. In some cases, the OP approximation is considered more efficient than the Hessian or QML approximation because it contains fewer random elements. The QML approximation is computationally the most complex, because it requires both the outer product of gradients and the Hessian matrix. In most cases, the OP or Hessian approximation is preferred to the QML approximation. You need to use the QML approximation in some cases when the model is misspecified and the information matrix equality does not hold.

For SDMs, SEMs, SDEMs, and SAR models, only COVEST=HESSIAN is supported.

Displayed Output

PROC CSPATIALREG produces the following displayed output.

Class Level Information

If you specify the CLASS statement, the CSPATIALREG procedure displays a table that contains the following information:

- CLASS variable name
- number of levels of the CLASS variable
- list of values of the CLASS variable

Model Fit Summary

When you fit a single model in PROC CSPATIALREG, the “Model Fit Summary” table contains the following information:

- dependent variable name
- number of observations used
- data table name
- name of the spatial weights data table (specified by the WMAT= option)
- spatial ID variable

- type of model that was fit
- log-likelihood value at the solution
- maximum absolute gradient at the solution
- number of iterations
- AIC value at the solution (a smaller value indicates a better fit)
- SBC value at the solution (a smaller value indicates a better fit)

Below the “Model Fit Summary” table is a statement about whether the algorithm successfully converged.

If you fit multiple models by using the TYPE=ALL or TYPE=AUTO option, the “Model Fit Summary” table contains the following information:

- dependent variable name
- number of observations used
- data table name
- name of the spatial weights data table (specified by the WMAT= option)
- spatial ID variable
- type of model that was fit

Parameter Estimates

The “Parameter Estimates” table displays the estimates of the model parameters, which include both regression coefficients and other, model-specific parameters. For the CAR, SAR, and SDM models, the table also displays estimates of the spatial autoregressive coefficient ρ and the variance of the error terms σ^2 . For the SDEM, SDMA, SEM, and SMA models, it also displays estimates of the spatial autocorrelation coefficient λ and the variance of the error terms σ^2 . For the linear and SLX models, the table also displays estimates of the variance of the error terms σ^2 . For SAC, SARMA, SDARMA, and SDAC models, it also displays estimates of the spatial autoregression coefficient ρ , the spatial autocorrelation coefficient λ , and the variance parameter σ^2 .

The internal name of the spatial autoregressive coefficient ρ is “_rho”. The t statistic that “Parameter Estimates” table displays for “_rho” is a test of the spatial autoregressive coefficient. In addition, “_lambda” is the internal name of the spatial autocorrelation coefficient λ , and “_sigma2” is the internal name of the variance parameter σ^2 .

Covariance of Parameter Estimates

If you specify the COVB option in the MODEL or PROC CSPATIALREG statement, the CSPATIALREG procedure displays the estimated covariance matrix, defined as the inverse of the information matrix, that is evaluated at the final iteration.

Correlation of Parameter Estimates

If you specify the CORRB option in the MODEL or PROC CSPATIALREG statement, the CSPATIALREG procedure displays the estimated correlation matrix. It is based on the Hessian matrix that is used in the final iteration.

OUTPUT OUT= Data Table

The OUTPUT statement creates a new data table that contains various estimates that you specify. You can request that the output data table contain the estimates of $\mathbf{x}'_i\boldsymbol{\beta}$, the expected value of the response variable, and the residual.

Because of potential space limitations on the client workstation, the data table that the OUTPUT statement creates does not contain the variables in the input data table.

ODS Table Names

PROC CSPATIALREG assigns a name to each table that it creates. You can use these names to denote the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in [Table 13.3](#).

Table 13.3 ODS Tables Produced by PROC CSPATIALREG

Table Name	Description	Statement	Option
ClassInfo	Level information from the CLASS statement	CLASS	Default
ConvergenceStatus	Convergence status	MODEL	Default
CorrB	Correlation of parameter estimates	MODEL	CORRB
CovB	Covariance of parameter estimates	MODEL	COVB
FitModelSummary	Summary of nonlinear estimation	MODEL	Default
ImpactEstimates	Summary of direct, indirect, and total impact estimates	PROC CSPATIALREG	IMPACT
InitialParameterEstimates	Initial parameter estimates	PROC CSPATIALREG or MODEL	PRINTALL
IterationHistory	Optimization iteration history	PROC CSPATIALREG or MODEL	PRINTALL
ModelComparison	Model comparison for multiple models	PROC CSPATIALREG or MODEL	TYPE=AUTO or TYPE=ALL
OptimizationResults	Optimization results	PROC CSPATIALREG or MODEL	PRINTALL

Table 13.3 *continued*

Table Name	Description	Statement	Option
ParameterEstimates	Parameter estimates	MODEL	Default
TestResults	Hypothesis test statistics	TEST	Default
TimingDetails	Detailed summary of time used for all phases of execution	PROC CSPATIALREG	PRINTTIMING=(DETAILS)
TimingSummary	Summary of time used for main phases of execution	PROC CSPATIALREG	PRINTTIMING
ZNLOSettings	Nonlinear optimizer settings	PROC CSPATIALREG or MODEL	PRINTALL

Examples: CSPATIALREG Procedure

Example 13.1: Columbus Crime Data

Data Description and Objective

The data set `CrimeOH` contains data from Columbus, Ohio, about numbers of crimes (including residential burglaries and vehicle thefts) and possible determinants of these crimes. This data set comes from Anselin (1988).

The variable `Crime` represents the number of crimes in 49 Columbus neighborhoods. Additional variables in the data set that you want to evaluate as determinants of these crimes include `Income` (household income by \$1,000) and `HValue` (housing value by \$1,000). Summary statistics for these variables are computed by the following statements and presented in [Output 13.1.1](#):

```
proc means data=mylib.CrimeOH;
    var crime income hvalue;
run;
```

Output 13.1.1 Summary Statistics

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
crime	49	35.1288367	16.7320385	0.1780000	68.8920000
income	49	14.3749388	5.7033781	4.4770000	31.0700000
hvalue	49	38.4362245	18.4660693	17.9000000	96.4000000

The spatial relationships among the 49 neighborhoods are summarized using the first-order neighbor contiguity matrix, contained in the `CrimeWmat` data set. This data set also comes from Anselin (1988).

Spatial Autoregressive (SAR) Model

The following statements fit a spatial autoregressive (SAR) model to the data by using the regressors Income and HValue:

```
/*-- SAR --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat NONORMALIZE;
    model crime=income hvalue / type=SAR;
    spatialid sid;
run;
```

In this example, the TYPE=SAR option in the MODEL statement specifies a SAR model. The NONORMALIZE option indicates that the spatial weights data table mylib.CrimeWmat should be used “as is” rather than be row-standardized. The parameter estimates for this model are shown in [Output 13.1.2](#). According to the results, the spatial autoregressive coefficient ρ is positive and significant at the 0.05 level on the basis of the t statistic. This indicates a positive spatial dependence in the data.

Output 13.1.2 Parameter Estimates of SAR Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	45.076368	7.870002	5.73	<.0001
income	1	-1.031514	0.328375	-3.14	0.0017
hvalue	1	-0.265923	0.088211	-3.01	0.0026
_rho	1	0.431032	0.123586	3.49	0.0005
_sigma2	1	95.470316	19.499458	4.90	<.0001

Spatial Durbin Model (SDM)

To fit a spatial Durbin model (SDM), you specify the SPATIALEFFECTS statement and specify the TYPE=SAR option in the MODEL statement or the PROC CSPATIALREG statement. In this example, the spatial lags of the regressors Income and HValue are considered in the SDM.

The following statements fit an SDM to the CrimeOH data:

```
/*-- SDM --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat NONORMALIZE;
    model crime=income hvalue / type=SAR;
    spatialeffects income hvalue;
    spatialid sid;
run;
```

The parameter estimates are shown in [Output 13.1.3](#). As in the SAR model, the spatial autoregressive coefficient ρ in the SDM is positive and significant at the 0.05 level on the basis of the t statistic, indicating a positive spatial dependence in the data.

Output 13.1.3 Parameter Estimates of SDM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	42.802410	13.923873	3.07	0.0021
income	1	-0.914200	0.336418	-2.72	0.0066
hvalue	1	-0.293746	0.088851	-3.31	0.0009
W_income	1	-0.519612	0.594739	-0.87	0.3823
W_hvalue	1	0.245718	0.176843	1.39	0.1647
_rho	1	0.426506	0.167485	2.55	0.0109
_sigma2	1	91.768262	18.904590	4.85	<.0001

To avoid potential collinearity with the intercept term in the MODEL statement, the SPATIALEFFECTS statement always excludes the intercept term. This means that only variables that you explicitly specify in the SPATIALEFFECTS statement are used to construct spatial lag of covariate effects.

Spatial Error Model (SEM)

To fit a spatial error model (SEM), specify the TYPE=SEM option in the MODEL statement or the PROC CSPATIALREG statement.

The following statements fit an SEM to the CrimeOH data:

```

/*-- SEM --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SEM;
  spatialid sid;
  restrict _lambda<0.99;
run;

```

The parameter estimates are shown in [Output 13.1.4](#). According to this output, the p -value for the spatial autocorrelation parameter λ is 0.0002 on the basis of the t statistic. The results indicate a significant positive dependence in the error term.

Output 13.1.4 Parameter Estimates of SEM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	59.890244	5.882911	10.18	<.0001
income	1	-0.941190	0.370165	-2.54	0.0110
hvalue	1	-0.302255	0.090529	-3.34	0.0008
_lambda	1	0.561879	0.152369	3.69	0.0002
_sigma2	1	95.523372	20.017061	4.77	<.0001

Spatial Durbin Error Model (SDEM)

To fit a spatial Durbin error model (SDEM), use the SPATIALEFFECTS statement and specify the TYPE=SEM option in the MODEL statement or the PROC CSPATIALREG statement. In this example, the spatial lags of the regressors Income and HValue are considered in the SDEM.

The following statements fit an SDEM to the CrimeOH data:

```
/*-- SDEM --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SEM;
  spatialeffects income hvalue;
  spatialid sid;
run;
```

The parameter estimates are shown in [Output 13.1.5](#).

Output 13.1.5 Parameter Estimates of SDEM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	73.540497	8.860552	8.30	<.0001
income	1	-1.051696	0.322416	-3.26	0.0011
hvalue	1	-0.275607	0.091148	-3.02	0.0025
W_income	1	-1.156543	0.592883	-1.95	0.0511
W_hvalue	1	0.111750	0.202354	0.55	0.5808
_lambda	1	0.425413	0.173824	2.45	0.0144
_sigma2	1	92.521238	19.084926	4.85	<.0001

Spatial Moving Average (SMA) Model

To fit an SMA model, specify the TYPE=SMA option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements fit an SMA model to the CRIMEOH data:

```
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SMA;
  spatialid sid;
run;
```

The parameter estimates are shown in [Output 13.1.6](#). According to the results, the spatial coefficient λ is negative and significant at the 0.05 level on the basis of the t statistic. This indicates a positive spatial dependence in the errors.

Output 13.1.6 Parameter Estimates of SMA Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	59.252971	5.934861	9.98	<.0001
income	1	-0.921806	0.363482	-2.54	0.0112
hvalue	1	-0.287393	0.086880	-3.31	0.0009
_lambda	1	-0.799089	0.277861	-2.88	0.0040
_sigma2	1	117.731990	26.373322	4.46	<.0001

Spatial Durbin Moving Average (SDMA) Model

To fit an SDMA model, use the SPATIALEFFECTS statement and specify the TYPE=SMA option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements consider the spatial lags of the regressors INCOME and HVALUE and fit an SDMA model to the CRIMEOH data:

```
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SMA;
  spatialeffects income hvalue;
  spatialid sid;
run;
```

Partial output is shown in [Output 13.1.7](#). According to the results, the spatial coefficient λ is negative and significant at the 0.05 level on the basis of the t statistic. This indicates a positive spatial dependence in the errors.

Output 13.1.7 Parameter Estimates of an SDMA Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	73.944211	9.083977	8.14	<.0001
income	1	-1.065635	0.312045	-3.42	0.0006
hvalue	1	-0.266840	0.092400	-2.89	0.0039
W_income	1	-1.074757	0.584955	-1.84	0.0662
W_hvalue	1	0.067568	0.209867	0.32	0.7475
_lambda	1	-0.642124	0.296638	-2.16	0.0304
_sigma2	1	103.502516	22.487027	4.60	<.0001

Spatial Autoregressive Confused (SAC) Model

To fit an SAC model, specify the TYPE=SAC option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements fit the SAC model to the CRIMEOH data:

```
proc cspatialreg data=mylib.CrimeOH wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SAC;
  spatialid sid;
  restrict _lambda<0.99;
run;
```

The parameter estimates are shown in [Output 13.1.8](#).

Output 13.1.8 Parameter Estimates of SAC Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	47.778144	9.278161	5.15	<.0001
income	1	-1.025824	0.333984	-3.07	0.0021
hvalue	1	-0.281635	0.093361	-3.02	0.0026
_rho	1	0.368158	0.181115	2.03	0.0421
_lambda	1	0.166521	0.298117	0.56	0.5765
_sigma2	1	95.584355	19.469071	4.91	<.0001

Spatial Durbin Autoregressive Confused (SDAC) Model

To fit an SDAC model, use the SPATIALEFFECTS statement and specify the TYPE=SAC option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements consider the spatial lags of the regressors INCOME and HVALUE and fit an SDAC model to the CRIMEOH data:

```
proc cspatialreg data=mylib.CrimeOH wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SAC;
  spatialeffects income hvalue;
  spatialid sid;
  restrict _lambda<0.99;
run;
```

The parameter estimates are shown in [Output 13.1.9](#).

Output 13.1.9 Parameter Estimates of SDAC Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	50.908729	31.033511	1.64	0.1009
income	1	-0.950768	0.353881	-2.69	0.0072
hvalue	1	-0.286432	0.091261	-3.14	0.0017
W_income	1	-0.692491	0.839213	-0.83	0.4093
W_hvalue	1	0.208611	0.222514	0.94	0.3485
_rho	1	0.315691	0.413932	0.76	0.4457
_lambda	1	0.154954	0.473634	0.33	0.7435
_sigma2	1	93.184031	19.197723	4.85	<.0001

Spatial Autoregressive Moving Average (SARMA) Model

To fit a SARMA model, specify the TYPE=SARMA option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements fit a SARMA model to the CRIMEOH data:

```
proc cspatialreg data=mylib.CrimeOH wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SARMA;
  spatialid sid;
run;
```

The parameter estimates are shown in [Output 13.1.10](#).

Output 13.1.10 Parameter Estimates of SARMA Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	48.973247	9.602039	5.10	<.0001
income	1	-1.016359	0.337215	-3.01	0.0026
hvalue	1	-0.287458	0.093079	-3.09	0.0020
_rho	1	0.336281	0.204317	1.65	0.0998
_lambda	1	-0.271945	0.426840	-0.64	0.5241
_sigma2	1	97.992936	21.253768	4.61	<.0001

Spatial Durbin Autoregressive Moving Average (SDARMA) Model

To fit an SDARMA model, use the SPATIALEFFECTS statement and specify the TYPE=SARMA option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements consider the spatial lag of the regressor HVALUE and fit an SDARMA model to the CRIMEOH data:

```
proc cspatialreg data=mylib.CrimeOH wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SARMA;
  spatialeffects hvalue;
  spatialid sid;
run;
```

The parameter estimates are shown in [Output 13.1.11](#).

Output 13.1.11 Parameter Estimates of SDARMA Model

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	30.280526	16.779263	1.80	0.0711
income	1	-0.960400	0.332845	-2.89	0.0039
hvalue	1	-0.293353	0.090767	-3.23	0.0012
W_hvalue	1	0.239748	0.206784	1.16	0.2463
_rho	1	0.590709	0.234325	2.52	0.0117
_lambda	1	0.191790	0.455187	0.42	0.6735
_sigma2	1	89.039641	18.819474	4.73	<.0001

Linear Regression Model

To fit a linear model, specify the TYPE=LINEAR option in the MODEL statement or the PROC CSPATIALREG statement.

The following statements fit a linear model to the CrimeOH data:

```
/*-- LINEAR --*/
proc cspatialreg data=mylib.CrimeOH;
  model crime=income hvalue / type=LINEAR;
run;
```

Partial output is shown in [Output 13.1.12](#).

Output 13.1.12 Parameter Estimates of Linear Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	68.618863	4.588210	14.96	<.0001
income	1	-1.597304	0.323739	-4.93	<.0001
hvalue	1	-0.273931	0.099989	-2.74	0.0062
_sigma2	1	122.751696	24.799493	4.95	<.0001

Spatial Lag of X (SLX) Model

To fit a spatial lag of X (SLX) model, use the SPATIALEFFECTS statement and specify the TYPE=LINEAR option in the MODEL statement or the PROC CSPATIALREG statement.

The following statements consider the spatial lags of the regressors Income and HValue and fit an SLX model to the CrimeOH data:

```

/*-- SLX --*/
proc cspatialreg data=mylib.CrimeOH wmat=mylib.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=LINEAR;
  spatialeffects income hvalue;
  spatialid sid;
run;

```

The parameter estimates are shown in [Output 13.1.13](#).

Output 13.1.13 Parameter Estimates of SLX Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	75.028184	6.279950	11.95	<.0001
income	1	-1.109020	0.354232	-3.13	0.0017
hvalue	1	-0.289734	0.096058	-3.02	0.0026
W_income	1	-1.370866	0.531889	-2.58	0.0100
W_hvalue	1	0.191785	0.189841	1.01	0.3124
_sigma2	1	107.292373	21.676329	4.95	<.0001

Conditional Autoregressive (CAR) Model

The following statements fit a conditional autoregressive (CAR) model to the data by including the two regressors—Income and HValue:

```

/*-- CAR --*/
proc cspatialreg data=mylib.CrimeOH wmat=mylib.CrimeCmat;
  model crime=income hvalue / type=CAR;
  spatialid sid;

```

```
run;
```

In this example, the TYPE=CAR option in the MODEL statement specifies a CAR model. By default, the spatial weights data table mylib.CrimeCmat is row-standardized. The parameter estimates for this model are shown in [Output 13.1.14](#). According to the results, the spatial autoregressive coefficient ρ is positive and significant at the 0.05 level on the basis of the t statistic. This indicates a positive spatial dependence in the data.

Output 13.1.14 Parameter Estimates of CAR Model

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	65.105998	5.240462	12.42	<.0001
income	1	-1.044047	0.360941	-2.89	0.0038
hvalue	1	-0.341777	0.099814	-3.42	0.0006
_rho	1	0.833485	0.154749	5.39	<.0001
_sigma2	1	418.230408	90.029974	4.65	<.0001

Example 13.2: Simulated Data Example

Data Description and Objective

This example uses two simulated data sets, SimData and SimW, to illustrate model fitting in PROC CSPATIALREG and to facilitate subsequent discussion.

The SimData data set contains 50 observations and five variables. The variable SID identifies each spatial unit in the data. The three explanatory variables are x1, x2, and x3. The dependent variable is y. The SimW data set defines the spatial contiguity for all 50 spatial units. The first column, SID, in the SimW data set identifies each spatial unit. The remaining entries in the SimW data set are binary and define whether two spatial units are neighbors. A value of 1 indicates that two spatial units are neighbors, and 0 indicates otherwise.

Summary statistics for all variables except SID in the SimData data set are computed by the following statements and presented in [Output 13.2.1](#):

```
proc means data=mylib.SimData;
  var x1 x2 x3 y;
run;
```

Output 13.2.1 Summary Statistics

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
x1	50	-0.0076329	1.0989504	-2.4523193	1.6539456
x2	50	-0.0829941	0.9671181	-2.5725767	2.4034547
x3	50	-0.0894387	0.9975304	-2.4470049	2.6720533
y	50	1.1569199	1.5687060	-1.9399423	4.7136835

The following statements fit a spatial autoregressive (SAR) model to the data by using three regressors, x1,

x2, and x3:

```
/*-- SAR --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW;
  model y=x1-x3 / type=SAR;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 13.2.2](#).

Output 13.2.2 Parameter Estimates of SAR Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.780650	0.098703	18.04	<.0001
x1	1	0.573329	0.047395	12.10	<.0001
x2	1	0.707048	0.057181	12.37	<.0001
x3	1	-0.902843	0.053314	-16.93	<.0001
_rho	1	-0.473713	0.063008	-7.52	<.0001
_sigma2	1	0.131509	0.026350	4.99	<.0001

To fit a spatial Durbin model (SDM) that includes exogenous interaction effects of x1, x2, and x3, submit the following statements:

```
/*-- SDM --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW;
  model y=x1-x3/ type=SAR;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 13.2.3](#).

Output 13.2.3 Parameter Estimates of SDM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932578	0.198882	9.72	<.0001
x1	1	0.548504	0.049806	11.01	<.0001
x2	1	0.686011	0.056266	12.19	<.0001
x3	1	-0.890161	0.053516	-16.63	<.0001
W_x1	1	0.172302	0.154018	1.12	0.2633
W_x2	1	0.023746	0.198557	0.12	0.9048
W_x3	1	-0.324808	0.228032	-1.42	0.1543
_rho	1	-0.639757	0.164651	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

If you want to fit another type of model, you need to change the value of the TYPE= option in the MODEL statement. As an example, to fit a spatial error model (SEM) instead of a SAR model to the data, you can use the following statements:

```
/*-- SEM --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW;
  model y=x1-x3 / type=SEM;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 13.2.4](#).

Output 13.2.4 Parameter Estimates of SEM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.166289	0.029514	39.52	<.0001
x1	1	0.487975	0.049086	9.94	<.0001
x2	1	0.634442	0.061776	10.27	<.0001
x3	1	-0.831250	0.054780	-15.17	<.0001
_lambda	1	-0.964826	0.132514	-7.28	<.0001
_sigma2	1	0.147434	0.031318	4.71	<.0001

The following statements fit a spatial autoregressive confused (SAC) model and spatial autoregressive moving average (SARMA) model to the data:

```
/*-- SAC --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW;
  model y=x1-x3 / type=SAC;
  spatialid SID;
run;

/*-- SARMA --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW;
  model y=x1-x3 / type=SARMA;
  spatialid SID;
run;
```

The parameter estimates for the SAC and SARMA models are shown in [Output 13.2.5](#) and [Output 13.2.6](#), respectively.

Output 13.2.5 Parameter Estimates of an SAC Model**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.745140	0.108969	16.01	<.0001
x1	1	0.574630	0.047481	12.10	<.0001
x2	1	0.699848	0.057547	12.16	<.0001
x3	1	-0.909416	0.053365	-17.04	<.0001
_rho	1	-0.449744	0.075425	-5.96	<.0001
_lambda	1	-0.235971	0.277450	-0.85	0.3950
_sigma2	1	0.128845	0.025938	4.97	<.0001

Output 13.2.6 Parameter Estimates of a SARMA Model**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.721937	0.111720	15.41	<.0001
x1	1	0.576752	0.047432	12.16	<.0001
x2	1	0.694100	0.058118	11.94	<.0001
x3	1	-0.910829	0.053061	-17.17	<.0001
_rho	1	-0.434840	0.079844	-5.45	<.0001
_lambda	1	0.393432	0.342912	1.15	0.2512
_sigma2	1	0.133287	0.027984	4.76	<.0001

Example 13.3: Compact Representation of a Spatial Weights Matrix

When a spatial weights matrix is sparse, you might want to provide its compact representation rather than the full matrix to the CSPATIALREG procedure. This example shows you how to use the compact representation of a spatial weights matrix in PROC CSPATIALREG. For illustration, the simulated data sets SimData and SimW from “[Example 13.2: Simulated Data Example](#)” on page 696 are used here. The compact representation of the spatial weights matrix in the SimW data set is created and saved in the data set SimW_Compact.

The first 10 observations in the SimW_Compact data set are shown in [Figure 13.3.1](#).

Output 13.3.1 SimW_Compact Data Set

Obs	SID	cSID	Value
1	L50	L45	1
2	L32	L35	1
3	L50	L25	1
4	L45	L36	1
5	L10	L8	1
6	L44	L48	1
7	L7	L6	1
8	L36	L15	1
9	L5	L19	1
10	L37	L46	1

To fit a spatial autoregressive (SAR) model, you can use the following statements:

```

/*-- SAR --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW_Compact;
  model y=x1-x3 / type=SAR;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 13.3.2](#).

Output 13.3.2 Parameter Estimates of SAR Model with Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates						
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t	
Intercept	1	1.780650	0.098703	18.04	<.0001	
x1	1	0.573329	0.047395	12.10	<.0001	
x2	1	0.707048	0.057181	12.37	<.0001	
x3	1	-0.902843	0.053314	-16.93	<.0001	
_rho	1	-0.473713	0.063008	-7.52	<.0001	
_sigma2	1	0.131509	0.026350	4.99	<.0001	

To fit a spatial error model (SEM) instead of a SAR model to the data, you can use the following statements:

```

/*-- SEM --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW_Compact;
  model y=x1-x3 / type=SEM;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 13.3.3](#).

Output 13.3.3 Parameter Estimates of SEM with Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.166289	0.029514	39.52	<.0001
x1	1	0.487975	0.049086	9.94	<.0001
x2	1	0.634442	0.061776	10.27	<.0001
x3	1	-0.831250	0.054780	-15.17	<.0001
_lambda	1	-0.964826	0.132514	-7.28	<.0001
_sigma2	1	0.147434	0.031318	4.71	<.0001

If you want to fit another type of model, you need to change the value of the TYPE= option in the MODEL statement. For example, to fit a spatial autoregressive moving average (SARMA) model and a spatial autoregressive confused (SAC) model to the data, you can use the following statements:

```

/*-- SARMA --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW_Compact;
  model y=x1-x3 / type=SARMA;
  spatialid SID;
run;

/*-- SAC --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW_Compact;
  model y=x1-x3 / type=SAC;
  spatialid SID;
run;

```

The parameter estimates for the SARMA and SAC models are shown in [Output 13.3.4](#) and [Output 13.3.5](#), respectively.

Output 13.3.4 Parameter Estimates of a SARMA Model with Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.721937	0.111720	15.41	<.0001
x1	1	0.576752	0.047432	12.16	<.0001
x2	1	0.694100	0.058118	11.94	<.0001
x3	1	-0.910829	0.053061	-17.17	<.0001
_rho	1	-0.434840	0.079844	-5.45	<.0001
_lambda	1	0.393432	0.342912	1.15	0.2512
_sigma2	1	0.133287	0.027984	4.76	<.0001

Output 13.3.5 Parameter Estimates of an SAC Model with Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.745140	0.108969	16.01	<.0001
x1	1	0.574630	0.047481	12.10	<.0001
x2	1	0.699848	0.057547	12.16	<.0001
x3	1	-0.909416	0.053365	-17.04	<.0001
_rho	1	-0.449744	0.075425	-5.96	<.0001
_lambda	1	-0.235971	0.277450	-0.85	0.3950
_sigma2	1	0.128845	0.025938	4.97	<.0001

Example 13.4: Taylor and Chebyshev Approximations

When you have a large data set (that is, the number of spatial units in your data is large), it becomes burdensome to fit some models. This is partly because all models except linear regression models require you to calculate the determinant of the matrix of a large size (such as $|\mathbf{I} - \rho\mathbf{W}|$ in a spatial autoregressive model). In these cases, using Taylor and Chebyshev approximations in the CSPATIALREG procedure can be helpful. PROC CSPATIALREG enables you to estimate spatial autoregressive, spatial Durbin, spatial error, and spatial Durbin error models that have a relatively large spatial weights matrix by using these two approximations. By using the two small data sets SimData and SimW from “[Example 13.2: Simulated Data Example](#)” on page 696, examples in this section show how you can invoke the two approximations in PROC CSPATIALREG.

The following statements fit a spatial autoregressive (SAR) model by using Chebyshev approximation:

```
/*-- SAR Model : full W --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW approximation=(ORDER=10);
  model y=x1-x3 / type=SAR;
  spatialid SID;
run;
```

You use the APPROXIMATION= option to specify which approximation technique to use. By default, Chebyshev approximation is used. The ORDER=10 option specifies a 10th-order Chebyshev polynomial to be used for approximation. The parameter estimates for this model are shown in [Output 13.4.1](#). Note that the spatial weights matrix in the SimW data set is a full matrix. Compared with [Output 13.2.2](#), Chebyshev approximation yields very similar parameter estimates.

Output 13.4.1 Parameter Estimates of SAR Model with Chebyshev Approximation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.780650	0.098703	18.04	<.0001
x1	1	0.573329	0.047395	12.10	<.0001
x2	1	0.707048	0.057181	12.37	<.0001
x3	1	-0.902843	0.053314	-16.93	<.0001
_rho	1	-0.473713	0.063008	-7.52	<.0001
_sigma2	1	0.131509	0.026350	4.99	<.0001

Using the compact representation of the spatial weights matrix, you can submit the following statements to fit a SAR model by using Chebyshev approximation:

```
/*-- SAR Model : compact form of W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW_Compact approximation=(ORDER=10);
  model y=x1-x3 / type=SAR;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 13.4.2](#), which is identical to [Output 13.4.1](#).

Output 13.4.2 Parameter Estimates of SAR Model with Chebyshev Approximation and Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.780650	0.098703	18.04	<.0001
x1	1	0.573329	0.047395	12.10	<.0001
x2	1	0.707048	0.057181	12.37	<.0001
x3	1	-0.902843	0.053314	-16.93	<.0001
_rho	1	-0.473713	0.063008	-7.52	<.0001
_sigma2	1	0.131509	0.026350	4.99	<.0001

The following statements fit a spatial Durbin model (SDM) by using Taylor approximation. You specify the keyword TAYLOR in the APPROXIMATION= option to request Taylor approximation. In particular, you request a 50th-order Taylor polynomial to be used for approximation by specifying the ORDER=50 option.

```
/*-- SDM : Taylor approximation with full W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW approximation=(Taylor ORDER=50);
  model y=x1-x3 / type=SAR;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 13.4.3](#). Compared with [Output 13.2.3](#), the SDM that is fit using Taylor approximation yields almost identical parameter estimates.

Output 13.4.3 Parameter Estimates of SDM with Taylor Approximation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932578	0.198882	9.72	<.0001
x1	1	0.548504	0.049806	11.01	<.0001
x2	1	0.686011	0.056266	12.19	<.0001
x3	1	-0.890161	0.053516	-16.63	<.0001
W_x1	1	0.172302	0.154018	1.12	0.2633
W_x2	1	0.023746	0.198557	0.12	0.9048
W_x3	1	-0.324808	0.228032	-1.42	0.1543
_rho	1	-0.639757	0.164651	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

With the compact representation, the following statements fit the SDM by using Taylor approximation:

```

/*-- SDM : Taylor approximation with compact W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW_Compact
    approximation=(Taylor ORDER=50);
    model y=x1-x3/ type=SAR;
    spatialeffects x1-x3;
    spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 13.4.4](#), which is identical to [Output 13.4.3](#).

Output 13.4.4 Parameter Estimates of SDM with Taylor Approximation and Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932578	0.198882	9.72	<.0001
x1	1	0.548504	0.049806	11.01	<.0001
x2	1	0.686011	0.056266	12.19	<.0001
x3	1	-0.890161	0.053516	-16.63	<.0001
W_x1	1	0.172302	0.154018	1.12	0.2633
W_x2	1	0.023746	0.198557	0.12	0.9048
W_x3	1	-0.324808	0.228032	-1.42	0.1543
_rho	1	-0.639757	0.164651	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

To use Chebyshev approximation for the preceding SDM, submit the following statements:

```

/*-- SDM : Chebyshev approximation with full W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW approximation=(ORDER=10);
    model y=x1-x3/ type=SAR;
    spatialeffects x1-x3;
    spatialid SID;

```

```
run;
```

The parameter estimates for this model are shown in [Output 13.4.5](#), which is similar to [Output 13.4.3](#).

Output 13.4.5 Parameter Estimates of SDM with Chebyshev Approximation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932574	0.198876	9.72	<.0001
x1	1	0.548505	0.049806	11.01	<.0001
x2	1	0.686012	0.056266	12.19	<.0001
x3	1	-0.890162	0.053516	-16.63	<.0001
W_x1	1	0.172300	0.154015	1.12	0.2633
W_x2	1	0.023744	0.198554	0.12	0.9048
W_x3	1	-0.324805	0.228027	-1.42	0.1543
_rho	1	-0.639755	0.164646	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

To use Chebyshev approximation for this model with compact representation, submit the following statements:

```
/*-- SDM : Chebyshev approximation with compact W --*/
proc cspatialreg data=mylib.SimData wmat=mylib.SimW_Compact approximation=(ORDER=10);
  model y=x1-x3/ type=SAR;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 13.4.6](#).

Output 13.4.6 Parameter Estimates of SDM with Chebyshev Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932574	0.198876	9.72	<.0001
x1	1	0.548505	0.049806	11.01	<.0001
x2	1	0.686012	0.056266	12.19	<.0001
x3	1	-0.890162	0.053516	-16.63	<.0001
W_x1	1	0.172300	0.154015	1.12	0.2633
W_x2	1	0.023744	0.198554	0.12	0.9048
W_x3	1	-0.324805	0.228027	-1.42	0.1543
_rho	1	-0.639755	0.164646	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

The following statements fit a spatial error model (SEM) by using Chebyshev approximation:

```

/*-- SEM : full W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW approximation=(ORDER=10);
  model y=x1-x3 / type=SEM;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 13.4.7](#). Note that the spatial weights matrix in the mylib.SimW data table is a full matrix. Compared with [Output 13.2.4](#), Chebyshev approximation yields similar parameter estimates.

Output 13.4.7 Parameter Estimates of SEM with Chebyshev Approximation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.166221	0.029666	39.31	<.0001
x1	1	0.488222	0.049184	9.93	<.0001
x2	1	0.635419	0.061432	10.34	<.0001
x3	1	-0.831401	0.054913	-15.14	<.0001
_lambda	1	-0.957363	0.117021	-8.18	<.0001
_sigma2	1	0.148034	0.031092	4.76	<.0001

Using the compact representation of the spatial weights matrix, you can submit the following statements to fit an SEM by using Chebyshev approximation:

```

/*-- SEM : compact form of W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW_Compact approximation=(ORDER=10);
  model y=x1-x3 / type=SEM;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 13.4.8](#), which is almost identical to [Output 13.4.7](#).

Output 13.4.8 Parameter Estimates of SEM with Chebyshev Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.166221	0.029666	39.31	<.0001
x1	1	0.488222	0.049184	9.93	<.0001
x2	1	0.635419	0.061432	10.34	<.0001
x3	1	-0.831401	0.054913	-15.14	<.0001
_lambda	1	-0.957363	0.117021	-8.18	<.0001
_sigma2	1	0.148034	0.031092	4.76	<.0001

The following statements fit a spatial Durbin error model (SDEM) by using Taylor approximation:

```

/*-- SDEM : Taylor approximation with full W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW approximation=(Taylor ORDER=50);
  model y=x1-x3/ type=SEM;
  spatialeffects x1-x3;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 13.4.9](#).

Output 13.4.9 Parameter Estimates of SDEM with Taylor Approximation

The CSPATIALREG Procedure

Parameter Estimates						
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t	
Intercept	1	1.193178	0.047129	25.32	<.0001	
x1	1	0.566497	0.056782	9.98	<.0001	
x2	1	0.722060	0.059717	12.09	<.0001	
x3	1	-0.909212	0.061286	-14.84	<.0001	
W_x1	1	-0.135284	0.128339	-1.05	0.2918	
W_x2	1	-0.418116	0.152899	-2.73	0.0062	
W_x3	1	0.290041	0.192397	1.51	0.1317	
_lambda	1	-0.723853	0.209844	-3.45	0.0006	
_sigma2	1	0.126723	0.026820	4.73	<.0001	

With the compact representation, the following statements fit the SDEM by using Taylor approximation:

```

/*-- SDEM : Taylor approximation with compact W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW_Compact
  approximation=(Taylor ORDER=50);
  model y=x1-x3/ type=SEM;
  spatialeffects x1-x3;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 13.4.10](#), which is identical to [Output 13.4.9](#).

Output 13.4.10 Parameter Estimates of SDEM with Taylor Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates						
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t	
Intercept	1	1.193178	0.047129	25.32	<.0001	
x1	1	0.566497	0.056782	9.98	<.0001	
x2	1	0.722060	0.059717	12.09	<.0001	
x3	1	-0.909212	0.061286	-14.84	<.0001	
W_x1	1	-0.135284	0.128339	-1.05	0.2918	
W_x2	1	-0.418116	0.152899	-2.73	0.0062	
W_x3	1	0.290041	0.192397	1.51	0.1317	
_lambda	1	-0.723853	0.209844	-3.45	0.0006	
_sigma2	1	0.126723	0.026820	4.73	<.0001	

To use Chebyshev approximation to fit the preceding SDEM, submit the following statements:

```
/*-- SDEM : Chebyshev approximation with full W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW approximation=(ORDER=10);
  model y=x1-x3/ type=SEM;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 13.4.11](#), which is similar to [Output 13.4.9](#).

Output 13.4.11 Parameter Estimates of SDEM with Chebyshev Approximation

The CSPATIALREG Procedure

Parameter Estimates						
Parameter	DF	Estimate	Standard Error	t Value	Pr > t	Approx
Intercept	1	1.193178	0.047129	25.32	<.0001	
x1	1	0.566500	0.056780	9.98	<.0001	
x2	1	0.722062	0.059716	12.09	<.0001	
x3	1	-0.909213	0.061285	-14.84	<.0001	
W_x1	1	-0.135290	0.128335	-1.05	0.2918	
W_x2	1	-0.418122	0.152895	-2.73	0.0062	
W_x3	1	0.290046	0.192393	1.51	0.1317	
_lambda	1	-0.723821	0.209774	-3.45	0.0006	
_sigma2	1	0.126724	0.026819	4.73	<.0001	

To use Chebyshev approximation for this model with compact representation, submit the following statements:

```
/*-- SDEM : Chebyshev approximation with compact W --*/
proc cspatialreg data=mylib.SimData Wmat=mylib.SimW_Compact approximation=(ORDER=10);
  model y=x1-x3/ type=SEM;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 13.4.12](#).

Output 13.4.12 Parameter Estimates of SDEM with Chebyshev Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.193178	0.047129	25.32	<.0001
x1	1	0.566500	0.056780	9.98	<.0001
x2	1	0.722062	0.059716	12.09	<.0001
x3	1	-0.909213	0.061285	-14.84	<.0001
W_x1	1	-0.135290	0.128335	-1.05	0.2918
W_x2	1	-0.418122	0.152895	-2.73	0.0062
W_x3	1	0.290046	0.192393	1.51	0.1317
_lambda	1	-0.723821	0.209774	-3.45	0.0006
_sigma2	1	0.126724	0.026819	4.73	<.0001

Example 13.5: Impact Estimation

Data Description and Objective

Examples in this section use the CrimeOH and CrimeWmat data sets that are described in “[Example 13.1: Columbus Crime Data](#)” on page 687 to demonstrate impact estimation in PROC CSPATIALREG.

Spatial Lag of X (SLX) Model

The following statements fit a spatial lag of X (SLX) model to the data by using Income and HValue as explanatory variables, and they estimate the average direct impacts, the average indirect impacts, and the average total impacts for these two explanatory variables.

```

/*-- SLX --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat impact(nmc=1000);
  model crime=income hvalue / type=LINEAR;
  spatialeffects income hvalue;
  spatialid sid;
run;

```

In this example, the TYPE=LINEAR option in the MODEL statement and SPATIALEFFECTS statement specify an SLX model. The IMPACT option requests that the average direct impacts, the average indirect impacts, and the average total impacts be computed for explanatory variables in the model. The impact estimates for this model are shown in [Output 13.5.2](#). According to [Output 13.5.2](#), Income has a direct impact of -1.11 and an indirect impact of -1.37 on Crime, which equal the coefficient estimate of Income and the spatial lag of Income, respectively, in [Output 13.5.1](#). Similarly, HValue has a direct impact of -0.29 and an indirect impact of 0.19 on Crime, which equal the coefficient estimate of HValue and the spatial lag of HValue, respectively, in [Output 13.5.1](#).

Output 13.5.1 Parameter Estimates of the SLX Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	75.028374	6.278808	11.95	<.0001
income	1	-1.108925	0.354227	-3.13	0.0017
hvalue	1	-0.289729	0.096056	-3.02	0.0026
W_income	1	-1.370974	0.531869	-2.58	0.0099
W_hvalue	1	0.191771	0.189837	1.01	0.3124
_sigma2	1	107.287543	21.675353	4.95	<.0001

Output 13.5.2 Impact Estimates of Regressors in the SLX Model

Impact Estimates Summary								
Parameter	Direct			Indirect			Total	
	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value	Mean	Standard Deviation
income	-1.108925	0.354227	-3.13	-1.370974	0.531869	-2.58	-2.479899	0.470529
hvalue	-0.289729	0.096056	-3.02	0.191771	0.189837	1.01	-0.097958	0.192176

Spatial Error Model (SEM)

The following statements fit an SEM model to the data by using Income and HValue as two explanatory variables, and they estimate impacts for these two variables:

```

/*-- SEM --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat impact(nmc=10000);
    model crime=income hvalue / type=SEM;
    spatialid sid;
run;

```

In this example, the TYPE=SEM option in the MODEL statement specifies an SEM model. The IMPACT option requests that the average direct impacts, the average indirect impacts, and the average total impacts be computed for explanatory variables in the model. The impact estimates for this model are shown in [Output 13.5.4](#). According to the results, both Income and HValue have direct impacts but not indirect impacts on the dependent variable Crime. Moreover, the direct impacts of Income and HValue equal their coefficient estimates in [Output 13.5.3](#).

Output 13.5.3 Parameter Estimates of the SEM Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	59.914436	5.881216	10.19	<.0001
income	1	-0.942892	0.370508	-2.54	0.0109
hvalue	1	-0.302218	0.090549	-3.34	0.0008
_lambda	1	0.560314	0.152809	3.67	0.0002
_sigma2	1	95.554137	20.025909	4.77	<.0001

Output 13.5.4 Impact Estimates of Regressors in the SEM Model

Impact Estimates Summary								
Parameter	Direct			Indirect			Total	
	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value	Mean	Standard Deviation
income	-0.942892	0.370508	-2.54	0	0	.	-0.942892	0.370508
hvalue	-0.302218	0.090549	-3.34	0	0	.	-0.302218	0.090549

Spatial Autoregressive (SAR) Model

The following statements fit a SAR model to the data by using the explanatory variables Income and HValue, and they request that the average direct impacts, the average indirect impacts, and the average total impacts be computed for the two explanatory variables:

```

/*-- SAR --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat
  impact(nmc=50000 order=100);
  model crime=income hvalue / type=SAR;
  spatialid sid;
run;

```

The impact estimates for this model are shown in [Output 13.5.5](#). On one hand, you see that Income exerts a negative direct and indirect impact on the number of crimes. This suggests that you would see a decrease in the number of crimes in neighborhoods that have higher incomes. Moreover, increased Income in nearby neighborhoods has a negative impact on the number of crimes. On the other hand, HValue exerts a negative direct impact on the number of crimes. This suggests that you would see a decrease in the number of crimes in neighborhoods that have higher housing values.

Output 13.5.5 Impact Estimates of Regressors in the SAR Model
The CSPATIALREG Procedure

Impact Estimates Summary								
Parameter	Direct			Indirect			Total	
	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value	Mean	Standard Deviation
income	-1.091669	0.332260	-3.29	-0.751319	0.372889	-2.01	-1.842988	0.565408
hvalue	-0.282143	0.094550	-2.98	-0.209330	0.141523	-1.48	-0.491473	0.211251

Spatial Durbin Model (SDM)

The following statements fit an SDM model to the data by using the explanatory variables Income and HValue, and they estimate impacts for these two variables:

```
/*-- SDM --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat
  impact(nmc=50000 order=100);
  model crime=income hvalue / type=SAR;
  spatialeffects income hvalue;
  spatialid sid;
run;
```

In this example, the TYPE=SAR option in the MODEL statement specifies a SAR model. The SPATIALEFFECTS statement requests that the spatial lag of explanatory variables Income and HValue be included in the model. The IMPACT option requests that the average direct impacts, the average indirect impacts, and the average total impacts be computed for explanatory variables in the model. To estimate impacts, the NMC= option specifies 50,000 iterations of Monte Carlo simulation and the ORDER= option specifies 100 as the order of Neumann series. The impact estimates for this model are shown in [Output 13.5.6](#). On one hand, you see that the indirect impact of Income is not significant at the 5% significance level (because the *t* value of -1.58 is greater than -1.96). However, the direct impact of Income is negative and significant, suggesting that you would see a decrease in the number of crimes in neighborhoods that have higher incomes. Similarly, HValue exerts a negative direct impact on the number of crimes. This suggests that you would see a decrease in the number of crimes in neighborhoods that have higher housing values.

Output 13.5.6 Impact Estimates of Regressors in the SDM Model
The CSPATIALREG Procedure

Parameter	Impact Estimates Summary								
	Direct			Indirect			Total		
	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value
income	-1.021846	0.329139	-3.10	-1.493714	0.945050	-1.58	-2.515560	1.008252	-2.49
hvalue	-0.278730	0.093730	-2.97	0.201998	0.368789	0.55	-0.076733	0.406750	-0.19

Example 13.6: Comparison of Multiple Models

Data Description and Objective

Example in this section uses the CrimeOH and CrimeWmat data sets that are described in “[Example 13.1: Columbus Crime Data](#)” on page 687 to demonstrate fitting and comparison of multiple models in PROC CSPATIALREG.

Comparison of Multiple Models

The following statements fit multiple models to the CrimeOH and CrimeWmat data sets, which are described in “[Example 13.1: Columbus Crime Data](#)” on page 687, using the regressors Income and HValue:

```
/*-- TYPE=AUTO --*/
proc cspatialreg data=mylib.CrimeOH Wmat=mylib.CrimeWmat;
  model crime=income hvalue / type=AUTO;
  spatialid sid;
run;
```

In this example, the TYPE=AUTO option is specified in the MODEL statement to fit seven models—SAC, SARMA, SAR, SEM, SMA, CAR, and LINEAR. However, the CAR model is not fitted because the input spatial weights matrix **W** is not symmetric. The parameter estimates for the six remaining models—SAC, SARMA, SAR, SEM, SMA, and LINEAR—are shown in [Output 13.6.1](#). The comparison of these six models is provided in [Output 13.6.2](#). If AIC is used for model comparison, [Output 13.6.2](#) suggests that the SAR model is the best model among the six candidate models, because its AIC value is the smallest.

Output 13.6.1 Parameter Estimates of Multiple Models
The CSPATIALREG Procedure

Parameter Estimates						
Model Type	Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
SAC	Intercept	1	47.784032	9.279544	5.15	<.0001
	income	1	-1.025892	0.334029	-3.07	0.0021
	hvalue	1	-0.281653	0.093369	-3.02	0.0026
	_rho	1	0.368061	0.181191	2.03	0.0422
	_lambda	1	0.166684	0.298147	0.56	0.5761
	_sigma2	1	95.603530	19.475445	4.91	<.0001
SARMA	Intercept	1	48.802582	9.537830	5.12	<.0001
	income	1	-1.018561	0.334811	-3.04	0.0023
	hvalue	1	-0.286879	0.092709	-3.09	0.0020
	_rho	1	0.341517	0.200492	1.70	0.0885
	_lambda	1	-0.258955	0.416859	-0.62	0.5345
	_sigma2	1	96.762195	20.618634	4.69	<.0001
SAR	Intercept	1	45.079467	7.871241	5.73	<.0001
	income	1	-1.031616	0.328416	-3.14	0.0017
	hvalue	1	-0.265927	0.088222	-3.01	0.0026
	_rho	1	0.431018	0.123619	3.49	0.0005
	_sigma2	1	95.494608	19.507896	4.90	<.0001
SEM	Intercept	1	59.914436	5.881216	10.19	<.0001
	income	1	-0.942892	0.370508	-2.54	0.0109
	hvalue	1	-0.302218	0.090549	-3.34	0.0008
	_lambda	1	0.560314	0.152809	3.67	0.0002
	_sigma2	1	95.554137	20.025909	4.77	<.0001
SMA	Intercept	1	59.375752	5.872720	10.11	<.0001
	income	1	-0.928794	0.361190	-2.57	0.0101
	hvalue	1	-0.288025	0.086456	-3.33	0.0009
	_lambda	1	-0.788372	0.273593	-2.88	0.0040
	_sigma2	1	115.685990	25.337065	4.57	<.0001
LINEAR	Intercept	1	68.618863	4.588210	14.96	<.0001
	income	1	-1.597304	0.323739	-4.93	<.0001
	hvalue	1	-0.273931	0.099989	-2.74	0.0062
	_sigma2	1	122.751696	24.799493	4.95	<.0001

Output 13.6.2 Comparison of Multiple Models

Model Comparison							
Model Type	Status	Reason	Number of Iterations	Max Abs Gradient	Log Likelihood	AIC	SBC
SAC	0	Convergence criterion (ABSGCONV=0.00001) satisfied.	36	0.0000	-182.4	376.8950	388.2459
SARMA	0	Convergence criterion (FCONV=1E-8) satisfied.	16	0.0445	-182.4	376.8184	388.1693
SAR	0	Convergence criterion (ABSGCONV=0.00001) satisfied.	16	0.0000	-182.6	375.1147*	384.5738*
SEM	0	Convergence criterion (FCONV=1E-8) satisfied.	35	0.0706	-183.6	377.1833	386.6424
SMA	0	Convergence criterion (FCONV=1E-8) satisfied.	16	0.0255	-183.5	376.9300	386.3891
LINEAR	0	Convergence criterion (ABSGCONV=0.00001) satisfied.	16	0.0000	-187.4	382.7542	390.3215
Asterisk (*) denotes the best model in the column.							

References

- Anselin, L. (1988). *Spatial Econometrics: Methods and Models*. Amsterdam: Springer.
- Anselin, L. (2001). "Spatial Econometrics." In *A Companion to Theoretical Econometrics*, edited by B. H. Baltagi, 310–330. Oxford: Wiley-Blackwell.
- Elhorst, J. P. (2013). *Spatial Econometrics: From Cross-Sectional Data to Spatial Panels*. Berlin: Springer.
- LaMotte, L. R. (1994). "A Note on the Role of Independence in t Statistics Constructed from Linear Statistics in Regression Models." *American Statistician* 48:238–240.
- LeSage, J., and Pace, R. K. (2009). *Introduction to Spatial Econometrics*. Boca Raton, FL: CRC Press.

Chapter 14

CSSM Procedure

Contents

Overview: CSSM Procedure	719
PROC CSSM Compared with PROC SSM	720
Background	721
Using CAS Sessions and CAS Engine Librefs	721
Getting Started: CSSM Procedure	722
Syntax: CSSM Procedure	731
Functional Summary	732
PROC CSSM Statement	734
BY Statement	738
COMPONENT Statement	738
DEPLAG Statement	739
DISPLAY Statement	741
DISPLAYOUT Statement	742
EVAL Statement	743
ID Statement	743
IRREGULAR Statement	744
MODEL Statement	744
OBSINDEX Statement	745
OUTPUT Statement	746
PARMS Statement	747
Programming Statements	748
SCORE Statement	748
SIMSMOOTH Statement	749
STATE Statement	750
TREND Statement	755
Details	760
State Space Model and Notation	760
Types of Sequence Data	762
Overview of Model Specification Syntax	763
Building a Complex Model Specification	763
Model Specification Steps	764
Sparse Transition Matrix Specification	765
Regression Variable Specification in Multivariate Models	766
Filtering, Smoothing, Likelihood, and Structural Break Detection	767
Filtering Pass	768
Likelihood Computation and Model-Fitting Phase	769

Forecasting Phase	772
Smoothing Phase	772
Delete-One Cross Validation and Structural Breaks	773
Estimation of User-Specified Linear Combination of State Elements	775
Contrasting PROC CSSM with Other SAS Procedures	775
Predefined Trend Models	776
Trend Models for Regular Data	776
Trend Models for Irregular Data	778
Predefined Structural Models	779
Multivariate White Noise	781
Multivariate Random Walk Trend	781
Multivariate Local Linear Trend	782
Multivariate Cycle	782
Multivariate Season	783
Multivariate ARMA	784
Continuous-Time Cycle	785
Models with Dependent Lags	785
Simulation Smoothing	787
Output Tables Associated with Simulation Smoothing	788
OUTCOMP= Table	788
OUTPRED= Table	789
OUTSTATE= Table	789
Temporal Aggregation and Temporal Distribution	790
Temporal Distribution	790
Temporal Aggregation	792
Covariance Parameterization	793
Missing Values	793
Scoring with State Space Models	794
Initial Score-Store Creation	794
Incorporating New Information	795
What-If Analysis and Process Control	795
Monitoring of Streaming Data	796
Computational Issues	797
A Well-Behaved Model	797
Convergence Problems	798
Computer Resource Requirements	798
Default Displayed Output	798
Displayed Output	799
ODS Graphics	801
OUT= Data Set	802
Examples: CSSM Procedure	803
Example 14.1: Bivariate Basic Structural Model	803
Example 14.2: Random-Effects and Autoregressive Models for Panel Data	809
Example 14.3: Backcasting, Forecasting, and Interpolation	813

Example 14.4: Smoothing of Repeated Measures for Longitudinal Data	814
Example 14.5: A User-Defined Trend Model	823
Example 14.6: Model with Multiple ARIMA Components	827
Example 14.7: A Dynamic Factor Model for the Yield Curve	831
Example 14.8: Diagnostic Plots and Structural Break Analysis	841
Example 14.9: Variable Bandwidth Smoothing for Longitudinal Data	848
Example 14.10: A Transfer Function Model for the Gas Furnace Data	853
Example 14.11: Dynamic Panel Model for the Cigar Data	859
Example 14.12: Long-Term Temperature Trends Based on a Multivariate Model . . .	864
Example 14.13: Bivariate Model for Sales of Mink and Muskrat Furs	871
Example 14.14: A Factor Model for Now-Casting the US Economy	876
Example 14.15: Longitudinal Data Analysis of Lung Function Data	881
Example 14.16: Estimating Monthly GDP by Temporal Distribution	884
Example 14.17: Temporal Aggregation-Based Triannual Nile River Level	889
Example 14.18: Invariance of the Marginal Likelihood under Linear Rescaling of the Diffuse Effects	891
Example 14.19: Scenario Analysis Using Scoring	894
Example 14.20: Monitoring Streaming Data Using Scoring	897
Example 14.21: Simulation-Smoothing-Based Sampling Distribution of the Sum of Forecasts	901
References	904

Overview: CSSM Procedure

State space models (SSMs) are used for analyzing continuous response variables that are recorded sequentially according to a numeric indexing variable. In many cases, the indexing variable is time and the observations are collected at regular time intervals—for example, hourly, weekly, or monthly. In such cases, the resulting data are called time series data. In other cases, the indexing variable might not be time or the observations might not be equally spaced according to the indexing variable. These more general types of sequential data are called longitudinal data. Because of their sequential nature, these types of data exhibit some characteristic features. For example, chronologically closer measurements tend to be highly correlated while measurements farther apart are essentially uncorrelated. Data can be trending in a particular direction and can have seasonal or other periodic patterns. SSMs are specially designed to model such sequential data. They apply to both univariate and multivariate response situations and can easily incorporate predictor (independent variable) information when it is available.

The CSSM procedure performs state space modeling of univariate and multivariate time series and longitudinal data. You can do the following with the CSSM procedure:

- analyze quite general linear state space models
- use an expressive language to specify an SSM. An SSM specification consists of specifying a variety of matrices—for example, the state transition matrix and the covariance matrices of the state and observation disturbances. The CSSM procedure provides language similar to a DATA step for specifying the

elements of these matrices. The matrix elements can be user-defined functions of data variables and unknown parameters.

- easily specify several commonly needed univariate and multivariate SSMs by using only a few keywords. These SSMs include the principal univariate and multivariate structural models for regularly spaced data and a variety of trend and cycle models for the longitudinal data.
- estimate unknown model parameters by (restricted) maximum likelihood. The likelihood function is computed by using the (diffuse) Kalman filter algorithm.
- print, or output to a data set, the series forecasts, residuals, and the full-sample estimates of any linear combination of the underlying state variables. These estimates are obtained by using the (diffuse) Kalman filter and smoother algorithm.
- generate residual diagnostic plots and plots useful for detecting structural breaks
- create a score store, a binary large object (BLOB) table, that can be subsequently used for efficient, model-based scenario analysis and for scoring new observations
- using the simulation smoothing algorithm, simulate the latent states conditional on the observed data. This helps generate the sampling distribution of arbitrarily complex functions of the latent states.

PROC CSSM Compared with PROC SSM

PROC CSSM is the CAS version of PROC SSM. This means that it requires SAS Cloud Analytic Services (CAS) in order to run. Except for this difference, PROC CSSM and PROC SSM are identical in their modeling capabilities. Because it works in the cloud environment, PROC CSSM enables you to efficiently handle large numbers of BY groups by exploiting all the available cores and concurrent threads to process them in parallel. The scoring features that are provided by the [SCORE](#) statement and the simulation smoothing features that are provided by the [SIMSMOOTH](#) statement are new in PROC CSSM version 2024.11 and are not available in PROC SSM. These two procedures share virtually identical syntax and should produce essentially identical output. However, there are some important differences between them that arise mainly because CAS tables do not use observation order:

- The ID statement is required in PROC CSSM; it is optional in PROC SSM.
- In cases where multiple observations can have the same ID value, such as in panel data settings, you must use the new [OBSINDEX](#) statement to ensure the reproducibility of the computations. The ID variable together with the OBSINDEX variable uniquely indexes each observation within a BY group.

Background

State space models are widely used in a variety of fields, such as engineering, statistics, econometrics, and agriculture. There are numerous references that deal with state space modeling, particularly with the state space modeling of time series data. State space modeling of longitudinal data has received a little less attention. The primary reference for the modeling techniques implemented in the CSSM procedure is Harvey (1989), which contains treatments of both the time series and longitudinal data. Other useful books about this subject are Pelagatti (2015); Durbin and Koopman (2012); Jones (1993), and Anderson and Moore (1979). In addition, informative articles about state space modeling of longitudinal data include Wecker and Ansley (1983); Kohn and Ansley (1991); De Jong and Mazzi (2001); Eubank, Huang, and Wang (2003), and Selukar (2015). For the implementation details of the diffuse Kalman filter and smoother (the main computational tool used by the CSSM procedure), the main references are a series of articles (De Jong 1989, 1991; De Jong and Chu-Chun-Lin 2003) and the references therein.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section [“Introduction to Shared Concepts”](#) on page 54 in Chapter 4, [“Shared Concepts.”](#)

Getting Started: CSSM Procedure

This example illustrates how you can use the CSSM procedure to analyze a panel of time series. The following data set, *Cigar*, contains information about yearly per capita cigarette sales for 46 geographic regions in the United States over the period 1963–1992. The variables *lsales*, *lprice*, *lndi*, and *lpimin* denote the per capita cigarette sales, price per pack of cigarettes, per capita disposable income, and minimum price in adjoining regions per pack of cigarettes, respectively (all in the natural log scale). The variable *year* contains the observation year, and the variable *region* contains an integer between 1 to 46 that serves as the unique identifier for the region. For additional data description see Baltagi and Levin (1992); Baltagi (1995).

```
data cigar;
  input year region lsales lprice lndi lpimin;
  label lsales = 'Log cigarette sales in packs per capita';
  label lprice = 'Log price per pack of cigarettes';
  label lndi = 'Log per capita disposable income';
  label lpimin = 'Log minimum price in adjoining regions
                 per pack of cigarettes';
  year = intnx( 'year', '1jan63'd, year-63 );
  format year year.;
datalines;
63 1 4.54223 3.35341 7.3514 3.26194
63 2 4.82831 3.17388 7.5729 3.21487
63 3 4.63860 3.29584 7.3000 3.25037
63 4 4.95583 3.23080 7.9288 3.17388
63 5 5.05114 3.28840 7.9772 3.26576

... more lines ...
```

The following statements load the *Cigar* data into the table *mylib.Cigar*:

```
/* Load data into mylib */
data mylib.Cigar;
  set Cigar;
run;
```

The goal of the analysis is to study the impact of the regressors on the smoking behavior and to understand the changes in the smoking patterns in different regions over the years. Consider the following model for *lsales*:

$$\text{lsales}_{i,t} = \mu_{i,t} + \text{lprice} \beta_1 + \text{lndi} \beta_2 + \text{lpimin} \beta_3 + \epsilon_{i,t}$$

This model represents *lsales* in a region *i* and in a year *t* as a sum of region-specific trend components $\mu_{i,t}$, the regression effects due to *lprice*, *lndi*, and *lpimin*, and the observation noise $\epsilon_{i,t}$. Different variations of this model are obtained by considering different models for the trend component $\mu_{i,t}$. Proper modeling of the trend component is important because it captures differences between the regions that result from unrecorded factors such as demographic changes over time, results of anti-smoking campaigns, and so on. The following statements specify and fit one such model:

```

proc cssm data=mylib.Cigar plots=residual;
  id year interval=year;
  obsindex region;
  array RegionArray{46} region1-region46;
  do i=1 to 46;
    RegionArray[i] = (region=i);
  end;
  trend IrwTrend(11) cross(matchparm)=(RegionArray) levelvar=0;
  irregular wn;
  model lsales = lprice lndi lpimin IrwTrend wn;
  eval TrendPlusReg = IrwTrend + lprice + lndi + lpimin;
  output out=mylib.forCigar pdv press;
run;

```

The PROC CSSM statement specifies the input table, `mylib.Cigar`, which contains analysis variables such as the response variable, `lsales`, and the predictor variables, `lprice`, `lndi`, and `lpimin`. The `PLOTS=RESIDUAL` option in the PROC CSSM statement produces residual diagnostic plots. The `ID` statement specifies a numeric index variable (often a SAS date or datetime variable), which is `year` in this case. The `INTERVAL=YEAR` option in the `ID` statement indicates that the measurements are collected yearly. In this example, the `ID` variable (`year`) by itself is not enough to uniquely index each observation in the input table. In such cases, in order to ensure the repeatability of the analysis, you must supply a variable by using the `OBSINDEX` statement (`obsindex region;`) that, together with the `ID` variable, uniquely indexes each observation in the input table.

The next few statements define a 46-dimensional array of dummy variables, `RegionArray`, such that `RegionArray[i]` is 1 if region is i and is 0 otherwise. The next three statements, `TREND`, `IRREGULAR`, and `MODEL`, constitute the model specification part of the program:

- `trend IrwTrend(11) cross(matchparm)=(RegionArray) levelvar=0;` defines a trend, named `IrwTrend`, of local linear type (which is signified by the keyword `11` used within the parenthesis after the name). A local linear trend—a trend with time-varying level and time-varying slope—depends on two parameters: the disturbance variance of the level equation and the disturbance variance of the slope equation (see the section “[Local Linear Trend](#)” on page 776 for more information). The `LEVELVAR=0` specification fixes the disturbance variance of the level equation to 0, which results in a trend model called an *integrated random walk* (IRW). An IRW model tends to produce a smoother trend than a general local linear trend. In the limiting case, if the disturbance variance of the slope equation is also 0, the IRW trend reduces to a straight line (with a fixed intercept and slope). In addition, because of the use of the 46-dimensional array, `RegionArray`, in the `CROSS=` option (`cross(matchparm)=(RegionArray)`), this trend specification amounts to fitting a separate IRW trend for each region. This is because, as a result of the `CROSS=` option, `IrwTrend` is treated as a linear combination of 46 (the number of variables in `RegionArray`) stochastically independent, integrated random walks,

$$\text{IrwTrend}_t = \sum_{i=1}^{46} \text{RegionArray}[i] \mu_{i,t}$$

where each $\mu_{i,t}$ is an integrated random walk. Note that since `RegionArray[i]` is a binary variable, `IrwTrend` equals $\mu_{i,t}$ when region is i . Lastly, the use of `MATCHPARM` option specifies that the different IRW trends $\mu_{i,t}$ use the same disturbance variance parameter for their slope equation. This is

done mainly for parsimony. Based on the model diagnostics shown later, this appears to be a reasonable model simplification.

- **irregular wn**; defines the observation noise $\epsilon_{i,t}$, named **wn**, as a sequence of independent, identically distributed, zero-mean, Gaussian variables—a white noise sequence.
- **model lsales = lprice lndi lpimin lrwTrend wn**; defines the model for **lsales** as a sum of regression effects that involve **lprice**, **lndi**, and **lpimin**, a trend term, **lrwTrend**, and the observation noise **wn**.

The last two statements, **EVAL** and **OUTPUT**, control certain aspects of the procedure output. The following **EVAL** statement defines a linear combination, named **TrendPlusReg**, of selected terms in the **MODEL** statement:

```
eval TrendPlusReg = lrwTrend + lprice + lndi + lpimin;
```

This **EVAL** statement causes the CSSM procedure to produce an estimate of **TrendPlusReg** (and its standard error), which can then be printed or output to a data set. **TrendPlusReg** contains all the terms in the model except for the observation noise and thus can be regarded as the *explanatory* part of the model. In the **OUTPUT** statement, you can specify an output data set that stores all the component estimates that are produced by the procedure. The following **OUTPUT** statement specifies **mylib.forCigar** as the output data set:

```
output out=mylib.forCigar pdv press;
```

The **PDV** option causes variables such as **region1**–**region46**, which are defined by the **DATA** step statements within the CSSM procedure, also to be included in the output data set. The **PRESS** option causes the printing of fit measures that are based on the delete-one cross validation errors (see the section “[Delete-One Cross Validation and Structural Breaks](#)” on page 773 for more information).

All the models that are specified in the CSSM procedure possess a state space representation. For more information, see the section “[State Space Model and Notation](#)” on page 760. The CSSM procedure output begins with a table (not shown here) of the input data set that provides the name and other information. Next, the “Model Summary” table, shown in [Figure 14.1](#), provides basic model information, such as the following:

- the dimension of the underlying state equation, 92 (because each of the 46 IRW trends $\mu_{i,t}$ contributes two elements to the state)
- the diffuse dimension of the model, 95 (which is equal to the three regressors plus the 92 diffuse initial states of $\mu_{i,t}$)
- the number of model parameters, 2 (which is the common disturbance variance of the slope equation in **lrwTrend** and the variance of the noise term **wn**)

This information is very useful in determining the computational complexity of the model (the larger state size, 92, explains the relatively long computing time—as much as two minutes on some desktops—for this example).

Figure 14.1 Summary of the Underlying State Space Model

Model Summary	
Number of Model Equations	1
State Dimension	92
Dimension of the Diffuse Initial Condition	95
Number of Parameters	2

The index variable information is shown in Figure 14.2. Among other things, it categorizes the data to be of the type *Regular with Replication*, which implies that the data are regularly spaced with respect to the ID variable and at least some observations have the same ID value. This is clearly true in this example: the data are yearly without any gaps, and there are 46 observations in each year—one per region.

Figure 14.2 Index Variable Information

ID Variable Information					
Maximum Distinct					
Name	Start	End	Delta	Values	Type
year	1963	1992	1.0000	30	Regular with Replication

Figure 14.3 provides simple summary information about the response variable. It shows that *Isales* has no missing values and no induced missing values because the predictors in the model, *lprice*, *Indi*, and *lpimin*, do not have any missing values either.

Figure 14.3 Response Variable Summary

Response Variable Information							
Number of Observations							
		Induced		Standard			
Name	Total	Missing	Missing	Minimum	Maximum	Mean	Deviation
Isales	1380	0	0	3.978	5.697	4.793	0.2246

The regression coefficients of *lprice*, *Indi*, and *lpimin* are shown in Figure 14.4. As expected, the coefficient of *lprice* is negative and the coefficients of *Indi* and *lpimin* are positive, all being statistically significant. This is consistent with the expectation that the cigarette sales are adversely affected by the price and are positively correlated with the disposable income. The estimated effect of *lpimin*, called the bootlegging effect by Baltagi and Levin (1992), is statistically significant but smaller than the effects of *lprice* and *Indi*.

Figure 14.4 Estimated Regression Coefficients

Regression Parameter Estimates					
		Standard			
Response	Regressor	Estimate	Error	t Value	Pr > t
Isales	lprice	-0.3480	0.0232	-15.01	<.0001
Isales	Indi	0.1425	0.0344	4.15	<.0001
Isales	lpimin	0.0619	0.0269	2.30	0.0214

Figure 14.5 Estimated Model Parameters

Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
lrwTrend	LL Trend	Slope Variance	0.0002	0.0000	7.72
wn	Irregular	Variance	0.0006	0.0000	17.29

Figure 14.5 shows the estimates of the disturbance variance of the slope equation in lrwTrend and the variance of the noise term wn.

Figure 14.6 shows a panel of residual normality diagnostic plots. These plots show that the residuals are symmetrically distributed but contain slightly larger than expected number of extreme residuals. Figure 14.7 shows the plot of residuals versus time. There the residuals do not exhibit any obvious pattern; however, the plot does show that more extreme residuals appear before 1970 and after 1989. On the whole, however, these plots do not exhibit serious violations of model assumptions.

Figure 14.6 Residual Normality Check

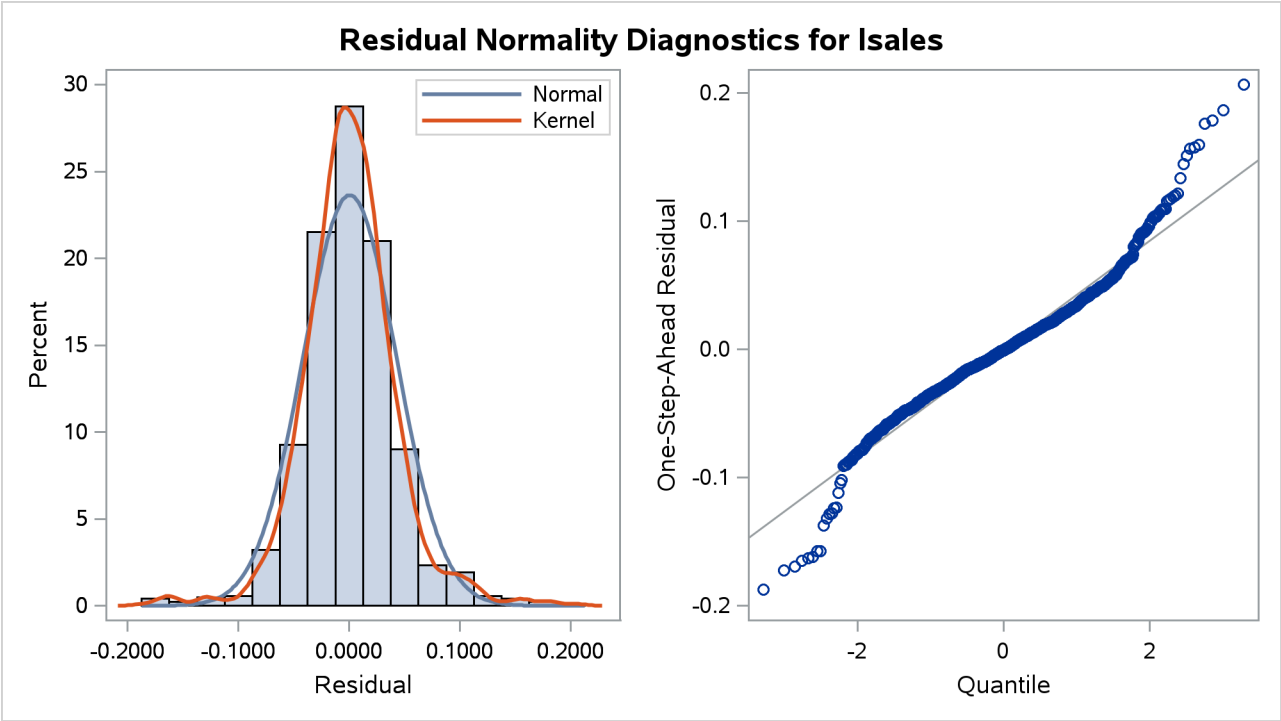


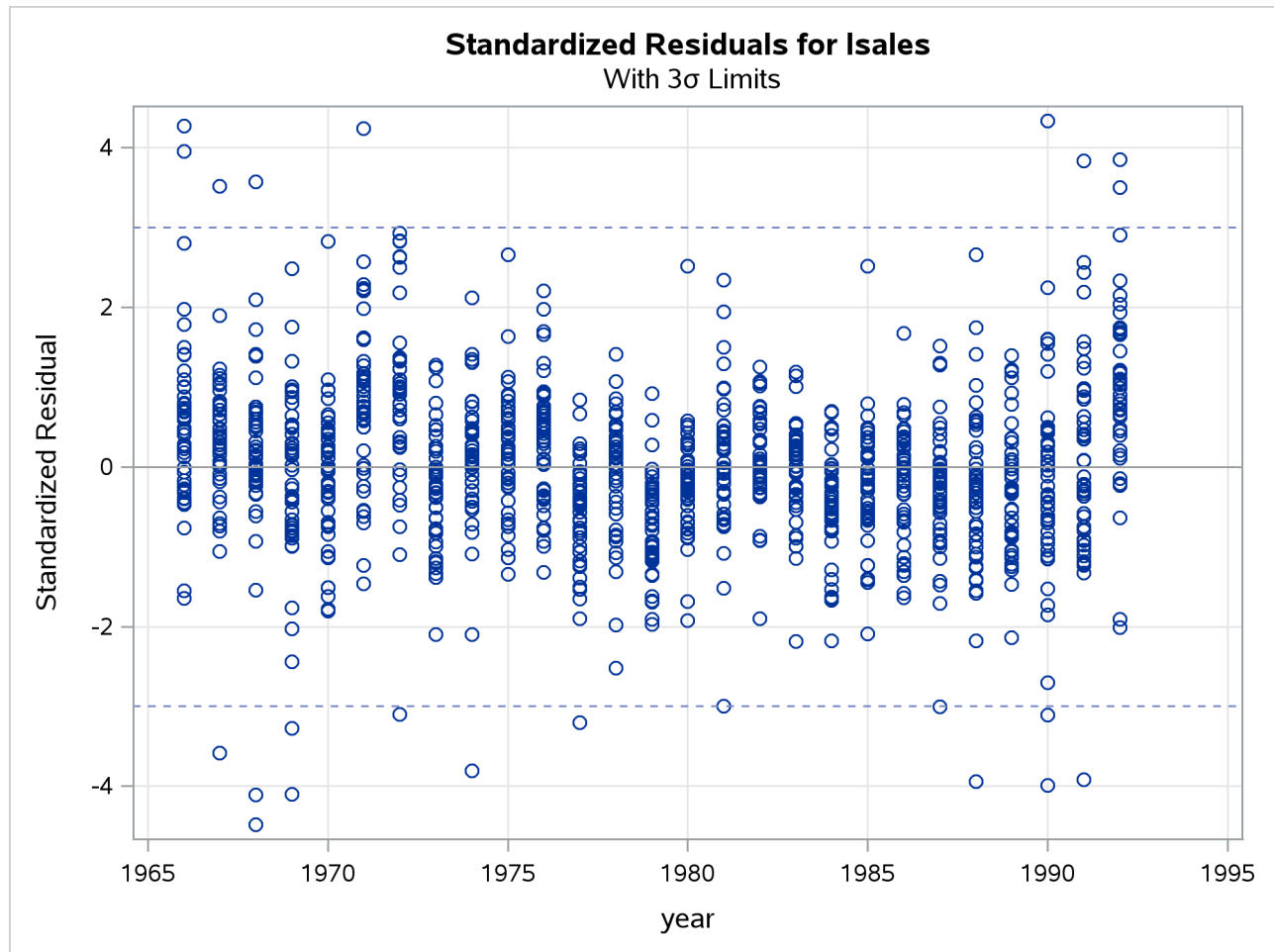
Figure 14.7 Standardized Residuals Plotted against Time

Figure 14.8 shows the details of the likelihood computations such as the number of nonmissing response values used and the likelihood of the fitted model. For more information, see the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769. Figure 14.8 shows the likelihood-based information criteria in lower-is-better format, which are useful for model comparison.

Figure 14.8 Likelihood Computation Details

Likelihood Computation Summary	
Nonmissing Response Values Used	1380
Estimated Parameters	2
Initialized Diffuse State Elements	95
Normalized Residual Sum of Squares	1285
Diffuse Log Likelihood	2246
Profile Log Likelihood	2170

Figure 14.9 Information Criteria

Information Criteria		
Statistic	Diffuse Likelihood	Profile Likelihood
AIC (lower is better)	-4488	-4145
BIC (lower is better)	-4478	-3638
AICC (lower is better)	-4488	-4130
HQIC (lower is better)	-4484	-3955
CAIC (lower is better)	-4476	-3541

In addition to the regression estimates, it is useful to analyze the estimates of different model components such as the trend component `lrwTrend` and the linear combination `TrendPlusReg`. These estimates can be printed by using the `PRINT=` option provided in the `TREND` and `EVAL` statements, or they can be output to a data set (as it is done in this illustration). This latter option is particularly useful for graphical exploration of these components by standard graphical procedures such as `SGPLOT` and `SGPANEL` procedures. The following statements produce a panel of plots that shows how well the proposed model fits the observed cigarette sales in the first three regions, which correspond to Alabama, Arizona, and Arkansas. The output data set, `mylib.forCigar`, contains all the needed information: `Smoothed_TrendPlusReg` contains the smoothed (full-sample) estimate of `TrendPlusReg`, and `Smoothed_Lower_TrendPlusReg` and `Smoothed_Upper_TrendPlusReg` contain its 95% lower and upper confidence limits. In addition, for easy readability, a user-defined format (`RegionFormat`), which is created by using the `FORMAT` procedure (not shown), is used to associate the region names to region values.

```
proc sgpanel data=mylib.forCigar noautolegend;
  where region <= 3;
  format region RegionFormat.;
  title 'Region-Specific Sales Patterns with 95% Confidence Band';
  panelby region / columns=3;
  band x=year lower=Smoothed_Lower_TrendPlusReg
  upper=Smoothed_Upper_TrendPlusReg;
  scatter x=year y=lsales;
  series x=year y= Smoothed_TrendPlusReg;
run;
```

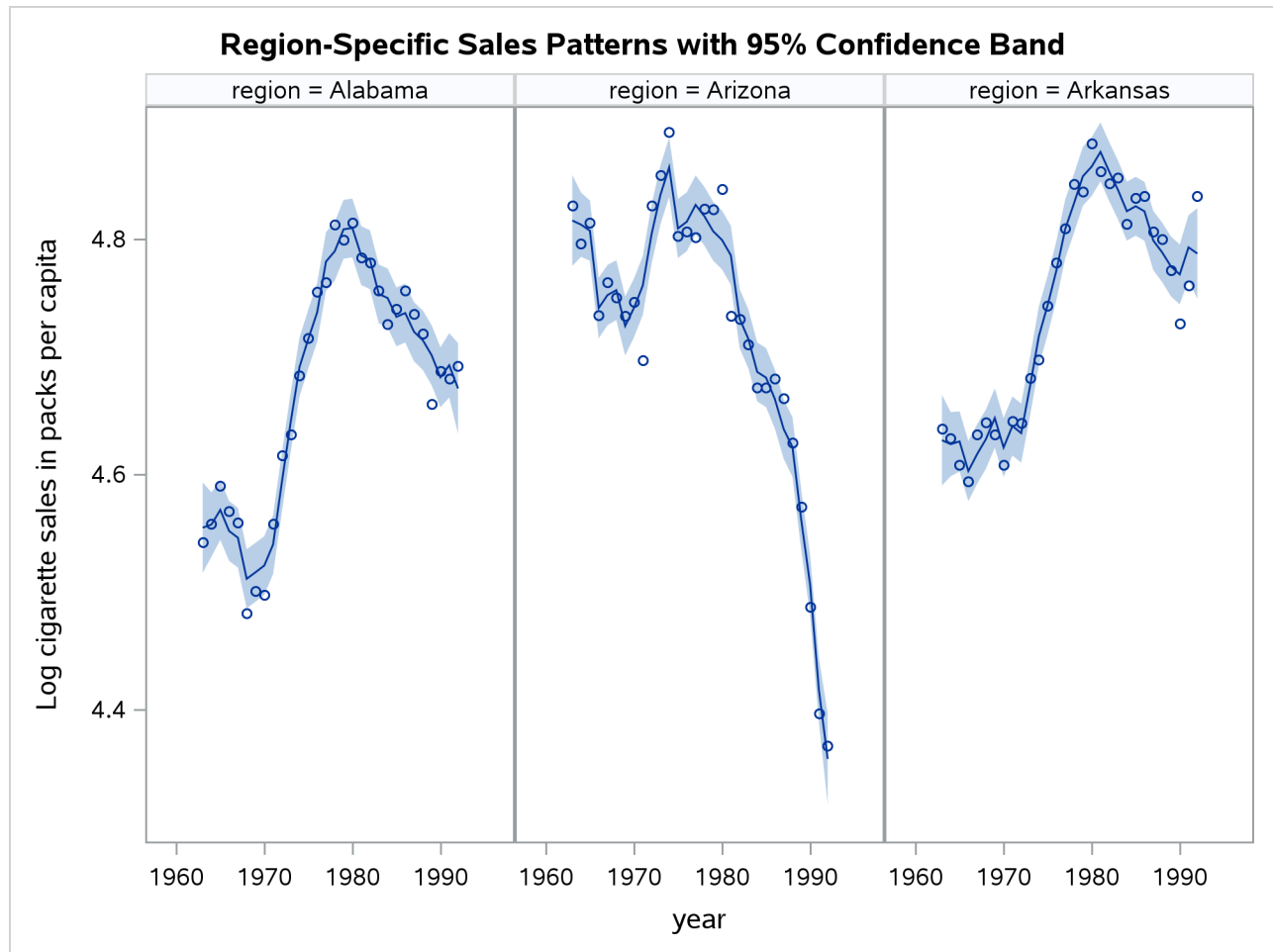
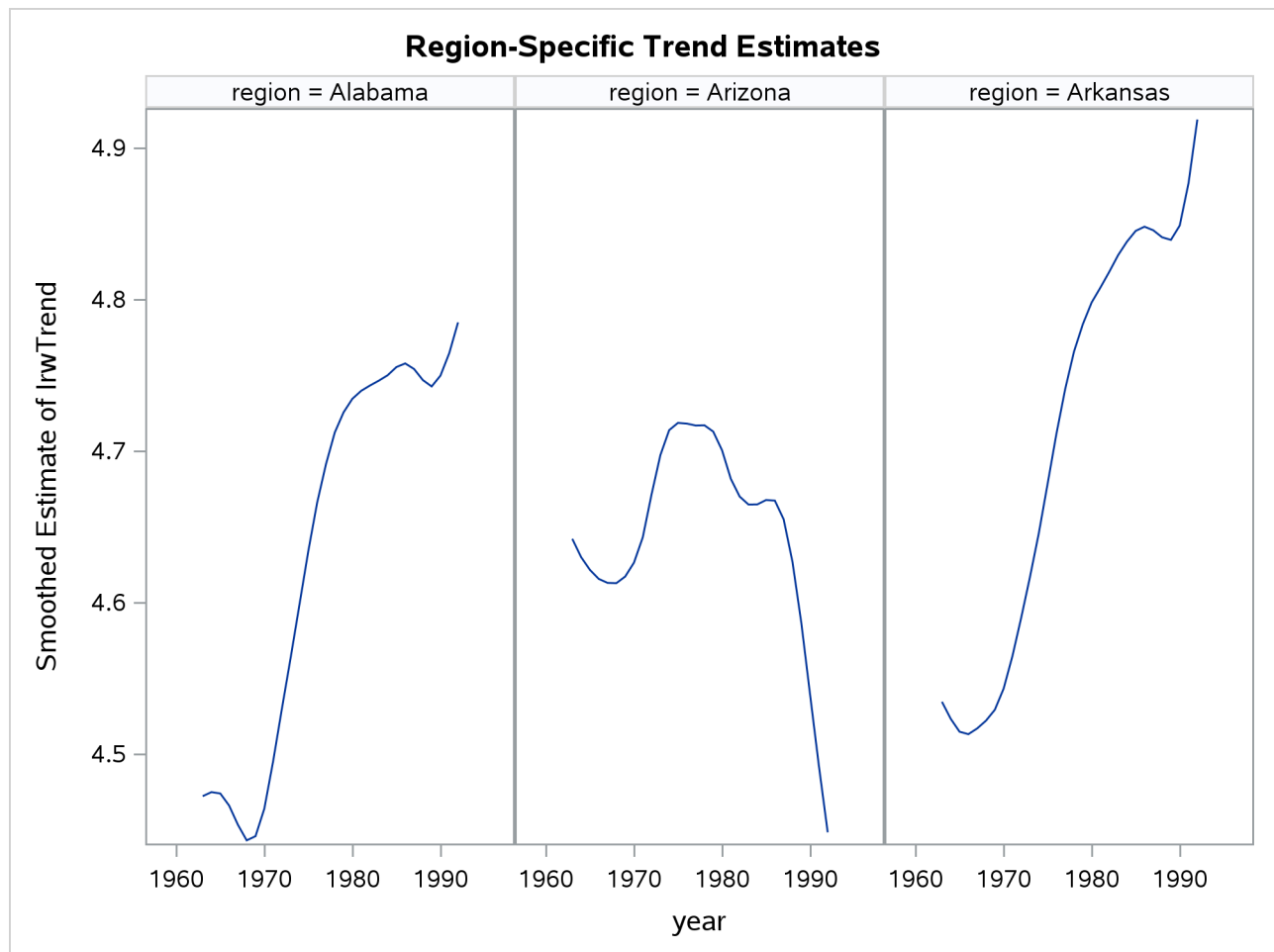
Figure 14.10 Cigarette Sales Patterns for the First Three Regions

Figure 14.10 seems to indicate that the model fits the data reasonably well. It also shows that Arizona differs markedly from Alabama and Arkansas in its cigarette sales pattern over the years. The following statements produce a similar panel of plots that show the estimate of trend without the regression effects:

```
proc sgpanel data=mylib.forCigar noautolegend;
  where region <= 3;
  format region RegionFormat.;
  title 'Region-Specific Trend Estimates';
  panelby region / columns=3;
  series x=year y= smoothed_IrwTrend;
run;
```

Figure 14.11 Estimate of $lrwTrend$ for the First Three Regions

The trend patterns, shown in Figure 14.11, seem to suggest that after accounting for the regression effects, per capita cigarette sales were on the rise in Alabama and Arkansas while they were declining in Arizona.

Syntax: CSSM Procedure

The following statements are available in the CSSM procedure:

```

PROC CSSM < options > ;
  BY variables ;
  COMPONENT name = (variables) * state < / options > ;
  DEPLAG name(response-variable) lag-term1 < lag-term2 ... > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  EVAL name = expression < / options > ;
  ID variable < option > ;
  IRREGULAR name < options > ;
  MODEL response = variables < / options > ;
  OBSINDEX variable ;
  OUTPUT < options > ;
  PARMS variables < / options > ;
  Programming statements ;
  SCORE options ;
  SIMSMOOTH options ;
  STATE name(dim)< options > ;
  TREND name(type)< options > ;

```

You can specify all statements multiple times, except the BY, ID, OBSINDEX, OUTPUT, SCORE, and SIMSMOOTH statements. The PROC CSSM statement, the ID statement, and at least one MODEL statement are required. An exception to this rule is during scoring and simulation smoothing. In that case, only the PROC CSSM statement, BY statement (when applicable), and SCORE statement (for scoring) or SIMSMOOTH statement (for simulation smoothing) are required. In addition to these statements, you can use most DATA step programming statements to define new variables that are needed for specifying different parts of the state space model.

NOTE: In the statement options described throughout this section, whenever you use a list to specify the elements of the system matrices, the list elements must all be of the same type: either all of them must be variables or all of them must be numbers. In addition, if the list contains more than one variable, then they cannot be of the array type. These are not serious restrictions. When the list contains mix of variables and numbers, you can redefine the numbers as constant variables. Similarly, you can reformulate a list that contains a mix of variables of array and non-array types as just one array by combining all its elements in a new array.

Functional Summary

Table 14.1 summarizes the statements and options that control the CSSM procedure. Most commonly needed scenarios are listed; for more information, see the individual statements.

Table 14.1 PROC CSSM Functional Summary

Description	Statement	Option
Data Set Options		
Specifies the input data set	PROC CSSM	DATA=
Writes series and component forecasts to an output data set	OUTPUT	OUT=
Model Specification Options		
Specifies the index variable	ID	
Defines variables as model parameters	PARMS	
Specifies a response variable and the associated observation equation	MODEL	
Specifies a state subsection	STATE	
Specifies the transition matrix of a state subsection	STATE	T
Specifies the disturbance covariance matrix of a state subsection	STATE	COV
Specifies the size of the diffuse initial condition of a state subsection	STATE	A1
Specifies the initial covariance matrix of a state subsection	STATE	COV1
Specifies a state subsection for a predefined structural model	STATE	TYPE=
Specifies the regressors in a state equation	STATE	W
Specifies the input vector in a state equation	STATE	SINPUT=
Specifies a component	COMPONENT	
Specifies a predefined trend component	TREND	
Likelihood Optimization Process Control Options		
Specifies the optimization technique	PROC CSSM	OPTIMIZER(TECH=)
Limits the number of iterations	PROC CSSM	OPTIMIZER(MAXITER=)
Outlier and Structural Break Detection Options		
Turns on the search for additive outliers (AO)		Default
Turns on the search for structural breaks in a state subsection	STATE	CHECKBREAK
Turns on the search for structural breaks in a state subsection associated with a trend	TREND	CHECKBREAK
Specifies the significance level for additive outlier tests	OUTPUT	AO(ALPHA=)

Table 14.1 *continued*

Description	Statement	Option
Limits the reported number of additive outliers	OUTPUT	AO(MAXNUM=)
Limits the reported number of additive outliers to a percentage of the series length	OUTPUT	AO(MAXPCT=)
Specifies the significance level for structural break tests	OUTPUT	BREAK(ALPHA=)
Limits the reported number of structural breaks	OUTPUT	BREAK(MAXNUM=)
Limits the reported number of structural breaks to a percentage of the series length	OUTPUT	BREAK(MAXPCT=)
Turns on the search for maximal state shock	OUTPUT	MAXSHOCK
Graphical Residual and Outlier Analysis Options		
Creates a panel of plots that consists of residual normality plots	PROC CSSM	PLOTS=RESIDUAL(NORMAL)
Creates the standardized residual plot against time	PROC CSSM	PLOTS=RESIDUAL(STD)
Creates a panel of plots that consists of prediction error normality plots	PROC CSSM	PLOTS=AO(NORMAL)
Creates the standardized prediction error plot against time	PROC CSSM	PLOTS=AO(STD)
Creates the plot of maximal state shock chi-square statistics against time	PROC CSSM	PLOTS=MAXSHOCK
Output Control Options		
Specifies the significance level of the forecast confidence limits	OUTPUT	ALPHA=
Prints the prediction error sum of squares table	OUTPUT	PRESS
Specifies a linear combination of components to be output	EVAL	
Global Printing and Plotting Options		
Turns off all printing for the procedure	PROC CSSM	NOPRINT
Turns on all printing options for the procedure	PROC CSSM	PRINTALL
Turns off all plotting for the procedure	PROC CSSM	PLOTS=NONE
Turns on all plotting options for the procedure	PROC CSSM	PLOTS=ALL
Printing State Equation System Matrix Options		
Prints the transition matrix that is associated with a state subsection	STATE	PRINT=T
Prints the disturbance covariance matrix that is associated with a state subsection	STATE	PRINT=COV
Prints the initial covariance matrix that is associated with a state subsection	STATE	PRINT=COV1
Prints the autoregressive coefficient matrix that is associated with a state subsection	STATE	PRINT=AR

Table 14.1 continued

Description	Statement	Option
Prints the moving average coefficient matrix that is associated with a state subsection	STATE	PRINT=MA
Printing Component, Series Forecast, and Smoothed Estimate Options		
Prints the series forecasts	MODEL	PRINT=FILTER
Prints the full-sample estimates of missing series values	MODEL	PRINT=SMOOTH
Prints the smoothed trend estimate	TREND	PRINT=SMOOTH
Prints the filtered trend estimate	TREND	PRINT=FILTER
Prints the smoothed component estimate	COMPONENT	PRINT=SMOOTH
Prints the filtered component estimate	COMPONENT	PRINT=FILTER
Prints the smoothed component estimate	EVAL	PRINT=SMOOTH
Prints the filtered component estimate	EVAL	PRINT=FILTER
BY Groups		
Specifies BY-group processing	BY	

PROC CSSM Statement

PROC CSSM <options> ;

The PROC CSSM statement is required. You can specify the following *options* in the PROC CSSM statement:

BREAKPEAKS

prints an alternate form of the break summary tables when the CHECKBREAK option is used in the **STATE** or **TREND** statement or when the MAXSHOCK option is used in the **OUTPUT** statement. In this alternate form, the summary tables report the significant peaks of the shock statistics curves; see “[Example 14.8: Diagnostic Plots and Structural Break Analysis](#)” on page 841 for examples of these curves.

DATA=*libref.data-table*

names the input data table for PROC CSSM to use. *libref.data-table* is a two-level name, where

<i>libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>libref</i> , see the section “ Using CAS Sessions and CAS Engine Librefs ” on page 721.
<i>data-table</i>	specifies the name of the input data table.

LIKE=DIFFUSE | MARGINAL

specifies the type of likelihood to use for parameter estimation. You can specify the following values:

DIFFUSE	specifies diffuse likelihood.
MARGINAL	specifies marginal likelihood.

By default, LIKE=DIFFUSE. For more information about different likelihood types, see the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769.

NOPRINT

turns off all the printing and plotting for the procedure. Any subsequent print options are ignored.

OPTIMIZER(< TECHNIQUE=*technique* > < HESSTYPE=*Hessian-type* > < MAXITER=*integer* >)

specifies options that are associated with the optimizer used in the maximum likelihood parameter estimation. The default settings of the optimization process are adequate for most problems. However, in some cases it might be useful to change the optimization technique, the Hessian type, or the maximum number of iterations. You can specify one of the following *techniques*:

ACTIVESET	uses the active-set method.
DBLDOG	uses the double-dogleg method.
INTERIORPOINT	uses the primal-dual interior point method.
IPDIRECT	uses the primal-dual interior point augmented Lagrangian method.
NEWRAP	uses the Newton-Raphson method.
QUANEW	uses the (dual) quasi-Newton method.
TRUREG	uses the trust region method.

By default, TECHNIQUE=TRUREG. If the technique is ACTIVESET, INTERIORPOINT, or IPDIRECT, you can specify one of the following *Hessian-types*:

BFGS	uses the quasi-Newton Broyden-Fletcher-Goldfarb-Shanno (BFGS) Hessian approximation.
FULL	uses the full Hessian.
SR1	uses the dense quasi-Newton symmetric rank 1 (SR1) Hessian approximation.

By default, HESSTYPE=FULL. The ACTIVESET, INTERIORPOINT, and IPDIRECT techniques and the Hessian types are documented in Chapter 10, “The Nonlinear Programming Solver” ([SAS/OR User’s Guide: Mathematical Programming](#)). The remaining techniques are documented in Chapter 6, “Nonlinear Optimization Methods” ([SAS/ETS User’s Guide](#)). You can alter the maximum number of iterations in the nonlinear optimization search by specifying a nonnegative *integer* as the MAXITER= value.

PLOTS <(global-plot-options)> = plot-request <(options)>

PLOTS<(global-plot-options)> = (plot-request <(options)> <...plot-request <(options)> >)

controls the plots produced with ODS Graphics. When you specify only one *plot-request*, you can omit the parentheses around it. Here are some examples:

```
plots=none
plots=all
plots=residual
plots=residual(normal)
plots=(maxshock residual(normal))
plots(unpack)=residual
```

If you do not specify any specific *plot-request*, then by default PROC CSSM produces the plot of standardized residuals against time. For general information about ODS Graphics, see Chapter 24, “Statistical Graphics Using ODS” (*SAS/STAT User’s Guide*).

Global Plot Options

The *global-plot-options* apply to all relevant plots generated by the CSSM procedure. The following *global-plot-option* is supported:

UNPACK

displays each graph separately. (By default, some graphs can appear together in a single panel.)

Specific Plot Options

The following list describes the specific *plot-requests* and their *options*:

ALL

produces all plots appropriate for the particular analysis.

AO< (prediction-error-plot-options)>

produces the prediction error plots—one for each response variable. You can specify the following *prediction-error-plot-options*:

NORMAL

produces a summary panel of the prediction error diagnostics, which consist of the following:

- histogram of prediction errors
- normal quantile plot of prediction errors

STD

produces a scatter plot of standardized prediction errors against time.

MAXSHOCK

produces a scatter plot of maximal state shock statistics against time.

NONE

suppresses all plots.

RESIDUAL <(residual-plot-options)>

produces the residuals plots—one for each response variable. You can specify the following *residual-plot-options*:

NORMAL

produces a summary panel of the residual diagnostics, which consist of the following:

- histogram of residuals
- normal quantile plot of residuals

STD

produces a scatter plot of standardized residuals against time.

For more information about the precise meaning of the terms *maximal state shock statistics* and *prediction errors*, see the section “[Delete-One Cross Validation and Structural Breaks](#)” on page 773.

PRINTALL

turns on all the printing options for the procedure. All subsequent NOPRINT options in the procedure are ignored.

STATEINFO

prints two tables that provide information about the composition of the state vector in terms of the components specified in the model. One table describes the composition of state α_t , and the other table describes the diffuse vector δ and the regressors, which are part of the initial condition specification α_1 . For more information about the state space model notation, see the section “[State Space Model and Notation](#)” on page 760.

ZSPARSE

enables the exploitation of the sparsity of the Z_t matrices in the observation equation during the modeling calculations (see the section “[State Space Model and Notation](#)” on page 760 for further information). The use of this option can improve the computational efficiency of models that have a large state dimension and sparse Z_t matrices—that is, many of their elements are zero. You should use the ZSPARSE option only when the state dimension is sufficiently large (at least 30) and a good percentage (at least 50%) of Z_t entries are zero; otherwise, the computational efficiency can in fact degrade. For example, the illustration that is discussed in the section “[Getting Started: CSSM Procedure](#)” on page 722 is a good candidate for the use of the ZSPARSE option:

```
proc cssm data=mylib.Cigar plots=residual zsparse;
```

BY Statement

BY *variables* ;

A BY statement can be used in the CSSM procedure to process a data set in groups of observations that are defined by the BY variables. The model specified by using the MODEL and other statements is applied to all the groups defined by the BY variables. The BY variables are one or more variables in the input data set. The BY variables cannot be used in the model specification; in particular, they cannot be used as response variables or regressors in a MODEL statement.

COMPONENT Statement

COMPONENT *name* = (*var1 var2 ...* | *number1 number2 ...*) * *state* </options> ;

COMPONENT *name* = *state*[*integer*] </options> ;

COMPONENT *name* = (*Variable* | *Number*) * *state*[*integer*] </options> ;

The COMPONENT statement specifies a component (a linear combination of state elements), named *name*. You can use *name* later as a term in the right-hand side of the MODEL statement, which defines the observation equation. The estimate of *name* is output to the OUT= data set that is specified in the OUTPUT statement. In addition, you can print the component estimate by using the PRINT= option.

The first form of the COMPONENT statement defines a component as a dot product of a state subsection *state* and a row vector (*var1 var2 ...*). The value of *state* can be the name of a state subsection that is defined by using a STATE statement elsewhere in the program, or it can be the name of the state that is associated with a trend component defined by using a TREND statement elsewhere in the program (see the section “TREND Statement” on page 755 for more information about the naming of the state that is associated with a trend component). The row vector (*var1 var2 ...*), which can be either a list of numbers or a list of variables, must be of the same dimension as the actual dimension of the state subsection. The dot product form—also called the explicit dot product form—of the component specification is unambiguous; however, it requires detailed knowledge of the state vector underlying the *state* specification. Suppose that *mystate* is a two-dimensional state defined by a STATE statement elsewhere in the program and that X1 and X2 are (numeric) predictor variables. The following are valid examples of the dot product form of the COMPONENT statement:

```
component c1 = (x1 x2) * mystate;
component c2 = (1 1) * mystate;
```

The second and the third forms of the COMPONENT statement are a shortened version of the first form. The second form defines the component as a particular element of *state*—for example, *state*[3] defines the component as the third element of *state*. The specified *integer* must lie between 1 and *dim*, the nominal dimension of *state*. The second form of component specification has another important use when the STATE statement that defines *state* uses the TYPE= option to set its type or when *state* is associated with a trend component. In these cases, the second form of the component specification assumes additional meaning when the nominal state dimension and the actual state dimensions differ (specifically the state types LL, SEASON, CYCLE, and VARMA and the states associated with all the trend types). For example, if *state* is a three-dimensional seasonal component, *state*[2] signifies an appropriate linear combination of *state* that results in the second of the three seasonals that constitute the three-dimensional seasonal. Similar interpretation holds for the CYCLE type. For more information, see the sections “Multivariate Season” on

page 783 and “[Predefined Structural Models](#)” on page 779. The third form extends the second form by permitting multiplication by a variable or a number.

NOTE: A component that is based on a state associated with a trend component cannot be used as a right-hand side term in any MODEL statement. That is, it is defined purely for output purposes (either printed or output to a data set). However, it can be used as a term in the expression that is specified in an [EVAL](#) statement to build more complex linear combinations for output.

You can specify the following *options* to print the filtered or smoothed estimate of the component:

PRINT=FILTER | SMOOTH

PRINT=(*< FILTER >* *< SMOOTH >*)

requests printing of the filtered or smoothed estimate of the specified component.

DEPLAG Statement

DEPLAG *name*(*response-variable*) *lag-term1* *< lag-term2 ... >* ;

The DEPLAG statement defines a term, named *name*, that consists of a linear combination of lagged response variables. You can use *name* later as a right-hand-side term in the MODEL statement for the response variable, as specified in *name*(*response-variable*). For a multivariate model, a separate DEPLAG statement is needed for each MODEL statement that has a right-hand-side term that involves lagged response variables. The linear combination of lagged response variables is specified by using one or more *lag-terms*. Each *lag-term* specifies the lags that are associated with one of the response variables.

A *lag-term* is specified in one of the following forms:

lag-response-variable(**LAGS**=*maximum-lag*)

lag-response-variable(**LAGS**=(*integer1 integer2 ...*))

lag-response-variable(**LAGS**=*maximum-lag* **COEFF**=(*number1 number2 ...*) | (*variable1 variable2 ...*))

lag-response-variable(**LAGS**=(*integer1 integer2 ...*) **COEFF**=(*number1 number2 ...*) | (*variable1 variable2 ...*))

The *lag-response-variable* in the lag term specification can be the same as the response variable that corresponds to the model equation (which is specified in *name*(*response-variable*)), or it can be a different response variable.

The first form of specification is useful when all lags up to the *maximum-lag*, which must be a positive integer, are present in the lag term. The second form is useful when only certain lags, which are specified as a list of positive integers in parentheses, are present. In these two cases, the lag coefficients are not specified and they are treated as unknown parameters to be estimated from the data.

The **COEFF**= option in the last two forms enable you to specify lag coefficients. The **COEFF**= option must follow the **LAGS**= option. You can use the **COEFF**=(*number1 number2 ...*) option to specify the lag coefficients as known values. Similarly, you can use the **COEFF**=(*variable1 variable2 ...*) option to specify user-defined variables as lag coefficients; the user-defined variables can be functions of parameters (which are defined by using the PARMS statement) and input variables. However, the lag coefficients cannot depend on any of the response variables. The number of coefficients specified in the **COEFF**= option must exactly equal the number of lags specified in the **LAGS**= option.

There can be at most one DEPLAG statement associated with a particular MODEL statement (you can specify all the needed lag terms in a single DEPLAG statement).

As an illustration, let lagsFORy1 and lagsFORy2 represent the following linear combinations of lagged response variables Y1, Y2, and Y3:

$$\text{lagsFORy1} = \theta_{11}Y_{1t-1} + \theta_{12}Y_{1t-2} + \theta_{22}Y_{2t-2} + \theta_{23}Y_{2t-3} + 1.2Y_{3t-1} - 2.1Y_{3t-2}$$

$$\text{lagsFORy2} = \text{Phi1}Y_{1t-1} + \text{Phi2}Y_{1t-2} + \theta_{21}Y_{2t-1}$$

where Phi1 and Phi2 denote user-defined variables and θ_{ij} denote generic parameters. You can specify lagsFORy1 (which is used in the model equation for Y1) and lagsFORy2 (which is used in the model equation for Y2) as follows:

```
deplag lagsFORy1(y1) y1(lags=2) y2(lags=(2 3)) y3(lags=2 coeff=(1.2 -2.1));
deplag lagsFORy2(y2) y1(lags=2 coeff=(phi1 phi2)) y2(lags=1);
... more statements ...;
model y1 = lagsFORy1 ...;
model y2 = lagsFORy2 ...;
model y3 = ...;
... more statements ...;
```

assuming that the right-hand side of the MODEL equation for Y3 does not have a term that involves lags of response variables.

The DEPLAG statement in PROC CSSM has the same purpose as the DEPLAG statement in PROC UCM (see Chapter 30, “[UCM Procedure](#)”). However, there are many differences in the syntax of the two statements, mainly because PROC CSSM supports much more complex models. The syntax difference between the two DEPLAG statements can be illustrated by considering the differencing specification— $(1 - B)(1 - B^{12}) = (1 - B - B^{12} + B^{13})$ —in the well-known airline model (ARIMA(0, 1, 1)(0, 1, 1)₁₂ model). You can specify the lag-term that is implied by the differencing in the airline model in PROC UCM as follows:

```
deplag lags=(1)(12) phi=1 1 noest;
```

In PROC CSSM the same specification has the following form:

```
deplag airLags(y) y(lags=(1 12 13) coeff=(1 1 -1));
```

Both these specifications define the same lag-term: $(y_{t-1} + y_{t-12} - y_{t-13})$.

For an example of the use of lagged response variables in a model specification, see “[Example 14.13: Bivariate Model for Sales of Mink and Muskrat Furs](#)” on page 871. For more information about models that have dependent lags, see the section “[Models with Dependent Lags](#)” on page 785.

NOTE: Models that have lagged response variables are permitted only if the data form a time series (either univariate or multivariate). The CSSM procedure adds one more restriction on the models that use lagged response variables: the variables in the list that define a component in any of the COMPONENT statements must be free of unknown parameters. This restriction is artificial and is made primarily to reduce the overall complexity of the model. In future versions of the CSSM procedure, this restriction might go away.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[Displayed Output](#)” on page 799. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the output table name.

key specifies *key* as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[Displayed Output](#)” on page 799. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

EVAL Statement

EVAL *name* = *number1*variable1 + number2*variable2 + ...* </options> ;

The EVAL statement defines a linear combination, named *name*, of the terms used in the right-hand side of a MODEL statement. You can specify any variables (for example, predictor variables and names of components) in the expression of the EVAL statement; however, you cannot specify in this expression any observation disturbances that are specified by the IRREGULAR statement and any model terms that are specified by the DEPLAG statement. Suppose C1 and C2 are two components (defined by COMPONENT statements elsewhere in the program), T1 is a trend component, and X1 is a regression variable used in a model. The following are valid examples of the EVAL statement:

```
eval e1 = c1 - c2;
eval e2 = t1 + c1 + x1;
eval e2 = t1 + 2*c1 - 1.5*x1;
```

The estimates of linear combinations defined by the EVAL statement (for example, E1, E2, and E3) are output to the OUT= data set that is specified in the OUTPUT statement.

The components used in a given EVAL expression must correspond to distinct state subsections. This requirement is imposed only to simplify the overall readability of the program and does not limit the type of linear combinations that can be specified; if two components in the right hand side of an EVAL expression share the same state subsection, a new component that combines the effect of these two components can always be defined.

In addition, you can print these estimates by using the following PRINT= options:

PRINT=FILTER | SMOOTH

PRINT=(< FILTER > < SMOOTH >)

requests printing of the filtered or smoothed estimate of the specified linear combination.

NOTE: The expression builder in the EVAL statement is primitive. For example, you cannot use parentheses to group terms.

ID Statement

ID *variable* <option> ;

The ID statement names a numeric variable to associate a sequence value—usually related to a timestamp—with the observations in the input data set. The ID statement is required. Internally, the CSSM procedure sorts the observations in ascending order by the ID variable within a BY group. If you also supply an **OBSINDEX** variable, the observations are sorted by both the ID variable and the OBSINDEX variable. Often the ID variable's values are SAS date, time, or datetime values, and each observation within a BY group has a unique ID value. Generally, however, the ID variable can be any numeric variable, and there can be multiple observations with the same ID value. If the ID values are SAS date, time, or datetime values, you can specify the associated unit of time—for example, day, week or month—by using the INTERVAL= option. A variable, **_ID_DELTA_**, is automatically created that can be used as any input data set variable in the programming statements. **_ID_DELTA_** contains the distance between two successive ID values. The first **_ID_DELTA_** value is arbitrarily taken as 1. If the INTERVAL= option is specified, the distance between the ID values is

measured in terms of the number of intervals; therefore, for regularly spaced data, `_ID_DELTA_` is identically equal to 1. You can specify the following *option* in the ID statement:

INTERVAL=*value*

specifies the unit of time interval that is used to measure the ID values. `INTERVAL=`*value* is used in conjunction with the ID variable to verify that the input data are in the proper order. For a complete discussion of the supported intervals, see Chapter 4, “Date Intervals, Formats, and Functions” ([SAS/ETS User’s Guide](#)).

IRREGULAR Statement

IRREGULAR *name* < *options* > ;

The IRREGULAR statement specifies a one-dimensional white noise component, which can be used to specify the observation error in a MODEL statement. You can specify the following *options* in the IRREGULAR statement:

PRINT=SMOOTH

requests printing of the smoothed estimate of the specified irregular component.

VARIANCE=*variable* | *number*

specifies the variance of the white noise. Any nonnegative value, including 0, is permissible. If the *variable* contains unknown parameters, they are estimated from the data. Similarly, if the `VARIANCE=` option is not specified, the variance is estimated from the data.

MODEL Statement

MODEL *response* = *variables* < / *options* > ;

A MODEL statement specifies an observation equation that describes a response variable as a sum of regression effects and components that are defined in the program. The response variable must be a numeric variable from the input data set. The variables used in the right-hand side of the model expression can be numeric variables from the input data set, numeric variables defined by using programming statements, or names of components that are specified in the COMPONENT, DEPLAG, TREND, or IRREGULAR statements.

For a multivariate model, a separate MODEL statement is needed for each of the response variables. In this case, the observation errors, which are specified in an IRREGULAR statement, must be different in each MODEL statement.

The components that are specified in a given MODEL statement must correspond to distinct state subsections. This requirement is imposed only to simplify the overall readability of the program and does not limit the type of models that can be specified; if two components on the right-hand side of a MODEL statement share the same state subsection, a new component that combines the effect of these two components can always be defined.

You can specify the following *options* in the MODEL statement; they must be separated from the list of terms in the right-hand side of the model equation by a slash (/):

AGGREGATE(START=*startFlag*)**SUM(START=*startFlag*)**

produces a table of full-sample predictions of the temporally aggregated values of the response variable that is specified in the MODEL statement. The variable that you specify in the START= option, *startFlag*, must be a zero-one variable that flags the start of an aggregation interval—equal to 1 at the start of an interval and 0 otherwise. For example, you can use this option to obtain the forecasts of weekly (or monthly) totals from a daily series. In this case, the value of *startFlag* is 1 at the start of the week (or month) and 0 otherwise. For more information, see the section “[Temporal Aggregation and Temporal Distribution](#)” on page 790. This option is valid only if the data form a time series (either univariate or multivariate). If you use the AGGREGATE option in a MODEL statement, you cannot use the DISTRIBUTE option in the same statement or in another MODEL statement.

DISTRIBUTE(START=*startFlag*)

indicates that the response variable that is specified in the MODEL statement is a temporally aggregated version of an unobserved variable. The variable that you specify in the START= option, *startFlag*, must be a zero-one variable that flags the start of an aggregation interval—equal to 1 at the start of an interval and 0 otherwise. This option can be used only when the data form a time series (either univariate or multivariate) and when the overall model specification does not contain terms that involve lagged response variables (that is, the model specification does not involve the use of [DEPLAG](#) statements). If you use the DISTRIBUTE option in a MODEL statement, you cannot use the AGGREGATE option in the same statement or in another MODEL statement. For more information, see the section “[Temporal Aggregation and Temporal Distribution](#)” on page 790.

PRINT=FILTER | SMOOTH**PRINT=(< FILTER > < SMOOTH >)**

requests printing of the filtered or smoothed estimate of the specified response variable. The filtered estimate is produced during the filtering phase, and the smoothed estimate is produced by the smoothing phase of the Kalman filter and smoother algorithm. The filtered estimate is also called the one-step-ahead forecast of the response variable. The smoothed estimate corresponds to the full-sample prediction of the response variable. Because the full-sample prediction of a nonmissing response value is that value itself, full-sample predictions are printed only for the missing response values.

OBSINDEX Statement

OBSINDEX *variable* ;

The OBSINDEX statement specifies a numeric input data variable that, together with the ID variable, uniquely identifies the observations within a BY group. This statement is essential for the reproducibility of the computations in situations where there can be multiple observations in a given BY group that have the same ID value. These types of situations arise when you are analyzing panels of time series or when the data are longitudinal. For examples of using this statement, see the section “[Getting Started: CSSM Procedure](#)” on page 722, “[Example 14.7: A Dynamic Factor Model for the Yield Curve](#)” on page 831, “[Example 14.9: Variable Bandwidth Smoothing for Longitudinal Data](#)” on page 848, and “[Example 14.11: Dynamic Panel Model for the Cigar Data](#)” on page 859.

OUTPUT Statement

OUTPUT *<options>* ;

The OUTPUT statement creates an optional output data set and also provides options to control certain aspects of the procedure output. If the OUT= option is specified, then an output data set is created to store estimates of the model components and series forecasts. If the OUT= option is omitted, then no data set is created by the OUTPUT statement. Other options in the OUTPUT statement produce additional information in the printed output generated by the procedure. For example, the AO and BREAK options control the search for additive outliers and structural breaks in the data, respectively.

AO(*< ALPHA=number > < MAXNUM=number > < MAXPCT=number >*)

controls the additive outlier search (see the section “[Delete-One Cross Validation and the Additive Outlier Detection](#)” on page 773 for more information). The ALPHA= suboption specifies the significance level for reporting the outliers. The default is ALPHA=0.05. The MAXNUM= suboption limits the number of outliers to search. The default is MAXNUM=5. The MAXPCT= suboption is similar to the MAXNUM= suboption. In the MAXPCT= option you can limit the number of outliers to search for according to a percentage of the series length. The default is MAXPCT=1. When you specify both of these options, the lesser of the two search numbers is used.

ALPHA=number

specifies the significance level of the forecast confidence intervals. For example, ALPHA=0.05, which is the default, results in a 95% confidence interval.

BREAK(*< ALPHA=number > < MAXNUM=number > < MAXPCT=number >*)

controls the structural break search (for more information, see the section “[Structural Breaks in the State Evolution](#)” on page 774). In order for this option to have any effect, the CHECKBREAK option in one of the STATE or TREND statements, or the MAXSHOCK option in the OUTPUT statement, must be turned on. The ALPHA= suboption specifies the significance level for reporting the breaks. The default is ALPHA=0.05. The MAXNUM= suboption limits the number of breaks to search. The default is MAXNUM=5. The MAXPCT= suboption is similar to the MAXNUM= suboption. In the MAXPCT= option, you can limit the number of breaks to search for according to a percentage of the number of distinct time points in the data. The default is MAXPCT=1. When you specify both of these options, the lesser of the two search numbers is used.

MAXSHOCK

causes the computation of the maximal state shock chi-square statistic at each distinct time point in the input data set. These statistics are output to the data set that is specified in the OUT= option. A time series plot of these statistics is produced if the PLOTS=MAXSHOCK option is specified in the PROC CSSM statement. These statistics are useful for detecting structural breaks in the state evolution process. This option can be computationally expensive for a model with large state size. For more information, see the section “[Structural Breaks in the State Evolution](#)” on page 774.

OUT=libref.data-table

specifies an output data set for the forecasts. The output data set contains the ID variable, the OBSINDEX variable (if specified), the response variables, the one-step-ahead and out-of-sample response variable forecasts, the forecast confidence intervals, the smoothed values of the response series, and the one-step-ahead and smoothed estimates of the model components—including expressions that are defined by using the EVAL statement. For more information, see the section “[OUT= Data Set](#)” on page 802.

PDV

causes the inclusion of the variables (variables in the program data vector) that are defined by using the programming statements in the CSSM procedure in the OUT= data set. The parameters defined by the PARMs statement are also included. The output data set contains the values of these variables evaluated for all the rows in the input data set that is specified in the DATA= option. The parameters in the PARMs statement contain their estimated values.

PRESS

prints the prediction error sum of squares (PRESS) and the generalized cross validation error sum of squares (GCV). The PRESS table also reports the number of summands that are used in these sums of squares. For more information, see the section “[Delete-One Cross Validation and the Additive Outlier Detection](#)” on page 773.

PARMS Statement

PARMS *variable*< =*number*> *variable*< =*number*> < / *options* > ;

The PARMs statement declares the parameters of a model and optionally sets their initial values. You can also specify the lower and upper limits of their validity range. The parameters declared by using the PARMs statement are called *named parameters* throughout this chapter. A model can have additional parameters: any unspecified quantity in the model specification becomes part of the parameter vector. You can specify the following *options*:

LOWER=(*number1 number2 ...*)

LOWER=(*number*)

specifies the lower bounds for the specified parameters. The list can contain exactly one number, which is taken to be the lower bound for all the listed parameters in the statement, or it must contain as many values as the number of parameters specified. A missing value, denoted by ., is a permissible value, which signifies that the parameter has no lower bound.

UPPER=(*number1 number2 ...*)

UPPER=(*number*)

specifies the upper bounds for the specified parameters. The list can contain exactly one number, which is taken to be the upper bound for all the listed parameters in the statement, or it must contain as many values as the number of parameters specified. A missing value, denoted by ., is a permissible value, which signifies that the parameter has no upper bound.

Programming Statements

To define the model, you can use most of the programming statements that are allowed in the SAS DATA step. For more information, see the *SAS DATA Step Statements: Reference*. For the most part, the syntax of programming statements used in PROC CSSM is identical to that used in the MODEL procedure (see Chapter 24, “MODEL Procedure” (*SAS/ETS User’s Guide*)) and the NLMIXED procedure (see Chapter 89, “The NLMIXED Procedure” (*SAS/STAT User’s Guide*)). However, there are some restrictions: the DATA step lagging and differencing functions are not allowed, and the use of character variables in the DATA step expressions is not permitted. These are not serious restrictions; usually you can overcome them by adding the variables that are created by such operations to the input data set before its use in the CSSM procedure.

SCORE Statement

SCORE <IN=score-table> <OUT=score-table> ;

The SCORE statement enables you to use the fitted model for scoring. By using the IN= and OUT= options in the SCORE statement, you can create, reuse, and update a score store. The score store contains sufficient context information so that you can perform many tasks, such as scenario analysis and forecasting for streaming data, without having to fit the model again. The score store is a binary large object (BLOB) table, which uses a binary file format that cannot be edited or viewed like a typical data table. For more information about using state space models for scoring, see the section “[Scoring with State Space Models](#)” on page 794.

IN=score-table

specifies a previously created score store to score the rows in the input data table. The rows in the input data table must represent a continuation of the sequential observation process that was used to create the specified score store. In particular, for a given BY group, the smallest value of the ID variable in the current input data table must be the same as or larger than the largest value of the ID variable in the data table that you used to create the score store. When you specify the IN= option in the SCORE statement, the statements that are used for model specification (such as the MODEL, STATE, and TREND statements, and so on) are not needed; in addition to the PROC CSSM statement and the SCORE statement, only the [BY](#) and [OUTPUT](#) statements are used when needed.

OUT <(positive-integer)> =score-table

specifies an output score-store table for storing the context information. The score store contains all the information that is needed for resuming the forecasting process from the last observation of a given BY group. If the OUT(*k*)= form of the option is used (where *k* is some positive integer), the score store keeps additional historical information so that the forecasting begins *k* time instances before the last observation. If *k* is larger than the total number of time points in a BY group, it is appropriately reduced. You should use the OUT(*k*)= form only at the time of score-store creation. In subsequent calls, the value of *k* that you specify at creation time remains in effect and cannot be changed. That is, *k* in OUT(*k*) is ignored when you also specify the IN= option.

When you specify both the IN= and OUT= options, the table names that you specify must be distinct. You can use the IN= and OUT= options by themselves or together. These options are used as follows:

Only OUT= Used for initial score-store creation.

Only IN= Used for scoring the rows of the input data table. This is useful for scenario analysis, where the scoring is done repeatedly, each time for different predictor settings. For an example of this use case, see “[Example 14.19: Scenario Analysis Using Scoring](#)” on page 894.

Both IN= and OUT= Used for creating an updated score store that incorporates new information. This is used in the streaming data setting, where new sets of observations become available on a regular basis and the score store must be updated to take into account the latest information. For an example of this use case, see “[Example 14.20: Monitoring Streaming Data Using Scoring](#)” on page 897.

NOTE: Scoring is not currently supported in situations where temporal aggregation or temporal distribution is requested. For more information, see the AGGREGATE and DISTRIBUTE options in the **MODEL** statement.

SIMSMOOTH Statement

SIMSMOOTH < **INMODEL**=*blob-table* > < **OUTMODEL**=*blob-table* > < *options* > ;

The SIMSMOOTH statement enables you to use the fitted model for simulation smoothing. (For more information, see the section “[Simulation Smoothing](#)” on page 787.) The **OUTMODEL=** option enables you to store the fitted model in a binary large object (BLOB) table to use later for simulation smoothing. The **INMODEL=** option enables you to load a previously stored fitted model and use it for simulation smoothing. The other options in this statement enable you to control different parts of the simulation smoothing process. For an illustration of simulation smoothing, see “[Example 14.21: Simulation-Smoothing-Based Sampling Distribution of the Sum of Forecasts](#)” on page 901.

You can specify the following *options*:

INMODEL=*blob-table*

specifies a previously created BLOB table that contains the fitted model for the input data table. When you specify this option in the SIMSMOOTH statement, the statements that are used for model specification (such as the MODEL, STATE, and TREND statements) are not needed. In addition to the PROC CSSM and SIMSMOOTH statements, only the options that control the simulation smoothing steps, such as the **NSIM=**, **SEED=**, and **OUTPRED=** options, are used.

NSIM=*number*

specifies the number of simulations to generate, where *number* is a positive integer. By default, NSIM=1.

OUTCOMP=*libref.data-table*

specifies an output data table to store the simulated components. This table contains the ID statement variable, the OBSINDEX statement variable (if specified), and the simulated components. For more information, see the section “[Output Tables Associated with Simulation Smoothing](#)” on page 788.

OUTMODEL=*blob-table*

specifies a BLOB table to store the fitted model specification for later use in simulation smoothing. When you specify this option in the SIMSMOOTH statement, other options in this statement (such as the INMODEL= or NSIM= option) are ignored.

OUTPRED=*libref.data-table*

specifies an output data table to store the simulated predictions of the missing response values. This table contains the ID statement variable, the OBSINDEX statement variable (if specified), and the simulated predictions of the missing response values. For more information, see the section “[Output Tables Associated with Simulation Smoothing](#)” on page 788.

OUTSTATE=*libref.data-table*

specifies an output data table to store the simulated latent states. This table contains the ID statement variable, the OBSINDEX statement variable (if specified), and the simulated states. For more information, see the section “[Output Tables Associated with Simulation Smoothing](#)” on page 788.

SEED=*seed*

specifies the start of the set of random number seeds to use during the simulation process. As an example, if the number of simulations that you specify in the NSIM= option is 100 and SEED=5, then the seeds between 5 and 104 are sequentially used for random number generation during the 100 simulations. By default, SEED=1.

The SIMSMOOTH statement is used both in creating the fitted-model-specification BLOB table (using the OUTMODEL= option) and later in simulation smoothing (using the INMODEL= option).

NOTE: Simulation smoothing is not currently supported in situations where temporal aggregation or temporal distribution is requested. For more information, see the AGGREGATE and DISTRIBUTE options in the MODEL statement.

STATE Statement

STATE *name (dim)< options>* ;

The STATE statement specifies a subsection of α_t , the overall state vector at time t (for more information, see the section “[State Space Model and Notation](#)” on page 760). Consider the state equations that define the state space model:

$$\begin{aligned}\alpha_{t+1} &= \mathbf{T}_t \alpha_t + \mathbf{W}_{t+1} \gamma + \mathbf{c}_{t+1} + \eta_{t+1} \\ \alpha_1 &= \mathbf{c}_1 + \mathbf{A}_1 \delta + \mathbf{W}_1 \gamma + \eta_1\end{aligned}$$

You can specify multiple STATE statements, each specifying a separate subsection. It is assumed that the subsections that are specified by using different STATE statements are mutually independent. This independence assumption implies a block-diagonal structure for the transition matrices $\mathbf{T}_t$ and the disturbance covariances $\mathbf{Q}_t$ for all $t \geq 1$. An appropriate block structure also applies to $\mathbf{W}_t$ and $\mathbf{A}_1$. The *options* in the STATE statement provide complete control over the description of the relevant blocks of $\mathbf{T}_t$, $\mathbf{Q}_t$, $\mathbf{W}_t$, and $\mathbf{A}_1$. The argument *dim* (a positive integer in *name (dim)*) specifies the nominal dimension of this subsection. In most situations, the nominal dimension and the actual dimension of the state subsection are the same. However, when you specify the TYPE= option, the actual dimension of the state subsection can be different

from the nominal dimension. The **TYPE=** option simplifies the state specification task for some commonly needed models.

NOTE: The **T**, **COV**, **W**, and **COV1** options, described later in this section, specify the relevant blocks of T_t , Q_t , W_t , and Q_1 , respectively. The structure of these matrix blocks is described in a similar way in the option descriptions. For example, the specification **COV(I)** corresponds to the identity form, **COV(D)** corresponds to the diagonal form, and **COV(G)** corresponds to the general form of the Q_t block.

You can use the following *options* in the STATE statement to specify the system matrices T_t , Q_t , W_t , and A_1 and to request printing of their estimates when they contain unknown parameters. You can also request the checking of unexpected changes—structural breaks—in the evolution of this state subsection by using the **CHECKBREAK** option.

A1(nd)

treats the last nd elements of the state subsection as diffuse. This becomes the dimension of the relevant subsection of the diffuse vector δ . The A_1 block is created by using appropriate columns of the identity matrix. The value of nd must lie between 1 and the nominal dimension, dim . The absence of this option signifies that this subsection of α_t is nondiffuse. If both the **COV1** and **A1** options are specified, the last nd rows and columns of the matrix specified in the **COV1** option are taken to be 0. This option cannot be used together with the **RANK=** option of the **COV1** option.

CHECKBREAK< (ELEMENTWISE | OVERALL) >

turns on the checking of breaks for this state subsection. The **ELEMENTWISE** suboption requests the elementwise checking of any unexpected change in the state subsection as it evolves from one time point to the next. The **OVERALL** suboption requests a similar check for the entire state subsection—that is, in this case the change is measured as a multidimensional change. The **ELEMENTWISE** suboption is the default. Unless the **PRINT=BREAKDETAIL** option is specified, only a summary of the most significant breaks is produced. If the **PRINT=BREAKDETAIL** is specified, tables that contain the break significance statistics at every distinct time point are produced—one for the **ELEMENTWISE** suboption and one for the **OVERALL** suboption. For more information about the structural break detection process, see the section “[Structural Breaks in the State Evolution](#)” on page 774. For an example of the use of the **CHECKBREAK** option, see “[Example 14.8: Diagnostic Plots and Structural Break Analysis](#)” on page 841.

COV(D) <= (var1 var2 ...) | (number1 number2 ...) >

COV(G) <= (var1 var2 ...) | (number1 number2 ...) >

COV(I) <= (variable) | (number) >

COV(RANK=integer)

specifies the relevant block of the disturbance covariance Q_t (for $t \geq 2$) in the transition equation. As with the **T** option, the absence of this option signifies that this Q -block consists of only zeros. The structure of the Q -block is also similarly specified. However, the following differences exist:

- The list that is specified to form the covariance must result in a symmetric, positive semidefinite matrix. For an example, see “[Example 14.5: A User-Defined Trend Model](#)” on page 823.
- You can specify a rank constraint on the Q -block by specifying **COV(RANK=integer)**, where the specified integer must lie between 1 and dim . A rank constraint is permissible only for the general form and only when its elements are not specified by using a list.
- The convention of treating unset variables as structural zeros, which is used in specifying sparsity of the T -block, is not used in the Q -block specification. Whenever you explicitly specify the

entries of the Q-block by specifying a list of variables in parentheses, all variables in the list must evaluate to nonmissing values.

The following examples illustrate different ways of specifying a Q-block. It is assumed that $dim = 2$.

- COV(G) specifies a general-form Q-block, which contributes $(2 * (2 + 1))/2 = 3$ unspecified elements to the parameter vector θ .
- COV(RANK=1) specifies a rank-one Q-block.

COV1(D) <= (var1 var2 ...) | (number1 number2 ...) >

COV1(G) <= (var1 var2 ...) | (number1 number2 ...) >

COV1(I) <= (variable) | (number) >

COV1(RANK=integer)

specifies the relevant block of the initial state covariance Q_1 . The different options in this case have the same meaning as the options of the **COV** option. However, the following differences exist:

- If the elements of Q_1 are specified by a list of variables in parentheses, then these variables must evaluate to constant values. In particular, they can depend on parameters that are specified by the **PARMS** statements; however, they cannot depend on any of the input data columns.
- If the initial condition is partially diffuse (that is, the diffuse dimension nd specified in the **A1** option is nonzero), the last nd rows and columns of the matrix specified in **COV1** are taken to be zero. Moreover, if the elements of Q_1 are specified by a list, its number of elements must correspond to a matrix of dimension $(dim - nd)$.

PRINT=AR | BREAKDETAIL | COV | COV1 | MA | T

PRINT=(<AR> <BREAKDETAIL> <COV> <COV1> <MA> <T>)

requests printing of the respective system matrices and the printing of the break statistics at each distinct time point. You can specify **PRINT=AR** or **PRINT=MA** only if you specify the **TYPE=VARMA** option. If any of these matrices are time-varying, the matrix that corresponds to the first time instance is printed. For the **BREAKDETAIL** suboption to have any effect, the **CHECKBREAK** option must be turned on. If **TYPE=** option is used, the result of **PRINT=COV** can be different than the matrix supplied in the **COV=** option.

SINPUT = (var1 var2 ...) | (number1 number2 ...)

specifies the relevant dim -dimensional block of the state input vector c_t . The absence of this option signifies that this block of the c_t vector consists of only zeros. If the elements of c_t are specified by a list of variables in parentheses, then these variables must be independent of unknown parameters. In particular, they cannot be functions of parameters that are defined by the **PARMS** statements.

T(D) <= (var1 var2 ...) | (number1 number2 ...) >

T(G) <= (var1 var2 ...) | (number1 number2 ...) >

T(I) <= (variable) | (number) >

specifies the relevant block of the transition matrix T_t . The absence of this option signifies that this block consists of only zeros. You can specify the structure of the T-block by specifying **T(I)** for the identity form, **T(D)** for the diagonal form, and **T(G)** for a general unstructured form. In addition, you can explicitly specify the entries of the T-block by specifying a list of numbers in parentheses, or by specifying in parentheses a list of variables that are defined by using the programming statements. The

unspecified elements of the T-block are included in the list of parameters to be estimated from the data. If the elements of the T-block are supplied by a list in parentheses, the number of elements in the list depends on its structure. For the diagonal form, the list must contain exactly *dim* elements. In the case of the identity form—T(I)—the block is already fully specified; however, a specification T(I)=(*variable*) is understood to mean that the identity block is scaled by the specified *variable* (or a *number*). In the general case—T(G)—the list must consist of *dim* * *dim* elements, specified in a rowwise fashion. An inappropriate number of elements in the list results in a syntax error.

The following examples illustrate different ways of specifying the transition matrix. It is assumed that *dim* = 2.

- T(I) specifies that the T-block is a two-dimensional identity matrix.
- T(D) specifies that the T-block is a two-dimensional diagonal matrix. The two unspecified diagonal entries become part of the parameter vector θ .
- T(D)=(1.1 2) fully specifies the two-dimensional diagonal T-block.
- T(D)=($X_1 X_2$) specifies a two-dimensional diagonal T-block where the diagonal elements are dynamically calculated based on the values of the variables X_1 and X_2 . In this case the T-block can change with time if X_1 or X_2 changes with time.
- T(G) specifies a general form T-block (with $2^2 = 4$ unspecified elements).
- T(G)=($X_1 X_2 X_3 X_4$) specifies a general form T-block where the first row is formed by X_1 and X_2 , and the second row is formed by X_3 and X_4 .

In practice the transition matrix is often sparse—that is, many of its elements are 0. The algorithms in the CSSM procedure exploit this sparsity structure for computational efficiency. Whenever you explicitly specify the entries of the T-block by specifying a list of variables in parentheses, you can leave the variables that correspond to the zero elements *unset*. These unset variables are treated as structural zeros by the CSSM procedure. The section “[Sparse Transition Matrix Specification](#)” on page 765 further explains how to use this sparsity convention.

TYPE=WN

TYPE=RW

TYPE=LL <(SLOPECOV(I | D | G) <= (var1, var2, ...) | (number1, number2, ...) >)>

TYPE=LL (SLOPECOV(RANK=*integer*)

TYPE=SEASON (LENGTH=*integer* <DROPH=*number-list*> <KEEPPH=*number-list*>)

TYPE=CYCLE <(<CT> <RHO=*variable* | *number*> <PERIOD=*variable* | *number*>)>

TYPE=VARMA (<p <(I | D)> =*integer*> <q<(D)> =*integer*>)

specifies a state subsection that corresponds to the specified type. You can specify either a number or a variable for the RHO= and PERIOD= suboptions. When TYPE=VARMA, the autoregressive and moving average orders can be at most 1 ($0 \leq p \leq 1$ and $0 \leq q \leq 1$). Moreover, by using the D and I flags with the order specification, you can impose additional structure on the autoregressive and moving average coefficient matrices—for example, specifying TYPE=VARMA(P=1) implies a VAR(1) model with general autoregressive coefficient matrix, whereas specifying TYPE=VARMA(P(D)=1) implies a VAR(1) model with diagonal autoregressive coefficient matrix. If you specify the TYPE= option, the T, COV1, SINPUT, and A1 options are not needed. In fact they are ignored, since the transition matrix T_t and the matrices in the initial condition (Q_1 and A_1) are implicitly defined by the choice of the type. However, the COV and W options can be useful. In fact, the specification of the

COV option does play a key role in the eventual form of $\mathbf{Q}_t$ —the covariance of the disturbance term in the transition equation. For the types LL, CYCLE, SEASON, and VARMA, the dimension of the resulting state subsection is a certain multiple of *dim*, the nominal dimension in the STATE statement. For example, the following specification results in a state subsection, named `cycleState`, of dimension $2*dim$:

```
state cycleState(dim) cov(g) type=cycle;
```

The name `cycleState` corresponds to the state underlying a *dim*-dimensional cycle component. All of these special state types require that the data be regular (replication is permissible); the only exception is **TYPE=Cycle(CT)**, which defines a continuous-time cycle and is applicable to any data type. [Table 14.2](#) summarizes some of this information for easy reference. For more information about these state types, see the section “[Predefined Structural Models](#)” on page 779.

The **TYPE=LL** specification results in a state that corresponds to a multivariate local linear trend. It is governed by two covariance matrices: the **COV** option specifies the covariance that corresponds to the level equation, and the **SLOPECOV** suboption specifies the covariance used in the slope equation. The omission of the **SLOPECOV** suboption signifies that the covariance used in the slope equation is zero. The form of the **SLOPECOV** suboption is exactly the same as that of the **COV** option.

The **TYPE=Cycle** option results in a state that corresponds to a (stochastic) cycle. By default, this cycle is assumed to be for the regular data type. If **TYPE=Cycle(CT)**, the resulting cycle is applicable to any data type. The CT option is available only for *dim* = 1; that is, only a univariate cycle is available for the irregular data type. The cycle specification depends on a covariance matrix and two numbers: the damping factor **RHO** and the cycle period **PERIOD**. The covariance can be specified by the **COV** option. The damping factor is specified by the **RHO=** suboption; its value must lie between 0.0 and 1.0. The cycle period can be specified by the **PERIOD=** suboption. If the CT suboption is not included, the period value must be larger than 2.0. On the other hand, if the CT suboption is included, its value must be strictly positive. If these parameters are not specified, they are estimated from the data.

The **TYPE=SEASON(LENGTH=integer)** specifies a multivariate trigonometric season that contains the full set of harmonics (for more information, see “[Multivariate Season](#)” on page 783). In some cases, you might want to drop some of the harmonics from this complete set to obtain a more parsimonious trigonometric season specification. You can use the **DROPH=** (to drop) or **KEEPH=** (to keep) suboption to control the harmonics that are included in the season specification as follows:

```
TYPE=SEASON(
  LENGTH=integer
  < DROPH=number-list | n TO m BY p>
  < KEEPH=number-list | n TO m BY p>
)
```

The **DROPH=** and **KEEPH=** lists can include any integer between 1 and *LENGTH*/2 if the season length is even and any integer between 1 and (*LENGTH* – 1)/2 if the season length is odd. For example, the following specification results in a specification of a trigonometric season with a season length 12 that consists of only the first four harmonics ζ_j , $j = 1, 2, 3, 4$:

```
type=season(length=12 DROPH=5 6) ...;
```

The last two *high*-frequency harmonics, ξ_5 and ξ_6 , are dropped. The DROPH= suboption cannot be used with the KEEPH= suboption.

Table 14.2 Summary of Predefined State Types

Type	Description	Parameters	State Dimension
WN	<i>dim</i> -variate white noise	COV	<i>dim</i>
RW	<i>dim</i> -variate random walk	COV	<i>dim</i>
LL	<i>dim</i> -variate local linear	COV, SLOPECOV	$2 * \textit{dim}$
SEASON(LENGTH= <i>length</i>)	<i>dim</i> -variate season	COV	$(\textit{length} - 1) * \textit{dim}$
CYCLE	<i>dim</i> -variate cycle	COV, RHO, PERIOD	$2 * \textit{dim}$
VARMA(P= <i>p</i> Q= <i>q</i>)	<i>dim</i> -variate VARMA(<i>p</i> , <i>q</i>)	COV, AR, MA	$\textit{dim} * \max(\textit{p}, \textit{q} + 1)$

W(D)= (*var1 var2 ...*) | (*number1 number2 ...*)

W(G)= (*var1 var2 ...*) | (*number1 number2 ...*)

W(I) <= (*variable*) | (*number*) >

specifies the relevant block of the design matrix $\mathbf{W}_t$ in the transition equation. The W-block is of dimension $sdim \times sg$, where *sdim* denotes the actual dimension of the state subsection (which can be the same as *dim*, the nominal dimension, or different if the TYPE= option is used) and *sg* denotes the desired size of the subsection of the overall state regression vector $\boldsymbol{\gamma}$. The absence of this option signifies that the state equation does not contain any regression effects. The number of variables supplied in the W(G)= list option must be a multiple of *sdim*. For example, if *sdim* = 4 and the W(G)= list contains 8 variables, then the implied size of $\boldsymbol{\gamma}$ subsection is 2. If the W(D)= or W(I)= option is used, then the W-block is assumed to be an *sdim*-dimensional diagonal matrix and the W(D)= list must contain exactly *sdim* variables. For examples of the use of this option, see “[Example 14.8: Diagnostic Plots and Structural Break Analysis](#)” on page 841, “[Example 14.10: A Transfer Function Model for the Gas Furnace Data](#)” on page 853, and “[Example 14.11: Dynamic Panel Model for the Cigar Data](#)” on page 859.

TREND Statement

TREND *name* (*type*)< *options* > ;

The TREND statement defines a term in the model that follows a stochastic pattern of a certain predefined type. The *options* in the TREND statement enable you to specify a wide variety of commonly used stochastic patterns. Each TREND statement in effect stands for a special pair of STATE and COMPONENT statements. You can specify more than one TREND statement. Each separate TREND statement defines a component that is assumed to be independent of all other component specifications in the model. Very often the TREND statement is used to specify a component that captures the time-varying level of the data. However, in many cases it is also used to define components of a more general nature; for example, it can be used to define a noise component that follows a stationary ARMA model.

You can refer to the state that is associated with a TREND statement by appending the string “_state_” to the end of its name. For example, *name_state_* is the state that is associated with a trend named *name*. You can use *name_state_* in a [COMPONENT](#) statement to define a linear combination of its elements. The estimate

of this linear combination can then be printed or output to a data set. The nominal dimension of `name_state_` is taken to be 1, or the number of variables in the list that is specified in the **CROSS=** option in the **TREND** statement that is used to define `name` (see “[Example 14.4: Smoothing of Repeated Measures for Longitudinal Data](#)” on page 814 for an example of such use of the **COMPONENT** statement).

Some of these trend specifications are applicable to all the data types—that is, they can be used for both regular data types and irregular data types, whereas the others require that the data be regular or regular with replication. Of course, the trend specification is only part of the overall model specification. Therefore, the other parts of the model can imply additional constraints on the data type.

[Table 14.3](#) lists the available trend models and their data requirements. The *type* column shows the admissible keywords that signify the particular trend type. For brevity, the Data Type column groups the data types regular and regular with replication into one category: regular. For more information about these trend models, see the section “[Predefined Trend Models](#)” on page 776.

Table 14.3 Summary of Trend Types

<i>type</i>	Data Type	Description	Parameters
ARIMA(<i>P=integer D=integer ...</i>)	Regular	ARIMA model specification	AR and MA coefficients, and the error variance σ^2
DLL	Regular	Damped local linear	Level and slope σ_1^2, σ_2^2 , damping factor ϕ
LL	Regular	Local linear	Level and slope σ_1^2, σ_2^2
RW	Regular	Random walk	Level σ^2
DECAY	Irregular	A type of decay pattern	Level σ^2 , decay rate ϕ
DECAY(OU)	Irregular	Ornstein-Uhlenbeck decay pattern	Level σ^2 , decay rate ϕ
GROWTH	Irregular	A type of growth pattern	Level σ^2 , growth rate ϕ
GROWTH(OU)	Irregular	Ornstein-Uhlenbeck growth pattern	Level σ^2 , growth rate ϕ
PS(<i>order</i>)	Irregular	Polynomial spline of a given order	Level σ^2

The keyword specification of different trend types, except possibly the ARIMA trend, is quite simple. For example, the following statement specifies `polySpline` as a trend of the type second-order polynomial spline:

```
trend polySpline(ps(2));
```

Similarly, the following statement defines `dampedTrend` as a damped local linear trend:

```
trend dampedTrend(dll) slopevar=x;
```

The variance parameter that governs the slope equation of this trend type is given by a variable `x`, which must be defined elsewhere in the program. The other parameters that define `dampedTrend` are left unspecified (and are estimated by using the data).

The ARIMA trend specification permits specification of trends that follow an $\text{ARIMA}(p,d,q) \times (P,D,Q)_s$ model. The specification of ARIMA models requires some notation, which is explained first.

Let B denote the backshift operator—that is, for any sequence ζ_t , $B\zeta_t = \zeta_{t-1}$. The higher powers of B represent larger shifts (for example, $B^3\zeta_t = \zeta_{t-3}$). A random sequence ζ_t follows an $\text{ARIMA}(p,d,q) \times (P,D,Q)_s$ model with nonseasonal autoregressive order p , seasonal autoregressive order P , nonseasonal differencing order d , seasonal differencing order D , nonseasonal moving average order q , and seasonal moving average

order Q if it satisfies the following difference equation, which is specified in terms of the polynomials in the backshift operator, where a_t is a white noise sequence and s is the season length:

$$\phi(B)\Phi(B^s)(1-B)^d(1-B^s)^D\zeta_t = \theta(B)\Theta(B^s)a_t$$

The polynomials ϕ , Φ , θ , and Θ are of orders p , P , q , and Q , respectively, which can be any nonnegative integers. The season length s must be a positive integer. For example, ζ_t satisfies an ARIMA(1,0,1) model (that is, $p = 1$, $d = 0$, $q = 1$, $P = 0$, $D = 0$, and $Q = 0$) if

$$\zeta_t = \phi_1\zeta_{t-1} + a_t - \theta_1a_{t-1}$$

for some coefficients ϕ_1 and θ_1 and a white noise sequence a_t . Similarly, ζ_t satisfies an ARIMA(0,1,1)×(0,1,1)₁₂ model if

$$\zeta_t = \zeta_{t-1} + \zeta_{t-12} - \zeta_{t-13} + a_t - \theta_1a_{t-1} - \Theta_1a_{t-12} + \theta_1\Theta_1a_{t-13}$$

for some coefficients θ_1 and Θ_1 and a white noise sequence a_t . An ARIMA process is zero-mean, stationary, and invertible if $d = 0$, $D = 0$, and the defining polynomials ϕ , Φ , θ , and Θ have all their roots outside the unit circle—that is, their absolute values are strictly larger than 1.0. It is assumed that the coefficients of the polynomials ϕ , Φ , θ , and Θ are constrained so that the stationarity and invertibility conditions are satisfied. The unknown coefficients of these polynomials become part of the model parameter vector that is estimated by using the data. The general form of the ARIMA trend specification is as follows:

ARIMA(<P=integer> <D=integer> <Q=integer> <SP=integer> <SD=integer> <SQ=integer> <S=integer>)

By default, the different orders are equal to 0 and the season length is equal to 1. The following examples illustrate a few different ARIMA trend specifications. The following statement defines `ima` as an integrated moving average trend:

```
trend ima(arima(d=1 q=1));
```

The following statement defines `airTrend` as a trend that satisfies the well-known airline model (ARIMA(0,1,1)(0,1,1)₁₂ model) for monthly seasonal data:

```
trend airTrend(arima(d=1 q=1 sd=1 sq=1 s=12));
```

The following statement defines `arma11` as a zero-mean ARMA(1,1) trend with autoregressive parameter fixed to 0.1:

```
trend arma11(arima(p=1 q=1)) ar=0.1;
```

For an example of the use of the ARIMA trend specification, see “[Example 14.6: Model with Multiple ARIMA Components](#)” on page 827.

You can use the following *options* in the TREND statement to specify the trend parameters and to request printing of the trend estimates. In addition, you can create a custom combination of a given trend type by specifying the **CROSS=** option to create a more general trend. For an example of using the **CROSS=** option, see the section “[Getting Started: CSSM Procedure](#)” on page 722 and the discussion of the second model in “[Example 14.4: Smoothing of Repeated Measures for Longitudinal Data](#)” on page 814. You can also check for the unexpected changes in the trend component by using the **CHECKBREAK** option.

AR= $\phi_1 \phi_2 \dots \phi_p$

lists the values of the coefficients of the nonseasonal autoregressive polynomial

$$\phi(B) = 1 - \phi_1 B - \dots - \phi_p B^p$$

where the order p is specified in the ARIMA trend specification. The coefficients ϕ_i must define a stationary autoregressive polynomial.

CHECKBREAK<(**ELEMENTWISE** | **OVERALL**)>

turns on the checking of breaks for this trend component. The **ELEMENTWISE** suboption requests the elementwise checking of any unexpected change in the state subsection that is associated with the trend component. The **OVERALL** suboption requests a similar check for the entire state subsection—that is, in this case the change is measured as a multidimensional change. The **ELEMENTWISE** suboption is the default. Unless the **PRINT=BREAKDETAIL** option is specified, only a summary of the most significant breaks is produced. If the **PRINT=BREAKDETAIL** is specified, tables that contain the break significance statistics at every distinct time point are produced—one for the **ELEMENTWISE** suboption and one for the **OVERALL** suboption. If the **CROSS=** option is specified and the **CROSS=** list contains more than one variable, the **OVERALL** suboption considers subsections that are associated with each **CROSS=** variable separately. For more information about the structural break detection process, see the section “[Structural Breaks in the State Evolution](#)” on page 774.

CROSS=(*var1*, *var2*, ...)

CROSS(MATCHPARM)=(*var1*, *var2*, ...)

creates a linear combination of one or more independent trend components that is based on the variables in the list. If the parameters of the trend are specified by options such as the **LEVELVAR=** option or the **PHI=** option, these parameters are shared by these constituent trends. For example, suppose that the **CROSS=** list contains two variables (X_1 and X_2) and the trend specification is of the type **RW**. The effect of **CROSS**=(X_1, X_2) is to create a component $\mu_t = X_1\mu_{1,t} + X_2\mu_{2,t}$, where $\mu_{1,t}$ and $\mu_{2,t}$ are two independent random walk trends. Moreover, if the random walk trend specification uses the **LEVELVAR=** option to specify the variance parameter, $\mu_{1,t}$ and $\mu_{2,t}$ share the same variance parameter; otherwise, two separate variance parameters are assigned to these random walks. If the second form of the **CROSS** option, **CROSS(MATCHPARM)=**, is used, then the constituent trends share all the relevant parameters no matter how they are specified. The **CROSS=** option is useful for a variety of situations. For example, suppose X is an indicator variable that is 1 before a certain time point t_0 and 0 thereafter. Then **CROSS**=(X) has the effect of turning off the trend component after time t_0 . Similarly, suppose G_1 and G_2 are indicators for gender—for example, $G_1 = (\text{GENDER}=1)$ and $G_2 = (\text{GENDER}=0)$ for male and female cases, respectively. Then **CROSS**=(G_1, G_2) results in separate trends according to the gender. The variables in the **CROSS=** list must be free of unknown parameters.

The **CROSS=** option can be computationally expensive; computationally it is equivalent to specifying as many separate trends as the number of variables in the specified list.

LEVELVAR=*variable* | *number*

specifies the disturbance variance parameter for all the trend types. For trend types **LL** and **DLL**, this option specifies σ_1^2 . Any nonnegative value, including 0, is permissible. If *variable* contains unknown parameters, they are estimated from the data. Similarly, if the **LEVELVAR=** option is not specified, σ^2 is estimated from the data.

MA= $\theta_1 \theta_2 \dots \theta_q$

lists the values of the coefficients of the nonseasonal moving average polynomial,

$$\theta(B) = 1 - \theta_1 B - \dots - \theta_q B^q$$

where the order q is specified in the ARIMA trend specification. The coefficients θ_i must define an invertible moving average polynomial.

NODIFFUSE

treats the diffuse elements in the initial state of the state subsection underlying the trend component as nondiffuse. This option is applicable to all trend types except ARIMA. For the ARIMA trend type, this option is ignored even if the nonseasonal or seasonal differencing orders are nonzero. The diffuse elements are assumed to be independent, zero-mean, Gaussian variables. Their variances become part of the parameter vector and are estimated by using the data. This option is useful for creating a trend component that can be interpreted as a deviation from an overall trend component (with diffuse initialization), which is defined separately.

PHI=*variable* | *number*

specifies the value of ϕ for trend types DLL, DECAY, DECAY(OU), GROWTH, and GROWTH(OU). For the type DLL, the specified value must be between 0.0 and 1.0. For types DECAY and DECAY(OU), ϕ must be strictly negative. For types GROWTH and GROWTH(OU), ϕ must be strictly positive. If *variable* contains unknown parameters, they are estimated from the data. Similarly, if the PHI= option is not specified, ϕ is estimated from the data.

PRINT=**BREAKDETAIL** | **COV** | **COV1** | **FILTER** | **SMOOTH** | **T**

PRINT= (< **BREAKDETAIL** > < **COV** > < **COV1** > < **FILTER** > < **SMOOTH** > < **T** >)

requests printing of the respective system matrices of the state equation that underlies the specified trend, the printing of its filtered and smoothed estimates, and the printing of the break statistics at each distinct time point. For the BREAKDETAIL suboption to have any effect, the [CHECKBREAK](#) option must be turned on. If any of these matrices are time-varying, the matrix that corresponds to the first time instance is printed.

SAR= $\Phi_1 \Phi_2 \dots \Phi_P$

lists the values of the coefficients of the seasonal autoregressive polynomial

$$\Phi(B^s) = 1 - \Phi_1 B^s - \dots - \Phi_P B^{sP}$$

where the order P is specified by using the SP= option in the ARIMA trend specification and the season length s is specified in the S= option. The coefficients Φ_i must define a stationary autoregressive polynomial.

SMA= $\Theta_1 \Theta_2 \dots \Theta_Q$

lists the values of the coefficients of the seasonal moving average polynomial

$$\Theta(B^s) = 1 - \Theta_1 B^s - \dots - \Theta_Q B^{sQ}$$

where the order Q is specified by using the SQ= option in the ARIMA trend specification and the season length s is specified in the S= option. The coefficients Θ_i must define an invertible moving average polynomial.

SLOPEVAR=*variable* | *number*

specifies the second disturbance variance parameter, σ_2^2 , for trend types LL and DLL. Any nonnegative value, including 0, is permissible. If *variable* contains unknown parameters, they are estimated from the data. Similarly, if the SLOPEVAR= option is not specified, σ_2^2 is estimated from the data.

Details

Throughout this section, vectors and matrices are denoted by boldface letters. Generally, Greek letters (such as α , β , and ϵ) denote unobserved or latent quantities—often estimated from the data—that represent model parameters, latent states, or noise variables. Capital letters such as X , Y , and Z are used to denote the observed data variables. Whenever there is no ambiguity, it is assumed that the matrices have appropriate dimensions when they are being multiplied—in particular, the vectors behave as column vectors or row vectors as the need arises. On many occasions, matrices are described inline—that is, they are described as parenthesized lists, in a rowwise fashion, with the rows separated by a comma. The term “dot product” is used to describe the scalar that results from the product of a row vector with a (conforming) column vector.

State Space Model and Notation

The (linear) state space model is described in the literature in a few different ways and with varying degree of generality. The description given in this section loosely follows the description given in Durbin and Koopman (2012, chap. 6, sec. 4). This formulation of SSM is quite general; in particular, it includes nonstationary SSMs with time-varying system matrices and state equations with a diffuse initial condition (these terms are defined later in this subsection).

Suppose that observations are collected in a sequential fashion (indexed by a numeric variable τ) on some variables: the vector $\mathbf{y} = (y_1, y_2, \dots, y_q)$, which denotes the q -variate response values, and the k -dimensional vector $\mathbf{x}$, which denotes the predictors. Suppose that the observation instances are $\tau_1 < \tau_2 < \dots < \tau_n$. The possibility that multiple observations are taken at a particular instance τ_i is not ruled out, and the successive observation instances do not need to be regularly spaced—that is, $(\tau_2 - \tau_1)$ does not need to equal $(\tau_3 - \tau_2)$. For $t = 1, 2, \dots, n$, suppose $p_t (\geq 1)$ denotes the number of observations recorded at instance τ_t . For notational simplicity, an integer-valued secondary index t is used to index the data so that $t = 1$ corresponds to $\tau = \tau_1$, $t = 2$ corresponds to $\tau = \tau_2$, and so on. Consider the following model:

$$\begin{aligned} \mathbf{Y}_t &= \mathbf{Z}_t \alpha_t + \mathbf{X}_t \beta + \epsilon_t && \text{Observation equation} \\ \alpha_{t+1} &= \mathbf{T}_t \alpha_t + \mathbf{W}_{t+1} \gamma + \mathbf{c}_{t+1} + \eta_{t+1} && \text{State transition equation} \\ \alpha_1 &= \mathbf{c}_1 + \mathbf{A}_1 \delta + \mathbf{W}_1 \gamma + \eta_1 && \text{Initial condition} \end{aligned}$$

The following list describes these equations:

- The *observation equation* describes the relationship between the $(p_t * q)$ -dimensional response vector $\mathbf{Y}_t$ and the unobserved vectors α_t , β , and ϵ_t . The q -variate responses are vertically stacked in a column to form this $(p_t * q)$ -dimensional response vector $\mathbf{Y}_t$. The m -dimensional vectors α_t are called *states*, the k -dimensional vector β is the regression coefficient vector associated with predictors $\mathbf{x}$, and the $(p_t * q)$ -dimensional vectors ϵ_t are called the *observation disturbances*. The matrices $\mathbf{Z}_t$ (of dimension $(q * p_t) \times m$) and $\mathbf{X}_t$ (of dimension $(q * p_t) \times k$) correspond to the *state effect* and the *regression effect*,

respectively. The elements of $\mathbf{X}_t$ are assumed to be fully known. The states $\boldsymbol{\alpha}_t$ and the disturbances $\boldsymbol{\epsilon}_t$ are *random* sequences. It is assumed that $\boldsymbol{\epsilon}_t$ is a sequence of independent, zero-mean, Gaussian random vectors with diagonal covariances, with the diagonal elements denoted by $\sigma_{t,i}^2, i = 1, 2, \dots, q * p_t$.

- The state sequence $\boldsymbol{\alpha}_t$ is assumed to follow a Markovian structure described by the state transition equation and the associated initial condition.
- The *state transition equation* postulates that a new instance of the state, $\boldsymbol{\alpha}_{t+1}$, is obtained by multiplying its previous instance, $\boldsymbol{\alpha}_t$, by an m -dimensional square matrix $\mathbf{T}_t$ (called the state transition matrix) and by adding three more terms: a known nonrandom vector $\mathbf{c}_{t+1}$ (called the state input); a regression term $\mathbf{W}_{t+1}\boldsymbol{\gamma}$, where $\mathbf{W}_{t+1}$ is an $m \times g$ -dimensional design matrix with fully known elements and $\boldsymbol{\gamma}$ is the g -dimensional regression vector; and a random disturbance vector $\boldsymbol{\eta}_{t+1}$. The m -dimensional state disturbance vectors $\boldsymbol{\eta}_t$ are assumed to be independent, zero-mean, Gaussian random vectors with covariances $\mathbf{Q}_t$ (not necessarily diagonal).
- The *initial condition* describes the starting condition of the state evolution equation. The starting state vector $\boldsymbol{\alpha}_1$ is assumed to be partially diffuse: it is the sum of a known nonrandom vector $\mathbf{c}_1$, a mean-zero Gaussian vector $\boldsymbol{\eta}_1$, and the terms $\mathbf{A}_1\boldsymbol{\delta}$ and $\mathbf{W}_1\boldsymbol{\gamma}$. $\mathbf{A}_1\boldsymbol{\delta}$ represents the contribution from a d -dimensional diffuse vector $\boldsymbol{\delta}$ (a diffuse vector is a Gaussian vector with infinite covariance). The observation and state regression vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are also assumed to be diffuse. The $m \times d$ matrix $\mathbf{A}_1$ is assumed to be completely known.
- The observation disturbances $\boldsymbol{\epsilon}_t$ and the state disturbances $\boldsymbol{\eta}_t$ (for $t \geq 1$) are assumed to be mutually independent. Either the elements of the matrices $\mathbf{Z}_t$, $\mathbf{T}_t$, and $\mathbf{Q}_t$ and the diagonal elements of the observation disturbance covariances $\sigma_{t,i}^2$ are assumed to be completely known, or some of them can be functions of a small set of unknown parameters (to be estimated from the data). Suppose that this unknown set of parameters is denoted by $\boldsymbol{\theta}$.
- The d -dimensional diffuse vector $\boldsymbol{\delta}$ from the state initial condition together with the observation and state regression vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ constitute the overall $(d + k + g)$ -dimensional diffuse initial condition of the model. For more information, see the section “[Filtering, Smoothing, Likelihood, and Structural Break Detection](#)” on page 767.

Although this description of the state space model might appear involved, it conveniently covers many variants of the SSMs that are encountered in practice and precisely describes the most general case that can be handled by the CSSM procedure. An important restriction about the preceding description of the model formulation is that it assumes that the matrices $\mathbf{X}_t$ and $\mathbf{W}_t$ that appear in the observation equation and the state equation respectively are free of unknown parameters and that the covariances of the observation disturbances $\boldsymbol{\epsilon}_t$ are diagonal. In most practical situations, the model under consideration can be easily reformulated to a statistically equivalent form that conforms to this restriction.

NOTE: The transition matrix $\mathbf{T}_t$ in the state equation relates the state $\boldsymbol{\alpha}_t$ at time t with the state $\boldsymbol{\alpha}_{t+1}$ at time $t + 1$. In many situations, such as when the observations are taken at irregular time intervals, $\mathbf{T}_t$ depends on information at both t and $t + 1$. Therefore, it is more appropriate to denote the transition matrix as $\mathbf{T}_t^{t+1}$. However, for simplicity, the former notation is used throughout this chapter. The same comment applies to the covariance matrix $\mathbf{Q}_t$ of the disturbance term $\boldsymbol{\eta}_t$.

For easy reference, [Table 14.4](#) summarizes the information contained in the SSM equations.

Table 14.4 State Space Model: Notation

Notation	Description
$\tau_1, \tau_2, \dots, \tau_n$	Distinct index values at which the observations are recorded
n	Number of distinct index instances
p_t	Number of observations recorded at index τ_t , $t = 1, 2, \dots, n$
q	Number of response variables in the model
$\mathbf{Y}_t = (y_{t,1}, y_{t,2}, \dots, y_{t,p_t * q})$	Vertically stacked vector of response values recorded at τ_t
$N = q * \sum_{t=1}^n p_t$	Total number of response values in the data set
k	Number of predictor (regressor) variables in the observation equation
$\mathbf{X}_t$	$(p_t * q) \times k$ matrix of predictor values recorded at τ_t
$\boldsymbol{\beta}$	k -dimensional regression vector that is associated with the predictors
$\boldsymbol{\epsilon}_t \sim N(0, (\sigma_{t,1}^2, \dots))$	$(q * p_t)$ -dimensional observation disturbance vector with diagonal covariance
m	Dimension of the state vectors $\boldsymbol{\alpha}_t$
$\boldsymbol{\alpha}_t$	m -dimensional state vector
$\mathbf{Z}_t$	$(q * p_t) \times m$ matrix that is associated with $\boldsymbol{\alpha}_t$ in the observation equation
$\mathbf{T}_t$	$m \times m$ state transition matrix
$\mathbf{c}_t$	m -dimensional state input vector
$\mathbf{W}_t$	$m \times g$ design matrix associated with $\boldsymbol{\gamma}$, the state regression vector
$\boldsymbol{\gamma}$	g -dimensional state regression vector
$\boldsymbol{\eta}_t \sim N(0, \mathbf{Q}_t)$	m -dimensional state disturbance vector
d	Dimension of the diffuse vector $\boldsymbol{\delta}$ in the state initial condition
$\boldsymbol{\delta} \sim N(0, \kappa \Sigma), \kappa \rightarrow \infty$	Diffuse vector in the state initial condition
$\mathbf{A}_1$	$m \times d$ constant matrix associated with $\boldsymbol{\delta}$
$\boldsymbol{\eta}_1 \sim N(0, \mathbf{Q}_1)$	m -dimensional state disturbance vector in the initial condition
$\boldsymbol{\theta}$	Parameter vector

Types of Sequence Data

The state space model specification in the CSSM procedure requires proper understanding of both the data organization and the form of the model. The SSMs that are appropriate for time series data might not be appropriate for irregularly spaced longitudinal data. The CSSM procedure distinguishes three types of data organization based on the way the observations are sequenced by the index variable. If an index variable is not specified, it is assumed that the observations are sequenced according to the observation number.

Regular: The observations are recorded at regularly spaced intervals; that is, $\tau_1, \tau_2, \dots, \tau_n$ are regularly spaced. Moreover, at each observation instance τ_i a single observation is recorded; that is, $p_t = 1$ for all t . The standard time series data (both univariate and multivariate) fall in this category.

Regular with Replication: The observations are recorded at regularly spaced intervals, but $p_t > 1$ for at least one t . Here the word replication is used loosely—it does not mean that the multiple observations at τ_t are replications in the precise statistical sense. The panel or cross-sectional data types fall into this category. In the panel data case with p cross sections, $p_t = p$ for all t .

Irregular: The observations are not recorded at regular intervals, and the number of observations p_t at each index instance can be different. The longitudinal data fall into this category.

It is not always easy to decide whether the specified model is appropriate for the given data type. Whenever possible, the CSSM procedure issues a note regarding the possible mismatch between the specified model and the data type being analyzed.

Overview of Model Specification Syntax

An SSM specification involves the description of the terms in the observation equation, the state transition equation, and the initial condition. For example, the response variables, the predictor variables, and the elements of the state transition matrix $\mathbf{T}_t$ must somehow be specified. The CSSM procedure syntax is designed so that little effort is needed to specify the more commonly needed models, while a highly flexible language is available for specifying more complex models. Two syntax features help achieve this goal: the ability to build a complex specification by combining simpler subspecifications, and a programming language for creating lists of variables to be used later in the model specification.

The CSSM procedure statements can be divided into two classes:

- **programming statements**, which are used to create lists of variables that can be used for a variety of purposes (for example, as the elements of the model system matrices)
- statements specific to the CSSM procedure that formulate the state space model and control its other aspects such as the input data specification and the resulting output

Since the matrices involved in the model specification can be specified as lists of variables, which you separately create by using the programming statements, you can finely control all aspects of the model specification. These programming statements permit the use of most DATA step language features such as the conditional logic (IF-THEN-ELSE), array type variables, and all the mathematical functions available in the DATA step. You can also use programming statements to define predictor variables on the fly.

Building a Complex Model Specification

In addition to being able to specify the system matrices in a flexible way, you can also build a complex model specification in a modular way by combining simpler subspecifications. Suppose that the state vector for the model to be specified is composed of subsections that are statistically independent, which is a common scenario in practical modeling situations. For example, suppose that $\boldsymbol{\alpha}_t$ can be divided into two disjoint subsections $\boldsymbol{\alpha}_t^a$ and $\boldsymbol{\alpha}_t^b$, which are statistically independent. This entails a corresponding block-diagonal structure to the system matrices $\mathbf{T}_t$, $\mathbf{W}_t$, and $\mathbf{Q}_t$ that govern the state equations. In this case the term $\mathbf{Z}_t \boldsymbol{\alpha}_t$ that appears in the observation equation also splits into the sum $\mathbf{Z}_t^a \boldsymbol{\alpha}_t^a + \mathbf{Z}_t^b \boldsymbol{\alpha}_t^b$ for appropriately partitioned matrices $\mathbf{Z}_t^a$ and $\mathbf{Z}_t^b$. The model specification syntax of the CSSM procedure makes building an SSM from such smaller pieces easy. Throughout this chapter, the linear combinations of the state subsections (such as $\mathbf{Z}_t^a \boldsymbol{\alpha}_t^a$) that appear in the observation equation are called *components*. An SSM specification in the CSSM procedure is created by combining separate component specifications. In general, you specify a component in two steps: first you define a state subsection $\boldsymbol{\alpha}_t^a$, and then you define a matching linear combination $\mathbf{Z}_t^a \boldsymbol{\alpha}_t^a$. For some special components, such as some commonly needed *trend* components, you can combine these two steps into one keyword specification.

The following list summarizes the (nonprogramming) CSSM procedure syntax statements used for model specification:

- The **ID** statement specifies the index variable (τ). It is assumed that the data within each BY group are ordered (in ascending order) according to the ID variable. The CSSM procedure automatically creates a variable, `_ID_DELTA_`, which contains the difference between the successive ID values. This variable is available for use by the programming statements to define time-varying system matrices. For example, in the case of SSMs used for modeling the longitudinal data, the T_t and Q_t matrices often depend on `_ID_DELTA_` (see “[Example 14.5: A User-Defined Trend Model](#)” on page 823).
- The **PARMS** statement specifies variables that serve as the parameters of the model. That is, it partially defines the model parameter vector θ . Other elements of θ are implicitly defined if your specification of the system matrices is not fully complete.
- The **STATE** statement specifies a subsection of the model state vector. Multiple STATE statements can be used in the model specification; each one defines a statistically independent subsection of the model state vector. For full customization, T_t , W_t , and Q_t blocks that govern this subsection can be specified as lists of variables that are created by programming statements. However, you can obtain many commonly needed state subsection types simply by using the `TYPE=` option in this statement. For example, the use of `TYPE=SEASON(LENGTH=12)` results in a state subsection that can be used to define a monthly seasonal component.
- The **COMPONENT** statement specifies a linear combination that matches a state subsection that is previously defined in a STATE statement. Thus, a matching pair of STATE and COMPONENT statements define a component.
- The **TREND** statement is used for easy specification of some commonly needed components that follow stochastic patterns of certain predefined types.
- The **IRREGULAR** statement specifies the observation disturbance for a particular response variable.
- The **DEPLAG** statement specifies the terms in the model that involve lagged response variables.
- The **MODEL** statement specifies the observation equation for one of the response variables. A separate MODEL statement is needed for each response variable in the multivariate case. The MODEL statement specifies an equation in which the left-hand side is the response variable and the right-hand side is a list that contains components and regression variables.

Model Specification Steps

To illustrate the model specification steps, suppose y is a response variable and variables x_1 and x_2 are predictors. The following statements specify a model for y that includes two components named `cycle` and `randomWalk`, predictors x_1 and x_2 , and an observation disturbance named `whiteNoise`:

```
parms lambda / lower=(1.e-6) upper=(3.14);
parms cycleVar / lower=(1.e-6);
array cycleT{4} c1-c4;
c1 = cos(lambda);
c2 = sin(lambda);
c3 = -c2;
c4 = c1;
state s_cycle(2) T(g)=(cycleT) cov(I)=(cycleVar) a1(2);
component cycle=(1 0)*s_cycle;
trend randomWalk(rw);
```



```

irregular whiteNoise;
model y = x1 x2 randomWalk cycle whiteNoise;

```

The specification begins with a PARMs statement that defines two parameters, `lambda` and `cycleVar`, along with their lower and upper bounds (essentially 0 and π for `lambda`, and 0 and infinity for `cycleVar`). Next, programming statements define an array of variables, `cycleT`, which contains four variables, `c1`–`c4`; these variables are used later for defining the elements of the transition matrix of a state subsection. The STATE statement specifies the two-dimensional subsection `s_cycle`; the dimension appears within the parentheses after the name (`s_cycle(2)`). The T= option specifies the transition matrix for the `s_cycle` ($\mathbf{T}(\mathbf{g}) = (\mathbf{cycleT})$); the $\mathbf{g}$ in $\mathbf{T}(\mathbf{g})$ signifies that the form of the T matrix is *general*. The COV= option (`cov(I) = (cycleVar)`) specifies the covariance of the state disturbance ($\mathbf{Q}_t$ for $t \geq 2$); because of the use of $\mathbf{I}$ in `cov(I)`, the covariance is of the form scaled identity, essentially a two-dimensional diagonal matrix with both diagonal elements equal to `cycleVar`. The initial condition for `s_cycle` is completely diffuse because the A1= option, which specifies $\mathbf{A}_1$, specifies that the dimension of the diffuse vector $\boldsymbol{\delta}$ is 2: `a1(2)`. In this case there is no need to specify the covariance $\mathbf{Q}_1$ in the initial condition. The COMPONENT statement specifies the component `cycle`. It specifies `cycle` as a dot product of two vectors—(1 0) and `s_cycle`, which merely selects the first element of `s_cycle`: `component cycle = (1 0) * s_cycle`. The TREND statement defines a component named `randomWalk`; its type is *rw*, which signifies random walk. The IRREGULAR statement defines an observation disturbance named `whiteNoise`. Both the `randomWalk` and `whiteNoise` specifications are only partially complete—for example, the disturbance variance of `whiteNoise` is not specified. These unspecified variances, `trendVar`, which corresponds to `randomWalk`, and `wnVar`, which corresponds to `whiteNoise`, are automatically included in the list of unknown parameters, $\boldsymbol{\theta}$, along with the parameters that are defined by the PARMs statements. Thus, the parameter vector for this model is $\boldsymbol{\theta} = (\text{lambda } \text{cycleVar } \text{trendVar } \text{wnVar})$. Finally, the model specification is completed by the MODEL statement, which specifies the components of the observation equation: the response variable `y`, the predictors `x1` and `x2`, the components `randomWalk` and `cycle`, and the irregular term `whiteNoise`.

The preceding statements result in an SSM with a three-dimensional state vector, which is the result of combining the two-dimensional state subsection, `s_cycle`, and a one-dimensional subsection underlying the trend, `randomWalk`. In this specification, the initial state is completely diffuse with $\mathbf{Q}_1$ a null matrix, and $\mathbf{A}_1$ equal to the three-dimensional identity. The other state system matrices $\mathbf{T}_t$ and $\mathbf{Q}_t$ are time-invariant:

$$\mathbf{T} = \begin{bmatrix} \cos(\text{lambda}) & \sin(\text{lambda}) & 0 \\ -\sin(\text{lambda}) & \cos(\text{lambda}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} \text{cycleVar} & 0 & 0 \\ 0 & \text{cycleVar} & 0 \\ 0 & 0 & \text{trendVar} \end{bmatrix}$$

The observation equation is obvious with $\mathbf{Z} = [1 \ 0 \ 1]$.

Sparse Transition Matrix Specification

It often happens that the transition matrix $\mathbf{T}$ (or $\mathbf{T}_t$ in the time-varying case) specified in a STATE statement is sparse—that is, many of its elements are zero. The algorithms in the CSSM procedure exploit this sparsity for computational efficiency. In most cases the sparsity of a T-block can be inferred from the context. However, if the elements of the T-block are supplied by a list of variables in parentheses, it can be difficult to recognize elements that are structurally zero (this is because of the generality of the DATA step language used for defining the variables). To simplify the specification of such sparse transition matrix, CSSM procedure has adopted a convention: the variables that correspond to structural zeros can (and should) be left *unset*—that is, these variables are declared, but no value is assigned to them. As an example, suppose that a three-dimensional state subsection has the following form of transition matrix for some variables `X1`, `X2`, and `X3`

defined elsewhere in the program:

$$T = \begin{bmatrix} X1 & 0 & 0 \\ X2 & X1 & 0 \\ X3 & 0 & X1 \end{bmatrix}$$

The following (incomplete) statements show how to specify such a T-block:

```
array tMat{3,3};
do i=1 to 3;
    tMat[i, i] = x1;
end;
tMat[2,1] = x2;
tMat[3,1] = x3;
state foo(3) T(g)=(tMat) ...;
```

In this specification only the nonzero elements of the tMat array, which contains $3 \times 3 = 9$ elements, are assigned a value. On the other hand, the following statements show an alternate way of specifying the same T-block. This specification explicitly sets the zeros in the T-block (the elements above the diagonal and tMat[3,2]) to 0.

```
array tMat{3,3};
do i=1 to 3;
    do j=1 to 3;
        if i=j then tMat[i, j] = x1;
        else if j > i then tMat[i, j] = 0;
    end;
end;
tMat[2,1] = x2;
tMat[3,1] = x3;
tMat[3,2] = 0;
state foo(3) T(g)=(tMat) ...;
```

The first specification is simpler, and is preferred. The second specification is mathematically equivalent (and generates the same output) but is computationally less efficient since its sparsity structure cannot always be reliably inferred due to the generality of the DATA step language. In the first specification, the unset elements are recognized to be *structural* zeros while the set elements are treated as nonzero for sparsity purposes. For a simple illustration, see “[Example 14.5: A User-Defined Trend Model](#)” on page 823. Proper sparsity specification can lead to significant computational savings for larger matrices.

Regression Variable Specification in Multivariate Models

Suppose that a regression variable in a multivariate model affects two or more response variables. For example, suppose that response variables y1 and y2 depend on a regression variable x. This dependence can be categorized as one of two types:

- In the more common case, the regression coefficient of x for y1 and the regression coefficient of x for y2 are different. The relationship can be described as follows:

$$\begin{aligned} y1 &= \beta_1 x + \text{other terms} \\ y2 &= \beta_2 x + \text{other terms} \end{aligned}$$

In the CSSM procedure you can specify this type of relationship in two equivalent ways:

- You can specify the variable x in the MODEL statement for y_1 and specify the variable x_copy (a copy of x) in the MODEL statement for y_2 as follows:

```
x_copy = x;           /* create a copy of x */
model y1 = x ...;
model y2 = x_copy ...;
```

- You can specify the variable x in MODEL statements for both y_1 and y_2 as follows:

```
model y1 = x ...;
model y2 = x ...;
```

This specification avoids creating x_copy .

Of these two alternate ways, the first is preferred because x and x_copy can then be unambiguously used in an EVAL statement to refer to the terms $\beta_1 x$ and $\beta_2 x$, respectively.

- In the less common case, y_1 and y_2 share a common regression coefficient. The relationship can be described as follows:

$$\begin{aligned} y_1 &= \beta x + \text{other terms} \\ y_2 &= \beta x + \text{other terms} \end{aligned}$$

You can specify this type of relationship by placing the regression coefficient in the model state vector as follows:

```
state beta(1) T(I) A1(1) ;      /* beta is a constant state */
comp xeffect = beta*(x) ;
model y1 = xeffect ...;
model y2 = xeffect ...;
```

Here the STATE statement defines β as a one-dimensional, time-invariant constant (because the transition matrix is identity, the disturbance covariance is 0 and the initial state is diffuse). Next, the COMP statement defines $xeffect$ as the product between β and the variable x . Subsequently, both y_1 and y_2 use $xeffect$ in their respective MODEL statements.

Filtering, Smoothing, Likelihood, and Structural Break Detection

The Kalman filter and smoother (KFS) algorithm is the main computational tool for using SSM for data analysis. This subsection briefly describes the basic quantities generated by this algorithm and their relationship to the output generated by the CSSM procedure. For proper treatment of SSMs with a diffuse initial condition or when regression variables are present, a modified version of the traditional KFS, called diffuse Kalman filter and smoother (DKFS), is needed. A good discussion of the different variants of the traditional and diffuse KFS can be found in Durbin and Koopman (2012). The DKFS implemented in the CSSM procedure closely follows the treatments in De Jong and Chu-Chun-Lin (2003) and Francke, Koopman, and de Vos (2010). Additional details can be found in these references.

The state space model equations (see the section “[State Space Model and Notation](#)” on page 760) imply that the combined response data vector $\mathbf{Y} = (\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_n)$ has a Gaussian probability distribution. This

probability distribution is *proper* if d , the dimension of the diffuse vector δ in the initial condition, is 0 and if $(k + g)$, the total number of regression variables in the observation and state equations, is also 0 (the regression vectors β and γ are also treated as a diffuse vectors). Otherwise, this probability distribution is *improper*. The KFS algorithm is a combination of two iterative phases: a forward pass through the data, called *filtering*, and a backward pass through the data, called *smoothing*, that uses the quantities generated during filtering. One of the advantages of using the SSM formulation to analyze the time series data is its ability to handle the missing values in the response variables. The KFS algorithm appropriately handles the missing values in $\mathbf{Y}$. For additional information about how PROC CSSM handles missing values, see the section “Missing Values” on page 793.

Filtering Pass

The filtering pass sequentially computes the quantities shown in Table 14.5 for $t = 1, 2, \dots, n$ and $i = 1, 2, \dots, q * p_t$.

Table 14.5 KFS: Filtering Phase

Quantity	Description
$\hat{y}_{t,i} = E(y_{t,i} y_{t,i-1}, \dots, y_{t,1}, \mathbf{Y}_{t-1}, \dots, \mathbf{Y}_1)$	One-step-ahead prediction of the response values
$v_{t,i} = y_{t,i} - \hat{y}_{t,i}$	One-step-ahead prediction residuals
$F_{t,i} = \text{Var}(y_{t,i} y_{t,i-1}, \dots, y_{t,1}, \mathbf{Y}_{t-1}, \dots, \mathbf{Y}_1)$	Variance of the one-step-ahead prediction
$\hat{\alpha}_{t,i} = E(\alpha_t y_{t,i-1}, \dots, y_{t,1}, \mathbf{Y}_{t-1}, \dots, \mathbf{Y}_1)$	One-step-ahead prediction of the state vector
$\mathbf{P}_{t,i} = \text{Cov}(\alpha_t y_{t,i-1}, \dots, y_{t,1}, \mathbf{Y}_{t-1}, \dots, \mathbf{Y}_1)$	Covariance of $\hat{\alpha}_{t,i}$
$\mathbf{b}_{t,i}$	$(d + k + g)$ -dimensional vector
$\mathbf{S}_{t,i}$	$(d + k + g)$ -dimensional symmetric matrix
$\begin{pmatrix} \hat{\delta} & \hat{\beta} & \hat{\gamma} \end{pmatrix}_{t,i}' = \mathbf{S}_{t,i}^{-1} \mathbf{b}_{t,i}$	Estimates of δ , β , and γ by using the data up to (t, i)
$\mathbf{S}_{t,i}^{-1}$	Covariance of $\begin{pmatrix} \hat{\delta} & \hat{\beta} & \hat{\gamma} \end{pmatrix}_{t,i}'$
$\mathbf{S}_{t,i}^*$	$(d + k + g)$ -dimensional symmetric matrix needed in the marginal likelihood computation

Here the notation $E(y_{t,i} | y_{t,i-1}, \dots, y_{t,1}, \mathbf{Y}_{t-1}, \dots, \mathbf{Y}_1)$ denotes the *conditional expectation* of $y_{t,i}$ given the history up to the index $(t, i - 1)$: $(y_{t,i-1}, \dots, y_{t,1}, \mathbf{Y}_{t-1}, \dots, \mathbf{Y}_1)$. Similarly $\text{Var}(y_{t,i} | y_{t,i-1}, \dots, y_{t,1}, \mathbf{Y}_{t-1}, \dots, \mathbf{Y}_1)$ denotes the corresponding conditional variance. The quantity $v_{t,i} = y_{t,i} - \hat{y}_{t,i}$ is set to missing whenever $y_{t,i}$ is missing. Note that $\hat{y}_{t,i}$ are *one-step-ahead* forecasts only when the model has only one response variable and the data are a time series; in all other cases it is more appropriate to call them *one-measurement-ahead* forecasts (since the next measurement might be at the same time point). Despite this, $\hat{y}_{t,i}$ are called one-step-ahead predictions (and $v_{t,i}$ are called one-step-ahead residuals) throughout this document. In the diffuse case, the conditional expectations must be appropriately interpreted. The vector $\mathbf{b}_{t,i}$ and the matrix $\mathbf{S}_{t,i}$ contain some accumulated quantities that are needed for the estimation of δ , β , and γ . Of course, when $(d + k + g) = 0$ (the nondiffuse case), these quantities are not needed. In the diffuse case, because the matrix $\mathbf{S}_{t,i}$ is sequentially accumulated (starting at $t = 1, i = 1$), it might not be invertible until some $t = t_*, i = i_*$. The filtering process is called *initialized* after $t = t_*, i = i_*$. In some situations, this initialization might not happen even after the entire sample is processed—that is, the filtering process remains *uninitialized*. This can happen if the regression variables are collinear or if the data are not sufficient to estimate the initial condition δ for some other reason. In

the diffuse case if the marginal likelihood is to be computed (see the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769), an additional matrix ($\mathbf{S}_{t,i}^*$) is computed at each step by sequential accumulation.

The filtering process is used for a variety of purposes. One important use of filtering is to compute the likelihood of the data. In the model-fitting phase, the unknown model parameters θ are estimated by maximum likelihood. This requires repeated evaluation of the likelihood at different trial values of θ . After θ is estimated, it is treated as a known vector. The filtering process is used again with the fitted model in the forecasting phase, when the one-step-ahead forecasts and residuals based on the fitted model are provided. In addition, this filtering output is needed by the smoothing phase to produce the full-sample component estimates and for the structural break analysis.

Likelihood Computation and Model-Fitting Phase

Because of the Gaussian nature of the response vector, the likelihood of $\mathbf{Y}$ can be computed by using the prediction-error decomposition. The desired prediction-error decomposition is obtained by the filtering pass described in the previous section (“[Filtering Pass](#)” on page 768). When the state space model under consideration has a nondiffuse initial condition and no regression effects are present in the observation and state equations, $\mathbf{Y}$ has a proper Gaussian distribution and its likelihood is defined unambiguously. Otherwise, the definition of the likelihood depends on the treatment of the diffuse quantities— δ , β , and γ . Francke, Koopman, and de Vos (2010) describe three variants of the likelihood function—*diffuse*-likelihood, *marginal*-likelihood, and *profile*-likelihood—that are commonly considered for state space models that have a diffuse initial condition. In the CSSM procedure, you can either use diffuse likelihood ($L_d(\mathbf{Y}, \theta)$) or marginal likelihood ($L_m(\mathbf{Y}, \theta)$) for parameter estimation. By default, diffuse likelihood is used for parameter estimation.

In terms of the quantities described in [Table 14.5](#), diffuse, marginal, and profile likelihoods are defined as follows:

$$\begin{aligned} -2 \log L_d(\mathbf{Y}, \theta) &= N_0 \log 2\pi + \sum_{t=1}^n \sum_{i=1}^{q \cdot p_t} \left(\log F_{t,i} + \frac{v_{t,i}^2}{F_{t,i}} \right) - \mathbf{b}_{n,p_n}' \mathbf{S}_{n,p_n}^{-1} \mathbf{b}_{n,p_n} + \log(|\mathbf{S}_{n,p_n}|) \\ -2 \log L_m(\mathbf{Y}, \theta) &= N_0 \log 2\pi + \sum_{t=1}^n \sum_{i=1}^{q \cdot p_t} \left(\log F_{t,i} + \frac{v_{t,i}^2}{F_{t,i}} \right) - \mathbf{b}_{n,p_n}' \mathbf{S}_{n,p_n}^{-1} \mathbf{b}_{n,p_n} + \log(|\mathbf{S}_{n,p_n}|) \\ &\quad - \log(|\mathbf{S}_{n,p_n}^*|) \\ -2 \log L_p(\mathbf{Y}, \theta) &= N \log 2\pi + \sum_{t=1}^n \sum_{i=1}^{q \cdot p_t} \left(\log F_{t,i} + \frac{v_{t,i}^2}{F_{t,i}} \right) - \mathbf{b}_{n,p_n}' \mathbf{S}_{n,p_n}^{-1} \mathbf{b}_{n,p_n} \end{aligned}$$

In the preceding formulas, the terms that are associated with the missing response values $y_{t,i}$ are excluded and N denotes the total number of nonmissing response values in the sample. In addition, $N_0 = (N - k - g - d)$; $|\mathbf{S}_{n,p_n}|$ and $|\mathbf{S}_{n,p_n}^*|$ denote the determinants of $\mathbf{S}_{n,p_n}$ and $\mathbf{S}_{n,p_n}^*$, respectively; and $\mathbf{b}_{n,p_n}'$ denotes the transpose of the column vector $\mathbf{b}_{n,p_n}$. If $\mathbf{S}_{n,p_n}$ is not invertible, then a generalized inverse is used in place of $\mathbf{S}_{n,p_n}^{-1}$, and $|\mathbf{S}_{n,p_n}|$ and $|\mathbf{S}_{n,p_n}^*|$ are computed based on the nonzero eigenvalues of $\mathbf{S}_{n,p_n}$ and $\mathbf{S}_{n,p_n}^*$, respectively. Moreover, in this case, $N_0 = N - \text{Rank}(\mathbf{S}_{n,p_n})$. When $(d + k + g) = 0$, the terms that involve $\mathbf{S}_{n,p_n}$, $\mathbf{S}_{n,p_n}^*$, and $\mathbf{b}_{n,p_n}$ are absent.

The expression for marginal likelihood is derived by treating the diffuse quantities as fixed but unknown parameters. The expression can be shown to be based on a linear transformation of the N -dimensional response vector $\mathbf{Y}$. For a suitably chosen $N \times N$ matrix $\mathbf{H}$, let $\mathbf{U} = \mathbf{H}\mathbf{Y}$. $\mathbf{H}$ is chosen such that the

N -dimensional transformed vector $\mathbf{U}$ partitions into two uncorrelated (and independent because of their Gaussian nature) subvectors: an N_0 -dimensional vector $\mathbf{U}_1$ and a $(d + k + g)$ -dimensional vector $\mathbf{U}_2$. Furthermore, the distribution of $\mathbf{U}_1$ does not depend on the diffuse vectors $(\boldsymbol{\delta}, \boldsymbol{\beta}, \text{ and } \boldsymbol{\gamma})$, and $\mathbf{U}_2$ stores the generalized least squares estimates of the diffuse vectors: $\mathbf{U}_2' = (\hat{\boldsymbol{\delta}} \ \hat{\boldsymbol{\beta}} \ \hat{\boldsymbol{\gamma}})'$. It turns out that the N_0 -dimensional vector, $\mathbf{U}_1$, has a proper Gaussian distribution and the marginal likelihood, $\log \mathbf{L}_m(\mathbf{Y}, \boldsymbol{\theta})$, is the proper likelihood of $\mathbf{U}_1$. The diffuse likelihood, $\log \mathbf{L}_d(\mathbf{Y}, \boldsymbol{\theta})$, is also based on $\mathbf{U}_1$. However, rather than assuming the diffuse quantities as unknown parameters, the expression of the diffuse likelihood is derived by assuming that $\boldsymbol{\delta}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ are random vectors with diffuse priors. Even though the marginal and diffuse likelihoods are based on different interpretations of the diffuse quantities, their expressions differ by only one term: the diffuse likelihood does not have the term $-\log(|\mathbf{S}_{n,p_n}^*|)$. Since the marginal likelihood is the proper likelihood of $\mathbf{U}_1$, the diffuse likelihood can be interpreted as a quasi-likelihood of $\mathbf{U}_1$. Apart from being essential to make it a proper likelihood, the extra term in the marginal likelihood plays another useful role: it makes the marginal likelihood invariant to linear rescaling of the diffuse effects, a desirable property in a likelihood. The diffuse likelihood is not invariant to linear rescaling of the diffuse effects. The profile likelihood, $\log \mathbf{L}_p(\mathbf{Y}, \boldsymbol{\theta})$, is the likelihood of the response vector $\mathbf{Y}$ evaluated at the generalized least squares estimates of the diffuse vectors: $(\boldsymbol{\delta} \ \boldsymbol{\beta} \ \boldsymbol{\gamma})' = (\hat{\boldsymbol{\delta}} \ \hat{\boldsymbol{\beta}} \ \hat{\boldsymbol{\gamma}})'$. It is derived by treating the diffuse quantities as fixed but unknown parameters and, like the marginal likelihood, is invariant to linear rescaling of the diffuse effects. For an illustration of this invariance property, see “[Example 14.18: Invariance of the Marginal Likelihood under Linear Rescaling of the Diffuse Effects](#)” on page 891.

In the literature, the marginal likelihood (in addition to the diffuse likelihood) is also called the *restricted*-likelihood, and the estimate of the parameter vector $\boldsymbol{\theta}$ that is obtained by maximizing $\log \mathbf{L}_m(\mathbf{Y}, \boldsymbol{\theta})$ (or $\log \mathbf{L}_d(\mathbf{Y}, \boldsymbol{\theta})$) is called the restricted maximum likelihood estimate (REML). In this section, the REML estimate that is based on marginal likelihood is denoted by REML_M and the REML estimate that is based on diffuse likelihood is denoted by REML_D. In addition, the estimate of $\boldsymbol{\theta}$ that is obtained by maximizing the profile likelihood is called the maximum likelihood estimate (ML). In the absence of the diffuse quantities, all three likelihoods are the same and the REML_M, REML_D, and ML estimates coincide. When diffuse quantities are present, there is some evidence to prefer REML_M, the estimate of $\boldsymbol{\theta}$ that is based on the marginal likelihood. For more information, see Francke, Koopman, and de Vos (2010) and the references therein. By default, the CSSM procedure uses diffuse likelihood for parameter estimation. You can switch to marginal likelihood by using the `LIKE=MARGINAL` option in the PROC CSSM statement. In the current release of the CSSM procedure, you cannot request parameter estimation by using profile likelihood.

Interestingly, for many types of state space models, REML_M and REML_D coincide even when diffuse effects are present. This is because for these models the extra term in the marginal likelihood, $-\log(|\mathbf{S}_{n,p_n}^*|)$, turns out to be independent of the parameter vector $\boldsymbol{\theta}$. Specifically, for models that are specified by using the PROC CSSM syntax, REML_M and REML_D differ only if at least one of the following conditions hold:

- The transition matrix ($\mathbf{T}_t$) that is implied by a STATE statement depends on at least one unknown parameter and the diffuse dimension of the associated state subsection is nonzero.
- The list of variables that is specified in a COMPONENT statement depends on at least one unknown parameter and the diffuse dimension of the associated state subsection is nonzero.
- At least one lag term in a DEPLAG statement depends on an unknown parameter.
- A TREND statement with GROWTH or GROWTH(OU) type is present and the growth parameter, ϕ , is unknown.

In particular, if the parameter vector affects only the disturbance covariances (Q_t) in the state equation and the error variances ($\sigma_{t,i}^2$) in the observation equation (see [Table 14.4](#)), REML_M and REML_D coincide. These observations also imply that REML_M and REML_D coincide for the most commonly used univariate and multivariate unobserved component models and for ARIMA models, with or without regression effects.

NOTE: For many examples in the section “[Examples: CSSM Procedure](#)” on page 803, one of the preceding conditions does hold and the REML_M and REML_D estimates do differ. However, in all these cases, it turns out that the differences in REML_M and REML_D are not large enough to change the overall conclusions of the analysis. As verification, you can rerun the analyses that are described in “[Example 14.10: A Transfer Function Model for the Gas Furnace Data](#)” on page 853, “[Example 14.13: Bivariate Model for Sales of Mink and Muskrat Furs](#)” on page 871, and “[Example 14.14: A Factor Model for Now-Casting the US Economy](#)” on page 876 by using the `LIKE=MARGINAL` option in the PROC CSSM statement. Of course, this will not be true in general.

The REML_D and REML_M estimates of the unknown parameter vector θ (each denoted as $\hat{\theta}$), are computed by maximizing the diffuse (or marginal) likelihood. This is done by using a nonlinear optimization process that involves repeated evaluations of $L_d(Y, \theta)$ (or $L_m(Y, \theta)$) at different values of θ . Approximate standard errors of $\hat{\theta}$ are computed by taking the square root of the diagonal elements of its (approximate) covariance matrix. This covariance is computed as $-\mathbf{H}^{-1}$, where $\mathbf{H}$ is the Hessian (the matrix of the second-order partials) of $\log L_d(Y, \theta)$ (or $\log L_m(Y, \theta)$) evaluated at the optimum $\hat{\theta}$. It is known that under mild regularity assumptions (as the number of distinct time points tends toward infinity), $\hat{\theta}$ is consistent and efficient. For good discussions about REML_D, REML_M, and ML estimates, see Searle, Casella, and McCulloch (1992); Laird (2004); Francke, Koopman, and de Vos (2010).

If the marginal likelihood is used for parameter estimation, the CSSM procedure reports the values of all three likelihoods at the parameter estimate $\hat{\theta}$. Otherwise, PROC CSSM reports the values of the diffuse and profile likelihoods that are calculated at the parameter estimate $\hat{\theta}$. Let $\dim(\theta)$ denote the dimension of the parameter vector θ . After PROC CSSM completes the parameter estimation, it prints the “Likelihood Computation Summary” table, which summarizes the likelihood calculations at $\hat{\theta}$, as shown in [Table 14.6](#).

Table 14.6 Likelihood Computation Summary

Quantity	Formula
Nonmissing response values used	N
Estimated parameters	$\dim(\theta)$
Initialized diffuse state elements	$\text{rank}(\mathbf{S}_{n,p_n})$
Normalized residual sum of squares	$\sum_{t=1}^n \sum_{i=1}^{q \cdot p_t} \left(\frac{v_{t,i}^2}{F_{t,i}} \right) - \mathbf{b}'_{n,p_n} \mathbf{S}_{n,p_n}^{-1} \mathbf{b}_{n,p_n}$
Diffuse log likelihood	$\log L_d(Y, \hat{\theta})$
Marginal log likelihood	$\log L_m(Y, \hat{\theta})$
Profile log likelihood	$\log L_p(Y, \hat{\theta})$

In addition to the likelihood computation summary, PROC CSSM also reports the information criteria that are based on the diffuse and profile likelihoods. It also reports the information criteria that are based on the marginal likelihood if marginal likelihood is used for parameter estimation. A variety of information criteria are reported. All these criteria are functions of twice the negative likelihood ($-2 \log \mathbf{L}$, where the likelihood can be diffuse, marginal, or profile), N_* (the effective sample size), and $nparm$ (the effective number of model parameters). For information criteria that are based on the diffuse and marginal likelihoods,

the effective sample size, N_* , is equal to N_0 and the effective number of model parameters, $nparm$, is equal to $\dim(\theta)$. For information criteria that are based on the profile likelihood, the effective sample size, N_* , is equal to N and the effective number of model parameters, $nparm$, is equal to $\dim(\theta) + d + k + g$. Table 14.7 summarizes the reported information criteria in smaller-is-better form.

Table 14.7 Information Criteria

Criterion	Formula	Reference
AIC	$-2 \log \mathbf{L} + 2nparm$	Akaike (1974)
AICC	$-2 \log \mathbf{L} + 2nparm N_*/(N_* - nparm - 1)$	Hurvich and Tsai (1989) Burnham and Anderson (1998)
HQIC	$-2 \log \mathbf{L} + 2nparm \log \log(N_*)$	Hannan and Quinn (1979)
BIC	$-2 \log \mathbf{L} + nparm \log(N_*)$	Schwarz (1978)
CAIC	$-2 \log \mathbf{L} + nparm(\log(N_*) + 1)$	Bozdogan (1987)

Forecasting Phase

After the model-fitting phase, the filtering process is repeated again to produce the model-based one-step-ahead response variable forecasts ($\hat{y}_{t,i}$), residuals ($v_{t,i}$), and their standard errors ($\sqrt{F_{t,i}}$). In addition, one-step-ahead forecasts of the components that are specified in the MODEL statements, and any other user-defined linear combinations of α_t , are also produced. These forecasts are set to missing as long as the index $t < t_*$ (that is, until the filtering process is initialized). If the filtering process remains uninitialized, then all the quantities that are related to the one-step-ahead forecast (such as $\hat{y}_{t,i}$ and $v_{t,i}$) are reported as missing. When the fitted model is appropriate, the one-step-ahead residuals $v_{t,i}$ form a sequence of uncorrelated normal variates. This fact can be used during model diagnostic process.

Smoothing Phase

After the filtering phase of KFS produces the one-step-ahead predictions of the response variables and the underlying state vectors, the smoothing phase of KFS produces the full-sample versions of these quantities—that is, rather than using the history up to $(t, i - 1)$, the entire sample $\mathbf{Y}$ is used. The smoothing phase of KFS is a backward algorithm, which begins at $t = n$ and $i = q * p_n$ and goes back toward $t = 1$ and $i = 1$. It produces the following quantities:

Table 14.8 KFS: Smoothing Phase

Quantity	Description
$\tilde{y}_{t,i} = E(y_{t,i} \mathbf{Y})$	Interpolated response value
$\tilde{F}_{t,i} = \text{Var}(y_{t,i} \mathbf{Y})$	Variance of the interpolated response value
$\tilde{\alpha}_t = E(\alpha_t \mathbf{Y})$	Full-sample estimate of the state vector
$\tilde{\mathbf{P}}_t = \text{Cov}(\alpha_t \mathbf{Y})$	Covariance of $\tilde{\alpha}_t$
$\begin{pmatrix} \tilde{\delta} & \tilde{\beta} & \tilde{\gamma} \end{pmatrix}' = \mathbf{S}_{n,p_n}^{-1} \mathbf{b}_{n,p_n}$	Full-sample estimates of δ , β , and γ
$\mathbf{S}_{n,p_n}^{-1}$	Covariance of $\begin{pmatrix} \tilde{\delta} & \tilde{\beta} & \tilde{\gamma} \end{pmatrix}'$

Note that if $y_{t,i}$ is not missing, then $\tilde{y}_{t,i} = E(y_{t,i}|\mathbf{Y}) = y_{t,i}$ and $\tilde{F}_{t,i} = \text{Var}(y_{t,i}|\mathbf{Y}) = 0$ because $y_{t,i}$ is completely known, given $\mathbf{Y}$. Therefore, $\tilde{y}_{t,i}$ provides nontrivial information only when $y_{t,i}$ is missing—in which case $\tilde{y}_{t,i}$ represents the best estimate of $y_{t,i}$ based on the available data. The full-sample estimates of components that are specified in the model equations are based on the corresponding linear combinations of $\tilde{\alpha}_t$. Similarly, their standard errors are computed by using appropriate functions of $\tilde{\mathbf{P}}_t$.

If the filtering process remains uninitialized until the end of the sample (that is, if $\mathbf{S}_{n,p_n}$ is not invertible), some linear combinations of δ , β , and γ are not estimable. This, in turn, implies that some linear combinations of α_t are also inestimable. These inestimable quantities are reported as missing. For more information about the estimability of the state effects, see Selukar (2010).

Delete-One Cross Validation and Structural Breaks

In addition to the interpolation of missing response values and the full-sample estimation of components in the model, the smoothing phase can also produce several useful diagnostic measures that can indicate outlying observations and breaks in the state evolution process. The treatment of additive outliers and structural breaks that is described in this section is based on De Jong and Penzer (1998). Also see Selukar (2017) for illustrative examples.

Delete-One Cross Validation and the Additive Outlier Detection

Let $\text{AO}_{t,i} = y_{t,i} - E(y_{t,i}|\mathbf{Y}^{t,i})$ denote the difference between the observed response value $y_{t,i}$ and its estimate or prediction by using all the data except $y_{t,i}$, which is denoted by $\mathbf{Y}^{t,i}$. The smoothing phase of DKFS can generate $\text{AO}_{t,i}$ (and its variance) at all (t, i) . A large value of $\text{AO}_{t,i}$ signifies that the observed response value ($y_{t,i}$) is unusual relative to the rest of the sample (according to the postulated model). Such values are called additive outliers. In the literature, $\text{AO}_{t,i}$ are referred by a few different names. Sometimes they are called *delete-one cross validation errors* or simply *prediction errors*. In this chapter, these names are used interchangeably. Like the one-step-ahead residuals, $v_{t,i}$, the prediction errors can be used in checking the adequacy of the model. The prediction errors are normally distributed; however, unlike $v_{t,i}$, they are not serially uncorrelated. $\text{AO}_{t,i}$ is set to missing when $y_{t,i}$ is missing. The CSSM procedure prints a summary table of extreme additive outliers by default. In addition, you can request the plotting of the standardized prediction errors, and they can be output to a data set.

The prediction error sum of squares (PRESS)

$$\sum_{t,i} \text{AO}_{t,i}^2$$

can be a useful measure of fit to compare different models. It is also called the *cross validation error sum of squares*. An additional measure of fit based on the prediction errors is called the *generalized cross validation error sum of squares* (GCV). Denoting the variance of $\text{AO}_{t,i}$ by $\text{VAR_AO}_{t,i}$, it is defined as

$$\frac{\sum_{t,i} (\text{AO}_{t,i}^2 / \text{VAR_AO}_{t,i})}{[\sum_{t,i} (1 / \text{VAR_AO}_{t,i})]^2}$$

You can request the printing of PRESS and GCV by specifying the PRESS option in the OUTPUT statement.

After inspecting the reported additive outliers, you can adjust the model to account for the effects of some of the extreme outlying observations. This can be done by including appropriate dummy variables in the observation equation.

Structural Breaks in the State Evolution

The additive outliers that are discussed in the preceding section are diagnostic measures associated with the measurement equation. The smoothing phase of DKFS can generate diagnostic measures that are also associated with the state equation.

For simplicity of notation and exposition, initially assume that the state equation has the following form:

$$\alpha_{t+1} = \mathbf{T}_t \alpha_t + \mathbf{c}_{t+1} + \eta_{t+1}$$

That is, the state regression term $\mathbf{W}_{t+1}\gamma$ is absent in the postulated model. Suppose that an unanticipated change of unknown size takes place in the i_0 th element of the state at time $(t_0 + 1)$. The model can then be adjusted to account for this change by including a suitable dummy regressor in the state equation as follows:

$$\alpha_{t+1} = \mathbf{T}_t \alpha_t + \mathbf{W}_{t+1}\gamma + \mathbf{c}_{t+1} + \eta_{t+1}$$

Here $\mathbf{W}_t$ is a sequence of m -dimensional column vectors such that $\mathbf{W}_{t_0+1}[i_0] = 1$ and $\mathbf{W}_t[i] = 0$ for all other t and i . The estimate of the regression coefficient γ provides information about the size of the unanticipated change in the i_0 th element of α_t at time $t = t_0 + 1$. Similarly, an unanticipated change in a subsection of α_t at a time $t = t_0 + 1$ can be estimated by using a set of appropriate dummies (the number of dummies equals the number of elements in the state subsection) in the state equation. The algorithm of De Jong and Penzer (1998) efficiently generates the estimates of such one-time changes in the state at all distinct time points in the sample in one smoothing pass. A statistically significant value of γ at a time point t_0 indicates an unanticipated change in the relevant element (or the subsection) of α_{t_0} . Note that the change associated with an additive outlier is temporary: the previous or the subsequent measurements are not affected. On the other hand, because of the evolutionary nature of the state equation, a one-time change in the state affects all the subsequent states, which in turn affect the subsequent observations. In this sense, a significant unanticipated change in the state is a *structural break*.

In the preceding discussion, the absence of the state regression variables in the postulated model was assumed only for notational simplicity. If the postulated model does contain some state regression variables, the dummy variable that is associated with the one-time state change is simply added to the existing set of state regression variables, and the interpretation of its regression coefficient as the measure of unanticipated change in the state remains unaffected.

In the CSSM procedure, you can request the computation of significance statistics that are associated with one-time changes in the state subsections specified by using the STATE statement in addition to the state subsections that are associated with the components specified by using the TREND statements. This is done by using the CHECKBREAK option in these statements. In addition, you can request the computation of such statistics for the entire state α_t by using the MAXSHOCK option in the OUTPUT statement. The significance statistics can be computed for both elementwise change and subsectionwise change. The computation of subsectionwise change statistics can be computationally expensive for large subsections (an inversion of a $p \times p$ -dimensional matrix at each distinct time point in the sample is needed for the computation of significance statistics for a state subsection of size p). For an example of structural break analysis, see “[Example 14.8: Diagnostic Plots and Structural Break Analysis](#)” on page 841.

Estimation of User-Specified Linear Combination of State Elements

By default, the CSSM procedure computes the estimates of all the components that are specified in the MODEL statements (you can print these estimates by using the PRINT= option in the respective TREND and COMPONENT statements, or you can output these estimates to a data set by specifying it in the OUT= option in the OUTPUT statement). However, in many cases it is desirable to obtain the estimates of additional linear combinations of the state elements and the regression effects. The CSSM procedure provides two statements, the COMPONENT statement and the EVAL statement, that are useful for specifying virtually any desired linear combination of the elements of the state vector and the regression effects in the observation equation. After a desired linear combination is specified, you can print or output its estimate as you would for a component that is used in the MODEL statement. This feature of the CSSM procedure is illustrated in many examples in the section “Examples: CSSM Procedure” on page 803. For example, in the second part of “Example 14.4: Smoothing of Repeated Measures for Longitudinal Data” on page 814, the COMPONENT and EVAL statements are used to define the contrasts between the growth profiles of cows that are receiving different treatments. Similarly, in “Example 14.7: A Dynamic Factor Model for the Yield Curve” on page 831, the EVAL statement is used to define the yield curve as a sum of the components that are used in the MODEL statement.

Contrasting PROC CSSM with Other SAS Procedures

The CSSM procedure complements several SAS/ETS procedures and the MIXED procedure in SAS/STAT software (see Chapter 84, “The MIXED Procedure” (*SAS/STAT User’s Guide*)). The statistical models underlying all these procedures can be formulated as state space models; however, in many cases this formulation effort can be considerable. Generally speaking, when a problem can be formulated and satisfactorily solved either by using the CSSM procedure or by using one of these other procedures, the other procedures are likely to be more efficient. However, in many instances, the CSSM procedure can solve more general problems or offer more detailed analysis, or both. Throughout this discussion, it is assumed that the problem being solved can be modeled as a linear statistical model with Gaussian response variables. In particular, situations that require models such as autoregressive conditional heteroscedasticity (ARCH) models, and models with categorical response variables are not considered. The following list provides a more specific comparison of the CSSM procedure with different procedures:

- All the SAS/ETS time series analysis procedures (the ARIMA, ESM, UCM, VARMAX, STATESPACE, and PANEL procedures) require time series data and are not applicable to the longitudinal data.
- For univariate time series analysis, the modeling facilities provided by the ARIMA, ESM, and UCM procedures are adequate in most cases. The CSSM procedure can handle cases that do not fit neatly into one of these categories.
- For multivariate time series data analysis, you can use the VARMAX procedure for vector ARIMA modeling and the STATESPACE procedure for state space modeling. The capabilities of the CSSM procedure are complementary to these procedures. In particular, the predefined multivariate structural models available in the CSSM procedure cannot be specified by either of these procedures. In addition, you can formulate a much wider range of multivariate models—for example, models for series with different frequencies, by using the CSSM procedure.

- When the $\mathbf{R}$ side effects are not too complicated (for example, if $\mathbf{R}$ is diagonal), the model considered by the MIXED procedure is a special case of the model considered by the CSSM procedure. In the case of diagonal $\mathbf{R}$, it is easy to see that the state vector $\boldsymbol{\alpha}_t$ is equal to $\boldsymbol{\gamma}$, the MIXED random-effects vector, for all $t \geq 1$ (that is, $\boldsymbol{\alpha}_t$ is *time invariant*). Therefore, the random-effects MIXED model is obtained by setting $\mathbf{T} = \text{Identity}$, $\mathbf{Q}_t = \mathbf{0}$, $t \geq 2$, $\mathbf{Q}_1 = \mathbf{G}$ (the MIXED $\mathbf{G}$ matrix), and $\mathbf{A}_1 = \mathbf{0}$.
- For the analysis of cross-sectional data, you can use the PANEL procedure. In this case, the CSSM procedure capabilities are complementary. PROC CSSM can provide alternate models, REML estimates, richer missing value support, and the estimates of the unobserved components (see the section “Getting Started: CSSM Procedure” on page 722 and the examples “Example 14.2: Random-Effects and Autoregressive Models for Panel Data” on page 809 and “Example 14.11: Dynamic Panel Model for the Cigar Data” on page 859 for more information). In some situations the cross-sectional studies contain many panels but very few distinct time points. The PANEL procedure based analysis is better suited in such settings. In order for the analysis based on PROC CSSM to be valid, the cross-sectional study must contain an adequate number of distinct time points.

Predefined Trend Models

The statistical models that govern the predefined trend components available in the CSSM procedure are divided into two groups: models that are applicable to equally spaced data (possibly with replication), and models that are applicable more generally (the irregular data type). Each trend component can be described as a dot product $\mathbf{Z}\boldsymbol{\alpha}_t$ for some (time-invariant) vector $\mathbf{Z}$ and a state vector $\boldsymbol{\alpha}_t$. The component specification is complete after the vector $\mathbf{Z}$ is specified and the system matrices that govern the equations of $\boldsymbol{\alpha}_t$ are specified. For trend models for regular data, all the system matrices are time-invariant. For irregular data, $\mathbf{T}_t$ and $\mathbf{Q}_t$ depend on the spacing between the distinct time points: $(\tau_{t+1} - \tau_t)$.

Trend Models for Regular Data

These models are applicable when the data type is either regular or regular with replication. A good reference for these models is Harvey (1989).

Random Walk Trend

This model provides a trend pattern in which the level of the curve changes with time. The rapidity of this change is inversely proportional to the disturbance variance σ^2 that governs the underlying state. It can be described as $\mathbf{Z}\boldsymbol{\alpha}_t$, where $\mathbf{Z} = (1)$ and the (one-dimensional) state α_t follows a random walk:

$$\alpha_{t+1} = \alpha_t + \eta_{t+1}, \quad \eta_t \sim N(0, \sigma^2)$$

Here $\mathbf{T} = 1$ and $\mathbf{Q} = \sigma^2$. The initial condition is fully diffuse. Note that if $\sigma^2 = 0$, the resulting trend is a fixed constant.

Local Linear Trend

This model provides a trend pattern in which both the level and the slope of the curve change with time. This variation in the level and the slope is controlled by two parameters: σ_1^2 controls the level variation, and σ_2^2 controls the slope variation. If $\sigma_1^2 = 0$, the resulting trend is called an *integrated random walk*. If both $\sigma_1^2 = 0$ and $\sigma_2^2 = 0$, then the resulting model is the deterministic linear time trend. Here $\mathbf{Z} = (1 \ 0)$, $\mathbf{T} = (1 \ 1, \ 0 \ 1)$, and $\mathbf{Q} = \text{Diag}(\sigma_1^2, \sigma_2^2)$. The initial condition is fully diffuse.

Damped Local Linear Trend

This trend pattern is similar to the local linear trend pattern. However, in the DLL trend the slope follows a first-order autoregressive model, whereas in the LL trend the slope follows a random walk. The autoregressive parameter or the damping factor, ϕ , must lie between 0.0 and 1.0, which implies that the long-run forecast according to this pattern has a slope that tends to 0. Here $\mathbf{Z} = (1 \ 0)$, $\mathbf{T} = (1 \ 1, \ 0 \ \phi)$, and $\mathbf{Q} = \text{Diag}(\sigma_1^2, \sigma_2^2)$. The initial condition is partially diffuse with $\mathbf{Q}_1 = \text{Diag}(0, \sigma_2^2/(1 - \phi * \phi))$.

ARIMA Trend

This section describes the state space form for a component that follows an $\text{ARIMA}(p,d,q) \times (P,D,Q)_s$ model. The notation for ARIMA models is explained in the **TREND** statement.

First the state space form for the stationary case—that is, when $d = 0$ and $D = 0$, is explained. A number of alternate state space forms are possible in this case; the one described here is based on Jones (1980). With slight abuse of notation, let $p = p + s * P$ denote the effective autoregressive order, and let $q = q + sQ$ denote the effective moving average order of the model. Similarly, let ϕ be the effective autoregressive polynomial, and let θ be the effective moving average polynomial in the backshift operator with coefficients $\phi_1, \dots, \phi_p$ and $\theta_1, \dots, \theta_q$, obtained by multiplying the respective nonseasonal and seasonal factors. Then, a random sequence ξ_t that follows an $\text{ARMA}(p,q) \times (P,Q)_s$ model with a white noise sequence a_t has a state space form with state vector of size $m = \max(p, q + 1)$. The system matrices are as follows: $\mathbf{Z} = [1 \ 0 \ \dots \ 0]$, and the transition matrix $\mathbf{T}$, in a blocked form, is given by

$$\mathbf{T} = \begin{bmatrix} 0 & I_{m-1} \\ \phi_m \ \dots & \phi_1 \end{bmatrix}$$

where $\phi_i = 0$ if $i > p$ and I_{m-1} is an $(m - 1)$ dimensional identity matrix. The covariance of the state disturbance matrix $\mathbf{Q} = \sigma^2 \psi \psi'$, where σ^2 is the variance of the white noise sequence a_t and the vector $\psi = [\psi_0 \ \dots \ \psi_{m-1}]'$ contains the first m values of the impulse response function—that is, the first m coefficients in the expansion of the ratio θ/ϕ . The covariance matrix of the initial state, $\mathbf{Q}_1$, is computed as

$$\text{vec}(\mathbf{Q}_1) = (\mathbf{I} - \mathbf{T} \otimes \mathbf{T})^{-1} \text{vec}(\mathbf{Q})$$

where $\otimes$ denotes the Kronecker product and the *vec* operation on a matrix creates a vector formed by vertically stacking the rows of that matrix.

A number of alternate state space forms are possible in the nonstationary case also. The form used by the CSSM procedure utilizes the state space form for the stationary case as a building block. Suppose that a random sequence ξ_t follows an $\text{ARIMA}(p,d,q) \times (P,D,Q)_s$ model with a white noise sequence a_t . As in the notation for the stationary case, with slight abuse of notation, let $d = d + s * D$ denote the effective differencing order, and let Δ be the effective differencing polynomial in the backshift operator with coefficients $\Delta_1, \dots, \Delta_d$. It can be shown that ξ_t has a state space form with state vector size $m^\dagger = m + d$. In what follows, the system matrices and related quantities in the nonstationary case are described in terms of similar entities in the stationary case. A superscript dagger ($\dagger$) has been added to distinguish the entities from the nonstationary case. $\mathbf{Z}^\dagger = [0 \ 0 \ \dots \ 1 \ \dots \ 0]$ where the only nonzero value, 1, is at the index $m + 1$, and the transition matrix, $\mathbf{T}^\dagger$, in a blocked form, is given by

$$\mathbf{T}^\dagger = \begin{bmatrix} \mathbf{T} & 0 & 0 \\ \mathbf{Z}\mathbf{T} & \Delta_1 \ \dots & \Delta_d \\ 0 & I_{d-1} & 0 \end{bmatrix}$$

The state disturbance matrix $\mathbf{Q}^\dagger$ is given by

$$\mathbf{Q}^\dagger = \begin{bmatrix} \mathbf{Q} & \mathbf{QZ}' & 0 \\ \mathbf{ZQ} & \mathbf{ZQZ}' & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Finally, the initial state is partially diffuse: the first m elements are nondiffuse and the last d elements are diffuse. The covariance matrix of the first m elements is $\mathbf{Q}_1$.

Trend Models for Irregular Data

A good reference for these models is De Jong and Mazzi (2001). Throughout this section $h_t = (\tau_{t+1} - \tau_t)$ denotes the difference between the successive time points. The system matrices $\mathbf{T}_t$ and $\mathbf{Q}_t$ that govern these models depend on h_t . However, whenever the notation is unambiguous, the subscript t is omitted.

Polynomial Spline Trend

This model is a general-purpose tool for extracting a smooth trend from the noisy data. The order of the spline governs the order of the local polynomial that defines the spline. The order-1 spline corresponds to Brownian motion (continuous-time random walk), the order-2 spline corresponds to integrated Brownian motion (continuous-time integrated random walk), and the order-3 spline provides a locally quadratic trend; the default order is 1. The dimension of the state underlying this component is the same as the order of the spline. The system matrices for the orders up to 3 are described as follows (in all the cases the initial condition is fully diffuse):

- order-1 spline: $\mathbf{Z} = (1)$, $\mathbf{T} = (1)$, and $\mathbf{Q} = \sigma^2(h)$
- order-2 spline: $\mathbf{Z} = (1 \ 0)$, $\mathbf{T} = (1 \ h, \ 0 \ 1)$, and $\mathbf{Q} = \sigma^2 \left(\frac{h^3}{3} \ \frac{h^2}{2}, \ \frac{h^2}{2} \ h \right)$
- order-3 spline: $\mathbf{Z} = (1 \ 0 \ 0)$, $\mathbf{T} = \left(1 \ h \ \frac{h^2}{2}, 0 \ 1 \ h, 0 \ 0 \ 1 \right)$, and

$$\mathbf{Q}[i, j] = \sigma^2 * \frac{h^{6-i-j+1}}{(6-i-j+1)(3-i)!(3-j)!} \quad 1 \leq i, j \leq 3$$

The system matrices for higher orders are similarly defined (for more information, see De Jong and Mazzi (2001)).

Note that, in addition to providing an estimate of the trend, this methodology can provide estimates of the higher-order derivatives of the trend. If $\boldsymbol{\alpha}$ denotes the k -dimensional subsection that is associated with a polynomial spline of order k , then its j th element ($1 \leq j \leq k$), $\boldsymbol{\alpha}[j]$, corresponds to the derivative of order $(j-1)$ of this polynomial spline. For an example of the estimation of the first derivative of a trend component, see “[Example 14.12: Long-Term Temperature Trends Based on a Multivariate Model](#)” on page 864. For additional information about using these types of trend patterns in data analysis, see Eubank, Huang, and Wang (2003); Kohn, Ansley, and Tharm (1991); Selukar (2015).

Decay and Growth Trends

There are two choices for the decay trend: DECAY and DECAY(OU). Similarly, there are two choices for the growth trend: GROWTH and GROWTH(OU). The “OU” stands for the Ornstein-Uhlenbeck form of these models. The decay trend is a sum of two correlated components: one component is a random walk, and the

other component is a stationary autoregression. In its Ornstein-Uhlenbeck form, the random walk component is replaced by a constant. The growth trend (and its Ornstein-Uhlenbeck variant) has the same form as the decay trend except that the autoregression is nonstationary (in fact, it is explosive). For growth trend models, floating-point errors can result for even moderately long forecast horizons because of the explosive growth in the trend values.

The system matrices for the decay and the growth types in their respective cases are identical, except for the sign of the rate parameter ϕ : $\phi < 0.0$ for the decay type, and $\phi > 0.0$ for the growth type. In addition, the initial conditions for the growth models are fully diffuse; they are only partially diffuse for the decay models. The underlying state vector for all these models is two-dimensional.

The system matrices for the DECAF type are

$$\begin{aligned} \mathbf{Z} &= (1 \ 1) \\ \mathbf{T} &= \text{Diag}(1, \exp(h\phi)) \\ \mathbf{Q} &= \frac{\sigma^2}{\phi^3} (h\phi \ 1 - \exp(h\phi), \ 1 - \exp(h\phi) \ (\exp(2h\phi) - 1)/2) \end{aligned}$$

The initial condition is partially diffuse with $\mathbf{Q}_1 = \text{Diag}(0, \frac{-\sigma^2}{2\phi^3})$. The system matrices for the GROWTH type are the same (with $\phi > 0.0$), except that the initial condition is fully diffuse; so $\mathbf{Q}_1 = 0$.

For the DECAF(OU) type, $\mathbf{Z}$ and $\mathbf{T}$ are the same as DECAF, whereas

$$\mathbf{Q} = \text{Diag} \left(0, \sigma^2 \frac{(\exp(2h\phi) - 1)}{2\phi} \right) \text{ and } \mathbf{Q}_1 = \text{Diag}(0, \frac{-\sigma^2}{2\phi})$$

The system matrices for the GROWTH(OU) type are the same (with $\phi > 0.0$), except that the initial condition is fully diffuse; so $\mathbf{Q}_1 = 0$.

Predefined Structural Models

A set of predefined models is available in the CSSM procedure for models called structural models in the time series literature. These predefined models can be used to model trend, seasonal, and cyclical patterns in the univariate and multivariate time series. For the most part, the multivariate models are straightforward generalizations of the corresponding univariate models—for example, the multivariate random walk trend described later in this section generalizes the univariate random walk trend that is described in the section “Random Walk Trend” on page 776. All of these models, with the exception of the continuous-time cycle model, are applicable only to the regular data type. The continuous-time cycle model is applicable to all the data types; however, it is available for the univariate case only.

To specify these models, you must first use the STATE statement with the correct TYPE= option. When you specify the TYPE=option, you do not need to specify other options of the STATE statement (for example, the T option, the COV1 option, and the A1 option). However, you must specify the COV option, which describes the covariance of the disturbance term that drives the state equation. Throughout this section, the symmetric matrix specified by using the COV option is denoted by Σ . For TYPE= LL, an additional matrix, specified by using the SLOPECOV suboption, also plays a role; it is denoted by Σ_{slope} . Subsequently you must specify one or more COMPONENT statements to define the (univariate) components that are based on this state subsection for their inclusion in the MODEL statement. These univariate components exhibit

interesting behavior based on the structure of Σ (and Σ_{slope} , whenever applicable)—for example, imposing rank restrictions on Σ in the multivariate random walk results in these univariate trends moving together. For additional information about these models, see Harvey (1989).

The following example summarizes the steps needed to define a multivariate structural model by using a sequence of STATE and COMPONENT statements. For a full example, see “[Example 14.1: Bivariate Basic Structural Model](#)” on page 803. Suppose that a three-dimensional time series is being studied with response variables y_1 , y_2 , and y_3 . Suppose you want to specify the trivariate structural model

$$y_t = \mu_t + \psi_t + \epsilon_t$$

where $y_t = (y_{1,t}, y_{2,t}, y_{3,t})$ denotes the response series, and μ_t , ψ_t , and ϵ_t denote the trivariate components, trend, cycle, and white noise, respectively. The three components of ϵ_t , the observation noise in the model, are not assumed to be independent. Therefore, you cannot specify them by using three IRREGULAR statements; you must include them in the state specification. The following (incomplete) statements show how to specify this model:

```
state whiteNoise(3)  type=wn ...;
component wn1 = whiteNoise[1];
component wn2 = whiteNoise[2];
component wn3 = whiteNoise[3];

state randomWalk(3)  type=rw ...;
component rw1 = randomWalk[1];
component rw2 = randomWalk[2];
component rw3 = randomWalk[3];

state cycleState(3)  type=cycle ...;
component c1 = cycleState[1];
component c2 = cycleState[2];
component c3 = cycleState[3];

model y1 = rw1 c1 wn1;
model y2 = rw2 c2 wn2;
model y3 = rw3 c3 wn3;
```

The first STATE statement defines `whiteNoise`, a state subsection that is needed for defining a three-dimensional white noise component. In turn, `whiteNoise` is used to define the three univariate white noise components: `wn1`, `wn2`, and `wn3`. The components `wn1`, `wn2`, and `wn3` are correlated—their correlation structure is controlled by the covariance specification of `whiteNoise`. The second set of STATE and COMPONENT statements result in three correlated random walk trend components: `rw1`, `rw2`, and `rw3`. Finally, the last set of STATE and COMPONENT statements result in three correlated cycle components: `c1`, `c2`, and `c3`. In the end, the desired multivariate model is defined by including these (univariate) components in the appropriate MODEL statements.

In the preceding example, it is important to note the relationship between the nominal dimension (denoted by *dim* throughout this section) that is specified in the STATE statement and the actual dimension of the resulting state subsection. Note that the three state subsections, `whiteNoise`, `randomWalk`, and `cycleState`, are defined by using the same *dim* specification: 3. However, the actual dimensions of these state subsections depend on their type; they do not need to equal this specified dimension. Here, `whiteNoise` and `randomWalk` do have the same size, 3, as the specified *dim*. However, the size of `cycleState`, which is of TYPE=CYCLE, is $2 * \text{dim} = 6$. Another important point to note: no matter what the underlying size of the state subsection, the

desired univariate components were obtained by using an identical specification scheme in the COMPONENT statement. This happens because the component specification style that is based on the element operator—[]—in the COMPONENT statement behaves differently when the TYPE= option is used to define the state subsection (for an illustration, see the section “Multivariate Season” on page 783).

The system matrices for all these models are time-invariant, with the exception of the continuous-time cycle model. In this section, α_t denotes the subsection of the overall model state α_t , and T , Q , and A_1 denote the corresponding blocks of the larger system matrices.

For the multivariate cycle system matrices described in the section “Multivariate Cycle” on page 782, the Kronecker product notation is useful: if A is an $m \times n$ matrix and B is a $p \times q$ matrix, then the Kronecker product $A \otimes B$ is an $mp \times nq$ block matrix:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

Multivariate White Noise

The STATE statement option **TYPE=WN** specifies white noise of dimension *dim*—that is, a sequence of zero mean, independent, Gaussian vectors with covariance Σ . The specification of the associated system matrices is trivial: T is zero, $Q = \Sigma$, and the initial condition is nondiffuse ($Q_1 = \Sigma$ and $A_1 = 0$).

Multivariate white noise is needed to specify the observation equation noise term for the multivariate models for the time series data. Since the state space formulation for the CSSM procedure requires the observation equation noise vector to have the diagonal form, you need to include the noise vector in the state. The noise term for the *i*th response variable is defined by a component that simply picks the *i*th element of this multivariate white noise. For example, the component `wn_i` defined as follows can be used as a noise term in the MODEL statement of the *i*th response variable:

```
state white(dim) type=wn ...;
component wn_3 = white[3];
```

Multivariate Random Walk Trend

The STATE statement option **TYPE=RW** specifies a *dim*-dimensional random walk

$$\alpha_{t+1} = \alpha_t + \eta_{t+1}$$

where η_t is a sequence of zero mean, independent, Gaussian vectors with covariance Σ . The specification of the associated system matrices is trivial: T is a *dim*-dimensional identity matrix, I_{dim} , $Q = \Sigma$, and the initial condition is fully diffuse ($Q_1 = 0$ and $A_1 = I_{dim}$).

The multivariate random walk is a useful trend model for multivariate time series data. The trend term for the *i*th response variable is defined by a component that simply picks the *i*th ($1 \leq i \leq dim$) element of α_t . For example, the component `rw_i` defined as follows can be used as a trend term in the MODEL statement of the *i*th response variable:

```
state randomWalk(3) type=rw ...;
component rw_2 = randomWalk[2];
```

Multivariate Local Linear Trend

The STATE statement option **TYPE=LL** specifies a $(2*dim)$ -dimensional α_t , needed for defining a dim -dimensional local linear trend. The first dim elements of α_t correspond to the needed multivariate trend, and the subsequent dim elements are needed to capture the slope vector of this trend. α_t can be defined as

$$\alpha_{t+1} = T\alpha_t + \eta_{t+1}$$

where η_t is a sequence of zero mean, independent, Gaussian vectors with covariance $\text{Diag}(\Sigma, \Sigma_{\text{slope}})$ and T is a $2*dim$ -dimensional block matrix $T = (\mathbf{I}_{dim} \ \mathbf{I}_{dim}, \ \mathbf{0} \ \mathbf{I}_{dim})$. The initial condition is fully diffuse ($\mathbf{Q}_1 = 0$ and $\mathbf{A}_1 = \mathbf{I}_{2*dim}$). This is a multivariate generalization of the univariate local linear trend.

The multivariate local linear trend is a useful trend model for multivariate time series data. The trend term for the i th response variable is defined by a component that simply picks the i th element ($1 \leq i \leq dim$) of α_t . For example, the component `ll_i` defined as follows can be used as a trend term in the MODEL statement of the i th response variable:

```
state localLin(dim) type=ll(slopecov...) ...;
component ll_3 = localLin[3];
```

Multivariate Cycle

The STATE statement option **TYPE=CYCLE** specifies a $(2*dim)$ -dimensional α_t , needed for defining a dim -dimensional cycle. As in the LL case, the first dim elements of α_t correspond to the needed dim -dimensional cycle, and the remaining dim elements contain some auxiliary quantities. The cycle model defined in this subsection requires a regular data type—that is, the CT option is not included. Let ρ denote the damping factor, and let $\lambda = 2\pi/\text{period}$ be the frequency associated with the cycle. The admissible parameter ranges are $0 < \rho \leq 1$ and $\text{period} > 2$, which implies that $0 < \lambda < \pi$. Let $\mathbf{C} = \rho(\cos(\lambda) \ \sin(\lambda), \ -\sin(\lambda) \ \cos(\lambda))$, a 2×2 matrix, and let $\mathbf{T} = \mathbf{C} \otimes \mathbf{I}_{dim}$, a $2 * dim \times 2 * dim$ matrix. With this notation, the transition equation associated with α_t is

$$\alpha_{t+1} = T\alpha_t + \eta_{t+1}$$

where η_t is a sequence of zero mean, independent, $(2 * dim)$ -dimensional Gaussian vectors with covariance $\text{Diag}(\Sigma, \Sigma)$. If $\rho = 1$, the initial condition is fully diffuse ($\mathbf{Q}_1 = 0$ and $\mathbf{A}_1 = \mathbf{I}_{2*dim}$). Otherwise, it is nondiffuse: $\mathbf{Q}_1 = \frac{1}{(1-\rho^2)}\text{Diag}(\Sigma, \Sigma)$ and $\mathbf{A}_1 = 0$.

The multivariate cycle is useful for capturing periodic behavior for multivariate time series data. The cycle term for the i th response variable is defined by a component that simply picks the i th element of α_t . For example, the component `cycle_i` defined as follows can be used as a cycle term in the MODEL statement of the i th response variable:

```
state cycleState(dim) type=cycle ...;
component cycle_2 = cycleState[2];
```


Multivariate Season

The STATE statement option **TYPE=SEASON(LENGTH=s)** specifies a $((s-1)*dim)$ -dimensional α_t , needed for defining a dim -dimensional trigonometric season component with season length s . A (multivariate) trigonometric season component, ζ , is a sum of (multivariate) cycles of different frequencies,

$$\zeta = \sum_{j=1}^{[s/2]} \zeta_j$$

where the constituent cycles ζ_j , called harmonics, have frequencies $\lambda_j = 2\pi j/s$. All the harmonics are assumed to be statistically independent, have the same damping factor $\rho = 1$, and are governed by the disturbances with the same covariance matrix Σ . The number of harmonics, $[s/2]$, equals $s/2$ if s is even and $(s-1)/2$ if it is odd. This means that specifying **TYPE=SEASON(LENGTH=s)** is equivalent to specifying $[s/2]$ cycle specifications with correct frequencies, damping factor $\rho = 1$, and the **COV** option restricted to the same covariance Σ . The resulting α_t is necessarily $((s-1)*dim)$ -dimensional. When the season length s is even, the last harmonic cycle, $\zeta_{s/2}$, has frequency π and requires special attention. It is of dimension dim rather than $2*dim$ because its underlying state equation simplifies to a dim -variate autoregression with autoregression coefficient $-\mathbf{I}_{dim}$. As a result of this discussion, it is clear that the system matrices **T** and **Q** associated with the $((s-1)*dim)$ -dimensional α_t are block-diagonal with the blocks corresponding to the harmonics. The initial condition is fully diffuse.

For all the models discussed so far, the first dim elements of α_t provided the needed (multivariate) component. This is not the case for the (multivariate) season component. Extracting the i th seasonal component from α_t requires accumulating the contributions from the $[s/2]$ harmonics that are associated with this i th seasonal, which are not organized contiguously in α_t . For example, suppose that dim is 2 and the season length s is 4. In this case $[s/2]$ is 2, and the bivariate seasonal component is a sum of two independent bivariate cycles, ζ_1 and ζ_2 . The cycle ζ_1 has frequency $\pi/2$ and its underlying state, say α_t^a , has dimension $2 * dim = 4$. The last harmonic, ζ_2 , has frequency π , and therefore its underlying state, say α_t^b , has dimension 2. The combined state $\alpha_t = (\alpha_t^a, \alpha_t^b)$ has dimension $6 = 4 + 2$. In order to extract the first bivariate seasonal component, you must extract the first components of bivariate cycles ζ_1 and ζ_2 , which in turn implies the first elements of α_t^a and α_t^b , respectively. Thus, obtaining the first bivariate seasonal component requires extracting the first and the fifth elements of the combined state α_t . Similarly, obtaining the second bivariate seasonal component requires extracting the second and the sixth elements of the combined state α_t . All this can be summarized by the dot product expressions

$$\begin{aligned} s_{1t} &= (1 \ 0 \ 0 \ 0 \ 1 \ 0) \alpha_t \\ s_{2t} &= (0 \ 1 \ 0 \ 0 \ 0 \ 1) \alpha_t \end{aligned}$$

where s_{1t} and s_{2t} denote the first and second components, respectively, of the bivariate seasonal component. Note that s_{1t} and s_{2t} are univariate seasonal components, each of season length 4, in their own right. They are correlated components; their correlation structure depends on Σ .

Obtaining the desired components of the multivariate seasonal component is made easy by a special syntax convention of the **COMPONENT** statement. Continuing with the previous example, the following examples illustrate two equivalent ways of obtaining s_{1t} and s_{2t} . The first set of statements explicitly specify the linear combinations needed for defining s_{1t} and s_{2t} :

```
state seasonState(2) type=season(length=4) ...;
component s_1 =( 1 0 0 0 1 0 ) * seasonState;
component s_2 =( 0 1 0 0 0 1 ) * seasonState;
```

The following simpler specification achieves the same result:

```
state seasonState(2) type=season(length=4) ...;
component s_1 = seasonState[1];
component s_2 = seasonState[2];
```

In the latter specification, the meaning of the element operator [] changes if the state in question is defined by using the **TYPE=** option.

Multivariate ARMA

You can specify a state vector that follows a multivariate autoregressive, moving average (VARMA) model by using the STATE statement option **TYPE=VARMA**. The autoregressive and moving average orders can be either 0 or 1 ($0 \leq p \leq 1$ and $0 \leq q \leq 1$)—that is, only VAR(1), MA(1), and VARMA(1,1) models can be specified. The notation and the state space form of the VARMA model described here is taken from Reinsel (1997), which is a good reference for VARMA modeling.

A dim -dimensional vector process ζ_t follows a zero-mean, autoregressive order p , moving average order q (VARMA(p, q)) model if it satisfies the following matrix difference equation:

$$\zeta_t = \sum_{i=1}^p \Phi_i \zeta_{t-i} + \epsilon_t - \sum_{j=1}^q \Theta_j \epsilon_{t-j}$$

Here Φ_i and Θ_j are dim -dimensional square matrices and ϵ_t is a dim -dimensional, Gaussian, white noise sequence with covariance matrix Σ . If autoregressive order p is 0, the term that involves Φ_i is absent; similarly, if the moving average order q is 0, the term that involves Θ_j is absent. Since AR and MA orders can be at most 1, the subscripts of Φ_i and Θ_j can be ignored in this discussion—when applicable, an AR coefficient matrix is denoted by Φ and an MA coefficient matrix is denoted by Θ . The unknown elements of Φ , Θ , and Σ constitute the parameter vector that is associated with a VARMA state. The process ζ_t defined by the VARMA difference equation is stationary and invertible (Reinsel 1997) if and only if the eigenvalues of Φ and Θ are strictly less than 1 in magnitude. By default, the CSSM procedure imposes these stationarity and invertibility restrictions on Φ and Θ . However, you can specify Φ to be an identity matrix, in which case the resulting process is nonstationary.

A VARMA model can be cast into a state space form. The state space form used by the CSSM procedure is described in Reinsel (1997, pp. 52–53). The system matrices for the supported VARMA models are as follows:

- The VAR(1) form is the simplest. In this case, the underlying state α_t is the same as the VAR(1) process ζ_t . Therefore, $T = \Phi$ and $Q_t = \Sigma$.
- Taking Φ equal to the zero matrix if $p = 0$, the VARMA(1,1) and MA(1) cases can be treated together. In this case, the underlying state α_t is $2*dim$ dimensional and the desired VARMA process ζ_t corresponds to its first dim elements. Let $\Psi = \Phi - \Theta$. Then, in the blocked form,

$$T = \begin{bmatrix} 0 & I_{dim} \\ 0 & \Phi \end{bmatrix} \quad \text{and} \quad Q_t = Q = \begin{bmatrix} \Sigma & \Sigma\Psi' \\ \Psi\Sigma & \Psi\Sigma\Psi' \end{bmatrix}$$

Unless Φ is restricted to be identity, the underlying state α_t is stationary and the covariance of the initial condition is computed by

$$\text{vec}(\mathbf{Q}_1) = (\mathbf{I} - \mathbf{T} \otimes \mathbf{T})^{-1} \text{vec}(\mathbf{Q})$$

where $\otimes$ denotes the Kronecker product and the *vec* operation on a matrix creates a vector formed by vertically stacking the rows of that matrix. If Φ is restricted to be identity, the initial condition is fully diffuse.

Continuous-Time Cycle

The STATE statement option **TYPE=CYCLE(CT)** specifies a two-dimensional α_t , needed for defining a univariate continuous time cycle. In this case the nominal dimension, *dim*, must be 1. In particular, Σ becomes one-dimensional, which is denoted by σ^2 . This cycle can be used for any data type. As before, the parameters of the cycle are a damping factor ρ , $0 < \rho \leq 1$, and *period* > 0 . Unlike in the discrete-time cycle described in the section “**Multivariate Cycle**” on page 782, the *period* is not required to be larger than 2. Let $\lambda = 2\pi/\text{period}$, and let $h_t = (\tau_{t+1} - \tau_t)$ denote the difference between successive time points. In this case, the system matrices $\mathbf{T}$ and $\mathbf{Q}$ that govern α_t depend on h_t . They are as follows:

$$\begin{aligned} \mathbf{T} &= \rho^{h_t} \begin{pmatrix} \cos(\lambda h_t) & \sin(\lambda h_t) \\ -\sin(\lambda h_t) & \cos(\lambda h_t) \end{pmatrix} \\ \mathbf{Q} &= \frac{\sigma^2(1 - \rho^{2h_t})}{-2 \ln(\rho)} * \mathbf{I}_2 \quad \text{if } \rho < 1 \\ \mathbf{Q} &= \sigma^2 h_t \mathbf{I}_2 \quad \text{if } \rho = 1 \end{aligned}$$

If $\rho < 1$, the initial condition is nondiffuse: $\mathbf{Q}_1 = \frac{\sigma^2}{-2 \ln(\rho)} \mathbf{I}_2$. For $\rho = 1$, the initial condition is fully diffuse.

The first element of α_t corresponds to the needed cycle, and the second element is an auxiliary quantity. You can define a cycle term based on this state as follows:

```
state cycleState(1) type=cycle(CT) ...;
component cycle = cycleState[1];
```

The CT option must be included in the use of **TYPE=CYCLE**.

Models with Dependent Lags

Many useful time series models relate the present value of a response variable to its own lagged values and, in the multivariate case, the lagged values of other response variables in the model. In the CSSM procedure, you can use the **DEPLAG** statement to specify the terms in the model that involve lagged response variables. These models apply only to the regular data type. This section describes the state space form of such models; for more information, see Harvey (1989, sec. 7.1.1). As an illustration, consider the following model, where the q -dimensional coefficient matrices Φ_1 and Φ_2 are either fully or partially known:

$$\begin{aligned} \mathbf{Y}_t &= \Phi_1 \mathbf{Y}_{t-1} + \Phi_2 \mathbf{Y}_{t-2} + \mathbf{Z}_t \alpha_t + \mathbf{X}_t \beta + \epsilon_t \\ \alpha_{t+1} &= \mathbf{T}_t \alpha_t + \mathbf{W}_{t+1} \gamma + \mathbf{c}_{t+1} + \eta_{t+1} \\ \alpha_1 &= \mathbf{c}_1 + \mathbf{A}_1 \delta + \eta_1 \end{aligned}$$

Except for the presence of the terms that involve lagged response vectors ($\Phi_1 \mathbf{Y}_{t-1}$ and $\Phi_2 \mathbf{Y}_{t-2}$) in the observation equation, the form of this model is the same as the standard state space form that is described in

the section “State Space Model and Notation” on page 760. It turns out that this model can be expressed in the standard state space form by suitably enlarging the latent vectors in the state equation and by appropriately reorganizing the system matrices. The enlarged latent vectors and the corresponding system matrices are distinguished by the presence of dagger ($\dagger$) as a superscript in the following reformulated model,

$$\begin{aligned} \mathbf{Y}_t &= \mathbf{Z}_t^\dagger \boldsymbol{\alpha}_t^\dagger \\ \boldsymbol{\alpha}_{t+1}^\dagger &= \mathbf{T}_t^\dagger \boldsymbol{\alpha}_t^\dagger + \mathbf{W}_{t+1}^\dagger \boldsymbol{\gamma}^\dagger + \mathbf{c}_{t+1}^\dagger + \boldsymbol{\eta}_{t+1}^\dagger \\ \boldsymbol{\alpha}_1^\dagger &= \mathbf{c}_1^\dagger + \mathbf{A}_1^\dagger \boldsymbol{\delta}^\dagger + \boldsymbol{\eta}_1^\dagger \end{aligned}$$

where the following conditions are true (column vectors are displayed horizontally to save space):

- The enlarged state vector ($\boldsymbol{\alpha}_t^\dagger$) is formed by vertically stacking the old state vector ($\boldsymbol{\alpha}_t$), the observation disturbance vector ($\boldsymbol{\epsilon}_t$), and the present and lagged response vectors ($\mathbf{Y}_t$ and $\mathbf{Y}_{t-1}$, respectively). That is, $\boldsymbol{\alpha}_t^\dagger = [\boldsymbol{\alpha}_t \ \boldsymbol{\epsilon}_t \ \mathbf{Y}_t \ \mathbf{Y}_{t-1}]$. Because $\boldsymbol{\alpha}_t$ is m -dimensional and $\boldsymbol{\epsilon}_t$, $\mathbf{Y}_t$, and $\mathbf{Y}_{t-1}$ are q -dimensional, the dimension of $\boldsymbol{\alpha}_t^\dagger$ is $m^\dagger = (m + 3 * q)$.
- The new state regression vector ($\boldsymbol{\gamma}^\dagger$) is formed by vertically stacking the old state regression vector ($\boldsymbol{\gamma}$) and the observation equation regression vector ($\boldsymbol{\beta}$). That is, $\boldsymbol{\gamma}^\dagger = [\boldsymbol{\gamma} \ \boldsymbol{\beta}]$.
- The enlarged disturbance vector ($\boldsymbol{\eta}_t^\dagger$) is formed by vertically stacking the old state disturbance vector ($\boldsymbol{\eta}_t$), the observation disturbance vector ($\boldsymbol{\epsilon}_t$), the vector sum ($\mathbf{Z}_t \boldsymbol{\eta}_t + \boldsymbol{\epsilon}_t$), and filling the rest of the vector with zeros. That is, $\boldsymbol{\eta}_t^\dagger = [\boldsymbol{\eta}_t \ \boldsymbol{\epsilon}_t \ (\mathbf{Z}_t \boldsymbol{\eta}_t + \boldsymbol{\epsilon}_t) \ \mathbf{0}]$.
- The deterministic vector $\mathbf{c}_{t+1}^\dagger = [\mathbf{c}_{t+1} \ \mathbf{0} \ \mathbf{Z}_{t+1} \mathbf{c}_{t+1} \ \mathbf{0}]$.
- The last $2q$ elements of the initial state vector ($\boldsymbol{\alpha}_1^\dagger$), which correspond to $\mathbf{Y}_1$, and $\mathbf{Y}_0$, are taken to be diffuse (which means that the diffuse vector $\boldsymbol{\delta}^\dagger$ has $2q$ additional elements compared to $\boldsymbol{\delta}$).

The new system matrices can be described in blockwise form in terms of the old system matrices as follows:

- The $q \times (m + 3 * q)$ -dimensional $\mathbf{Z}_t^\dagger = [\mathbf{0} \ \mathbf{0} \ \mathbf{I} \ \mathbf{0}]$, where $\mathbf{0}$ is either a $q \times m$ -dimensional or $q \times q$ -dimensional matrix of zeros and $\mathbf{I}$ is a q -dimensional identity matrix.
- The $m^\dagger \times m^\dagger$ matrices $\mathbf{T}_t^\dagger$ (transition matrix) and $\mathbf{Q}_t^\dagger$ (covariance of $\boldsymbol{\eta}_{t+1}^\dagger$) are

$$\mathbf{T}_t^\dagger = \begin{bmatrix} \mathbf{T}_t & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{Z}_{t+1} \mathbf{T}_t & \mathbf{0} & \boldsymbol{\Phi}_1 & \boldsymbol{\Phi}_2 \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{Q}_t^\dagger = \begin{bmatrix} \mathbf{Q}_t & \mathbf{0} & \mathbf{Q}_t \mathbf{Z}_{t+1}' & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{t+1} & \boldsymbol{\Sigma}_{t+1} & \mathbf{0} \\ \mathbf{Z}_{t+1} \mathbf{Q}_t & \boldsymbol{\Sigma}_{t+1} & (\mathbf{Z}_{t+1} \mathbf{Q}_t \mathbf{Z}_{t+1}' + \boldsymbol{\Sigma}_{t+1}) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\boldsymbol{\Sigma}_t$ denotes the covariance matrix (which is diagonal by design) of the observation error vector $\boldsymbol{\epsilon}_t$. Recall that the system matrices in the transition equation can depend on both t and $t + 1$ even if the subscripts of $\mathbf{T}$ and $\mathbf{Q}$ show dependence on t alone.

- The $m^\dagger \times (k + g)$ matrix $\mathbf{W}_t^\dagger$ is

$$\mathbf{W}_{t+1}^\dagger = \begin{bmatrix} \mathbf{W}_{t+1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{Z}_{t+1} \mathbf{W}_{t+1} & \mathbf{X}_{t+1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

This state space form can be easily extended to account for higher-order lags.

Models that contain dependent lag terms must be used with care. Because the CSSM procedure does not impose any special constraints on the lag coefficients (the elements of coefficient matrices Φ_1 , Φ_2 , and so on), the resulting models can often be explosive. For an example of a model with lagged response variables, see “[Example 14.13: Bivariate Model for Sales of Mink and Muskrat Furs](#)” on page 871.

PROC CSSM and PROC UCM (see Chapter 30, “[UCM Procedure](#)”) handle models that contain dependent lags in essentially the same way. However, there is one difference: if the model parameter vector contains unknown lag parameters, PROC UCM parameters are estimated by optimizing the nondiffuse part of the likelihood, whereas PROC CSSM continues to use the full diffuse likelihood for parameter estimation.

Simulation Smoothing

Like the Kalman filter and Kalman smoother algorithms (DKFS) that are described in the section “[Filtering, Smoothing, Likelihood, and Structural Break Detection](#)” on page 767, the simulation smoother algorithm is an important tool in SSM-based data analysis. The simulation smoother in the CSSM procedure is loosely based on the algorithm described in Durbin and Koopman (2012, chap. 4, sec. 9). Recall that the smoothing phase of DKFS provides the mean and the covariance of a latent state at any time point t , $E(\alpha_t|Y)$ and $\text{Cov}(\alpha_t|Y)$, conditional on the full observed sample Y (see the section “[Smoothing Phase](#)” on page 772). However, the DKFS does not provide information about the joint conditional distribution at different time points; for example, it does not provide the conditional covariance $\text{Cov}(\alpha_{t_1}, \alpha_{t_2}|Y)$ at two distinct time points, t_1 and t_2 . The simulation smoother provides a solution to this problem. You can use the simulation smoother to generate a random draw, $(\tilde{\beta}, \tilde{\gamma}, \tilde{\alpha}_1, \dots, \tilde{\alpha}_n)$, from the joint conditional distribution of $(\beta, \gamma, \alpha_1, \dots, \alpha_n)$, given the sample Y . By making a sufficiently large number of draws from the joint conditional distribution, you can obtain the sampling distribution of arbitrarily complex functions of the conditional estimates of $(\beta, \gamma, \alpha_1, \dots, \alpha_n)$.

In PROC CSSM, the simulation smoothing process is divided into two steps. In the first step, you save the fitted model to an output BLOB table by using the `OUTMODEL=` option in the `SIMSMOOTH` statement as follows:

```
proc cssm data=data-table;
  id ...;
  /*---model specification---*/
  ...;
  simsmooth outmodel=blob-table;
run;
```

In the second step, you use the input data table that was used for model fitting and the BLOB table that was saved from the first step to generate the desired number of simulations from the conditional distribution as follows:

```
proc cssm data=data-table;
  simsmooth inmodel=blob-table nsim=integer seed=integer
    outstate=output-table outcomp=output-table
    outpred=output-table;
run;
```

In this second step, because the loaded `INMODEL=` option BLOB table already contains the full model specification, you do not need to specify the statements that describe the model. Moreover, you can repeat

this second step as many times as needed to get the desired simulation output. The primary output of the simulation smoother is the desired number of random draws of the latent states. You can save this output for later use in a table that you specify in the **OUTSTATE=** option. Depending on the size of the underlying state vector, the number of distinct time points in the input data table, and the specified number of random draws, the **OUTSTATE=** table can get quite large. In many practical situations, you might not need to save this level of detail. In fact, you might be interested only in the simulations of the predictions of the missing response values or of the components in the model. Because they are known linear combinations of the latent states, you can easily obtain them as a by-product of the state simulation output. You can save the simulations of the predictions of the missing response values in a table by using the **OUTPRED=** option, and you can save the simulations of the components in the model by using the **OUTCOMP=** option. The **OUTPRED=** and **OUTCOMP=** tables are usually much smaller than the **OUTSTATE=** table. For more information about the contents of these tables, see the section “Output Tables Associated with Simulation Smoothing” on page 788. For an application of simulation smoothing, see “Example 14.21: Simulation-Smoothing-Based Sampling Distribution of the Sum of Forecasts” on page 901.

Output Tables Associated with Simulation Smoothing

The simulation smoothing process in PROC CSSM enables you to organize the simulation output in three tables: the **OUTSTATE=** table, which contains the simulations of the latent states; the **OUTCOMP=** table, which contains the simulations of the model components; and the **OUTPRED=** table, which contains the simulations of the predictions of the missing response variable values. All these tables share three columns: the **BY** statement variables column; the **SEED** variable column, which contains the seed value that is associated with the random draw; and the **ID** statement variable column. The values of the **SEED** variable range from *integer* to *integer+nsim-1*, where *integer* is the value of the **SEED=** option and *nsim* is the number of draws that you specify in the **NSIM=** option. Additionally, the mean of the sampling distribution for each simulated quantity is also included for reference, and the **SEED** value that is assigned to it is always -1. The mean of the sampling distribution is computed in the smoothing phase (for more information, see the section “Smoothing Phase” on page 772). The other columns in these tables depend on the specified model. The details are presented in the following sections.

OUTCOMP= Table

The **OUTCOMP=** table has the following columns:

- **BY** variables
- **SEED**
- **ID** variable
- **OBSINDEX** variable, if present
- **Smoothed_compName**, where *compName* is a component that is specified in either a **COMPONENT**, **EVAL**, **IRREGULAR**, or **TREND** statement

OUTPRED= Table

The OUTPRED= table has the following columns:

- BY variables
- SEED
- ID variable
- OBSINDEX variable, if present
- SimSmooth_response, where response is a response variable

OUTSTATE= Table

The OUTSTATE= table has the following columns:

- BY variables
- SEED
- ID variable
- stateName_STATE_i, which denotes the *i*th element of state named stateName
- stateName_STATE_REG_i, which denotes the *i*th element of the state regression vector (if present) for the state named stateName
- trendName_TREND_STATE_i, which denotes the *i*th element of the state that underlies the trend named trendName
- irregularName_STATE_i, which denotes the *i*th element of the state that underlies an irregular component, named irregularName, if it is part of the model state vector. In some cases, such as when a model contains a dependent lag term, the irregular component is part of the model state vector.
- deplagName_STATE_i, which denotes the *i*th element of the state that underlies a dependent lag term named deplagName
- response_COEFF_predVar, which denotes the regression coefficient of a predictor variable, predVar, that is specified in the model for a response variable named response

Temporal Aggregation and Temporal Distribution

The response variables in time series analysis are often classified as either stock variables or flow variables. Stock variables, such as interest rates or temperatures, are measured at a particular point in time. Flow variables, such as monthly income or weekly sales, are defined with respect to an interval of time. Flow variables have the property that they remain meaningful under the operations of temporal aggregation and temporal distribution—for example, aggregation of daily sales to weekly sales and distribution (or disaggregation) of weekly sales to daily sales are quite natural, whereas the same cannot be said of temperature readings. This section explains how you can use the CSSM procedure to do model-based temporal aggregation and distribution of flow variables. State space models are often used to carry out model-based temporal aggregation and distribution. Two properties of state space models make them particularly suitable for this purpose:

- If a variable is modeled by a state space model at a particular time interval, its aggregated form—for example, daily to monthly—also follows a state space model. Moreover, the state space forms of these two models have a simple relationship.
- State space models can easily handle missing response values.

The discussion in this section, which is based on Harvey (1989, chap. 6, sec. 3), is limited to regular data types—that is, the data must be either univariate or multivariate time series.

Temporal Distribution

For the sake of simplicity, consider a simple case of distributing weekly observations of a flow variable, y , at a daily interval. Even though the values of y are observed weekly (suppose they are recorded each Sunday), in this case it is necessary to treat the observations $y_t, t \geq 1$, as a daily time series such that y_t equals the weekly total when t corresponds to the end of the week (Sunday), and y_t is missing on other days of the week. In addition, suppose that $y_t^\dagger$ denotes the unobserved time series of daily values of y . In other words, if t corresponds to a Sunday, then

$$y_t = \sum_{s=t-6}^t y_s^\dagger$$

Suppose that the unobserved daily series $y_t^\dagger$ can be modeled by a state space model. For example, suppose the model for $y_t^\dagger$ is

$$\begin{aligned} y_t^\dagger &= \mathbf{Z}_t \boldsymbol{\alpha}_t + \epsilon_t \\ \boldsymbol{\alpha}_{t+1} &= \mathbf{T}_t \boldsymbol{\alpha}_t + \boldsymbol{\eta}_{t+1} \end{aligned}$$

Then it is easy to see that the aggregated series y_t follows a state space model of the form

$$\begin{aligned} y_t &= \mathbf{Z}_t^\dagger \boldsymbol{\alpha}_t^\dagger \\ \boldsymbol{\alpha}_{t+1}^\dagger &= \mathbf{T}_t^\dagger \boldsymbol{\alpha}_t^\dagger + \boldsymbol{\eta}_{t+1}^\dagger \end{aligned}$$

where the following are true (both the row and column vectors are displayed horizontally to save space):

- The new state vector ($\alpha_t^\dagger$) is formed by augmenting the old state vector (α_t) with a latent variable, y_t^f . That is, $\alpha_t^\dagger = [\alpha_t \ y_t^f]$. In fact, y_t^f represents the within-week running total of $y_t^\dagger$, so that when t corresponds to a Sunday, $y_t^f = y_t$.
- The new transition matrix $T_t^\dagger$ is

$$T_t^\dagger = \begin{bmatrix} T_t & 0 \\ Z_{t+1} T_t & \psi_{t+1} \end{bmatrix}$$

where ψ_t is a dummy variable that equals 1 when t is not the start of the week (not Monday) and equals 0 when t is the start of the week (Monday).

- The new disturbance vector ($\eta_t^\dagger$) is formed by augmenting the old disturbance vector (η_t) by $(Z_t \eta_t + \epsilon_t)$. That is, $\eta_t^\dagger = [\eta_t \ Z_t \eta_t + \epsilon_t]$.
- The new design matrix for the state effect ($Z_t^\dagger$) is $Z_t^\dagger = [0 \ 1]$, where 0 is a zero vector of the same size as the old state vector α_t .

This shows that you can do model-based distribution of y values by carrying out the following steps:

- 1 Organize the y values as a daily time series.
- 2 Define a dummy variable, `startWeek`, that flags the start of the week—that is, `startWeek` is 1 when the day is Monday and 0 otherwise. Note that $\psi_t = 1 - \text{startWeek}_t$.
- 3 Specify a suitable state space model for the unobserved daily series $y_t^\dagger$. This specification in turn implies a state space model specification for y .
- 4 Carry out the analysis—model fitting, component estimation, and forecasting—of y in the usual fashion by using this implied model specification.
- 5 The smoothed values of y from the previous step provide the estimates of y_t^f . In addition, the estimates of $y_t^\dagger$ can be obtained as the smoothed estimates of appropriate linear combination of the elements of α_t and ϵ_t .

The CSSM procedure enables you to carry out the key steps—Step 3 to Step 5—in this process quite easily. The usual model specification syntax that uses the `STATE`, `COMPONENT`, and `TREND` statements to define the terms in a `MODEL` statement is used to define a model for the unobserved daily series $y_t^\dagger$ (the first part of Step 3). Then, the use of the `DISTRIBUTE(START=startWeek)` option in the `MODEL` statement causes the CSSM procedure to use the implied model to analyze the observed y values. As a brief illustration, suppose that a data set `Test` contains two variables: `date`, a SAS date variable that indexes the daily observations, and `y`, the values of the weekly variable arranged as a daily series. Then the following PROC CSSM statements show you how to distribute y at the daily interval:

```
proc cssm data=mylib.test;
  id date interval=day;
  startWeek = (weekday(date) = 2); /* indicator of Monday */
  state ...;
  comp term1 = ...;
  ...;
```

```

state noise(1) type=wn ...;
comp wnoise = noise[1];
model y = term1 term2 ... wnoise / distribute(start=startWeek);
/* daily_Y = sum of all terms in the MODEL statement */
eval daily_Y = term1 + term2 + ...+ wnoise;
output out=...;
run;

```

Here are a few comments about this program:

- The terms in the MODEL statement correspond to the observation equation for the unobserved daily series $y_t^\dagger$. However, the DISTRIBUTE(START=*startWeek*) option causes the CSSM procedure to use the implied model (with the augmented state vector) to analyze y —the weekly variable arranged as a daily series.
- Because *wnoise*—the white noise term (ϵ_t) in the observation equation of $y_t^\dagger$ —is subsequently to be used in an EVAL statement, this program specifies it by using the STATE statement rather than by using the IRREGULAR statement.
- Because *daily_Y* (the component specified in the EVAL statement) is the sum of all the terms in the MODEL statement, it corresponds to the unobserved daily series $y_t^\dagger$. Therefore, the smoothed estimate of *daily_Y* (*smoothed_daily_Y*) provides the needed distribution of y at the daily interval.

In this release of the CSSM procedure, the last element of the augmented state vector, y^f , is always initialized with diffuse distribution. A more flexible specification of the initial distribution of y^f might become possible in a future release.

To keep the explanation simple, the preceding discussion was confined to a single response variable. In fact, you can use the CSSM procedure for temporal distribution in more general settings—for example, you can consider temporal distribution of one or more flow variables in a multivariate model that includes one or more response variables of stock type, one or more response variables of flow type, and one or more explanatory variables. An illustration of such modeling is shown in “[Example 14.16: Estimating Monthly GDP by Temporal Distribution](#)” on page 884. The modeling of a response variable as a temporal aggregate of some unobserved latent variable is also needed in a process known as benchmarking; see Durbin and Koopman (2012, chap. 3, sec. 10.2) and Pelagatti (2015, chap. 9, sec. 2). You can use the CSSM procedure in such benchmarking situations as well.

NOTE: Model specification in the temporal distribution setting requires some care. The model for the lower-frequency observed series, y_t , is based on the model specified for the unobserved higher-frequency series $y_t^\dagger$. Because aggregation from higher frequency to lower frequency involves loss of information, an otherwise identifiable model for $y_t^\dagger$ can lead to unidentifiable model for y_t .

Temporal Aggregation

Temporal aggregation is the reverse of temporal distribution. In this case, the observations are available on a finer time scale, and you are interested in estimating the aggregated values on some coarser time scale—for example, estimating weekly totals from daily data. Of course, the aggregation is trivial in the historical region where the observations on the finer scale are known—in fact, in this case the estimation of the aggregate values is done with no estimation error. However, when the aggregate values are to be estimated in the region where the observations on the finer scale are missing—for example, in the forecast

region—the problem becomes nontrivial. It is easier to explain the situation by using a simple example. Suppose y_t , $1 \leq t \leq 100$, denote the daily observations of a response variable, y . Let wy_t , $t \geq 1$, denote the within-week daily running totals—that is, wy_t represents the total of y values up to the day t in the week that contains the day t . Clearly, given the daily values y_t , $1 \leq t \leq 100$, the aggregate values wy_t , $1 \leq t \leq 100$, are fully known. The question is, assuming that y follows a state space model, how do you estimate and obtain appropriate confidence intervals for wy_t in the forecast region ($t = 101, 102, \dots$)? In this section you have already seen that when a variable is modeled by a state space model at a particular time interval, its aggregated form also follows a state space model. The AGGREGATE(START=) option in the MODEL statement of the CSSM procedure enables you to perform temporal aggregation for a response variable. An illustration of such aggregation is shown in “[Example 14.17: Temporal Aggregation-Based Triannual Nile River Level](#)” on page 889.

Covariance Parameterization

The covariance matrices specified by the COV and COV1 options in the STATE statement must be positive semidefinite. When these matrices are of general form and are not user-specified, they are internally parameterized by their Cholesky root. Suppose that Σ , an $m \times m$ positive semidefinite matrix of rank r , is such a covariance matrix. Then, Σ can always be written as

$$\Sigma = RR'$$

where the (generalized) Cholesky root, R , is an $m \times r$ lower triangular matrix with nonnegative diagonal elements (that is, $R[i, j] = 0$ if $j > i$ and $R[i, i] \geq 0$, $1 \leq i \leq r$). The CSSM procedure parameterizes Σ by the elements of its Cholesky root, which adds $r * (r + 1)/2 + r * (m - r)$ elements to the parameter vector θ .

Missing Values

For a variety of reasons the data might contain missing response and predictor values. Before starting the analysis of a particular BY group, CSSM procedure makes an internal copy of the data. The actual analysis is done by using this copy. The data in the copy are first examined for missing values in the response, predictor, and the ID variables. No missing values are permitted in the ID variable (if it is specified). If all the missing values are associated with only the response variables, then the internal copy of the data is not altered. However, if any of the predictors in the observation equation—the elements of $\mathbf{X}$ matrix—are found to contain missing values, the internal copy of the data is altered as follows: any missing predictor value is replaced by 0, and the response values that are dependent on that predictor in the corresponding row are set to missing. These missing response values are called the *induced missing values*. The reported analysis is based on the (possibly altered) internal copy of the BY group.

Missing values are not permitted in any of the other system matrices that define the state space model. In particular, missing values are not permitted in $\mathbf{Z}$, $\mathbf{T}$, $\mathbf{W}$, and $\mathbf{Q}$ matrices. In some cases the elements of these matrices depend on the data values. In such cases, care must be taken to ensure that these data values are not missing.

Scoring with State Space Models

Evaluating a previously fitted model at some new predictor setting is often called scoring. For example, a bank might score a new loan application by applying a rule that is based on a previously fitted logistic regression model. Similarly, you might want to use state space models for scoring. However, because of the sequential nature of the observation process, it is important to realize that the scoring process for state space models is inherently different from the scoring process for models that are based on independent observations, such as simple linear regression or logistic regression. Think of a sequential data process as a streaming movie, which you can pause and resume playing. Watching the movie for some time and learning about the main characters and their relationships to each other is akin to fitting a model to the initial portion of a sequential data stream. After you have learned something about the movie, you can make educated guesses about the subsequent plotline. You can also imagine different ways the movie could play out, while still being consistent with the story thus far. For sequential data, this process of forecasting and what-if analysis, based on the fitted model and ever-increasing history, is called scoring. To explain the scoring features of the CSSM procedure, it is useful to define the following notation, which is based on the section “[State Space Model and Notation](#)” on page 760:

- $(\{Y_t, X_t, W_t\}, t = 1, 2, \dots)$ denote sequential observations on the response variables Y and, optionally, the predictor variables X and W .
- For time points $t_1 \leq t_2$, let $D_{t_1:t_2}$ denote the data between t_1 and t_2 ; that is, $D_{t_1:t_2} = (\{Y_t, X_t, W_t\}, t_1 \leq t \leq t_2)$. With this notation, it is easy to describe a streaming data process in which data arrive sequentially in data packets, possibly of varying lengths. For example, the data packets $D_{1:100}, D_{101:110}, D_{111:112}, \dots$ denote the data that are associated with the first 100 time indices, the data between time indices 101 and 110, the data between time indices 111 and 112, and so on. When there can be multiple measurements that use the same time index, such as in the case of panel data, the ending time index of one data packet can be the same as the starting time index of the next data packet.
- α_t denotes the state vector at time index t , θ denotes the parameter vector, δ denotes the diffuse vector in the initial condition, and β and γ denote the regression vectors that are associated with the observation equation and state equation, respectively.

Armed with this notation, you can understand the main activities of scoring with a state space model, which are explained in the following sections.

Initial Score-Store Creation

After you fit a state space model by using a sufficiently long series of data and you determine that the model is satisfactory, it is ready to be used for scoring. The first step in this process is the creation of a score store. You can use the SCORE statement to create the initial score store, such as `mylib.inscore`, by fitting the model to a suitable series of input data $D_{1:T}$ (`score out=mylib.inscore;`). From this stage forward, the fitted model is assumed to hold for all future time points ($t \geq T$). In particular, the estimated parameter vector, $\hat{\theta}$, is treated as the true and known parameter vector in all subsequent steps. The score store `mylib.inscore` stores the following information:

- the model specification and the estimated parameter vector, $\hat{\theta}$. As part of the model specification, the structural break detection settings in the [STATE](#) and [TREND](#) statements are also stored. Similarly, the

settings of the BREAK and AO options in the **OUTPUT** statement are also saved. If necessary, the options in the **OUTPUT** statement can be modified in later scoring calls.

- $\tilde{\alpha}_T, \tilde{\delta}_T, \tilde{\beta}_T, \tilde{\gamma}_T$, and their covariances, which denote the full-sample (smoothed) estimates of these quantities on the basis of the data $\mathbf{D}_{1:T}$ (for more information, see the section “[Smoothing Phase](#)” on page 772).
- If the score store is created using **OUT(k)=** syntax (**score out (k)=mylib.inscore;**), then $\tilde{\alpha}_{T-k}, \tilde{\delta}_{T-k}, \tilde{\beta}_{T-k}, \tilde{\gamma}_{T-k}$, and their covariances are stored; these are full-sample estimates of these quantities based on the data $\mathbf{D}_{1:(T-k)}$. In this case, the data rows in $\mathbf{D}_{(T-k):T}$ are also stored.

Because of the Markovian property of state space models, the information that the score store contains is sufficient for you to forecast subsequent observations of the process, without having access to its past history.

Incorporating New Information

After you create an initial score store, such as `mylib.inscore`, the scoring process no longer needs the past data ($\mathbf{D}_{1:T}$) that you used to create it. However, as time passes and a data packet, $\mathbf{D}_{(T+1):(T+p)}$, arrives containing new information content, you need to create an updated score store, such as `mylib.outscore`, that incorporates the new information. Because they use the latest information, of course the forecasts for time points after $T + p$ that are based on `mylib.outscore` are better than the forecasts that are based on the old score store, `mylib.inscore`. In PROC CSSM, you can create the updated score store by using the data packet that contains new information ($\mathbf{D}_{(T+1):(T+p)}$) as the input data table and by using the **SCORE** statement with both the **IN=** and **OUT=** options as follows:

```
score in=mylib.inscore out=mylib.outscore;
```

After this, you can discard the old score store, `mylib.inscore`. The new score store, `mylib.outscore`, becomes the score store that is based on the full history up to time $T + p$ ($\mathbf{D}_{1:(T+p)}$). If you created the initial score store by using **OUT(k)=** syntax, the updated score store also stores the data rows in $\mathbf{D}_{(T+p-k):(T+p)}$. You can repeat this score-store updating process again and again as newer information arrives.

What-If Analysis and Process Control

Suppose that a score-store, such as `mylib.inscore`, contains context information based on the data up to time T ($\mathbf{D}_{1:T}$), and suppose you are interested in predicting and/or interpolating the response sequence for the next p time points. You can easily accomplish this in PROC CSSM by using a suitable data packet $\mathbf{D}_{(T+1):(T+p)}$ (or $\mathbf{D}_{T:(T+p)}$ when multiple observations can share the same time index) as the input data table and by using `mylib.inscore` as the input score store in the **SCORE** statement (**score in=mylib.inscore;**). This is called the scoring of data in $\mathbf{D}_{(T+1):(T+p)}$. Because the missing values in the response variables are an indication to PROC CSSM to compute model-based predictions/interpolations, you must set the values of **Y** to missing in appropriate rows of $\mathbf{D}_{(T+1):(T+p)}$ (of course, you must provide appropriate nonmissing values for the predictor variables **X** and **W**).

In what-if analysis, you are interested in predicting the future course of response values for different settings of the predictors **X** and **W**. You can achieve this by repeatedly scoring $\mathbf{D}_{(T+1):(T+p)}$ for different settings of **X** and **W**, while keeping the input score-store the same. See “[Example 14.19: Scenario Analysis Using Scoring](#)” on page 894 for an illustration of this type of what-if analysis.

In process control, the goal is to find suitable predictor settings so that the future response values are close to some desirable levels. In an informal, quick-and-dirty approach, you can do this by repeatedly scoring $\mathbf{D}_{(T+1):(T+p)}$ with different trial values of predictors. In a formal approach, you can use the scoring functionality to define a suitable objective function, which you can optimize by using sophisticated optimization methods to find the optimal predictor settings.

Monitoring of Streaming Data

In many fields, such as industrial process control, monitoring of atmospheric pollutant levels, and so on, a sequential data stream is monitored continuously. The goal is to flag unexpected changes in the normal behavior of the process as soon as possible. Scoring with state space models can be quite useful in these situations. See “[Example 14.20: Monitoring Streaming Data Using Scoring](#)” on page 897 for an illustration of how to use SSM-based scoring to detect level shift in a quarterly series of traffic accident data from the United Kingdom. The following list describes the usual steps in an SSM-based work flow in monitoring cases such as these:

- Find a good model for the normal behavior of the process that is being monitored.
- Create an initial score-store by using a suitably long series of data, $\mathbf{D}_{1:T}$. At creation time, use the $\text{OUT}(k)=$ form with a suitable data window length k , and set the desired structural break detection settings, such as the CHECKBREAK and MAXSHOCK options in the appropriate statements.
- After the initial score-store creation, the process monitoring starts as the new data packets $\mathbf{D}_{(T+1):(T+p)}$, $\mathbf{D}_{(T+p+1):(T+2p)}$, $\dots$ arrive and are scored in succession.
- Each time a new data packet arrives, it is scored, and an updated score store is created that becomes the input score store for processing the next data packet.
- As seen in “[Example 14.20: Monitoring Streaming Data Using Scoring](#)” on page 897, a structural break that happens at a particular time point to be confirmed as a structural break often requires several subsequent observations to be scored. You must make an appropriate choice of k at the time of score-store creation to ensure that a sufficient number of past data rows are retained for detecting structural breaks.
- If a structural break is confirmed at a time point, you can take the following types of action:
 - Flag the structural break, and continue the process monitoring without any further action.
 - Correct the model to take into account the structural break, and continue monitoring.
 - Pause the process, address the causes of the break, and resume monitoring (possibly after modifying the model and refitting).

Computational Issues

A Well-Behaved Model

The model defined by the state space model equations (see the section “[State Space Model and Notation](#)” on page 760) is very general. This generality is quite useful because it encompasses a wide variety of data generation processes. On the other hand, it also makes it easy to specify overly complex and numerically unstable models. If a suitable model is not already known and you are in the early phases of modeling, it is important to start with models that are relatively simple and well-behaved from the numerical standpoint. From the numerical and statistical considerations, two aspects of model formulation are particularly important: identifiability and numerical stability. A model is identifiable if the observed data has a distinct probability distribution for each admissible parameter vector. Unless proper care is taken, it is easy to specify an unidentifiable state space model. Similarly, predictions according to some types of state space models can display explosive growth or wild oscillations. This behavior is primarily governed by the transition matrix $\mathbf{T}$ (or $\mathbf{T}_t$ in the time-varying case). Unidentifiable models can run into difficulties during parameter estimation, and explosive growth (and wild oscillation) causes numerical problems associated with finite-precision arithmetic. Unfortunately, no simple identifiability check is available for a general state space model, and it is difficult to decide at the outset whether a specified model might suffer from numerical instability. For a discussion of identifiability issues, see Harvey (1989, chap. 4, sec. 4). For a discussion of the stability properties of time-invariant state space models, see Harvey (1989, chap. 3, sec. 3). The following guidelines are likely to result in models that are identifiable and numerically stable:

- Build models by composing submodels that are known to be well-behaved. The predefined models provided by the CSSM procedure are good submodel candidates (see the sections “[Predefined Trend Models](#)” on page 776 and “[Predefined Structural Models](#)” on page 779).
- Pay careful attention to the way the variety of system matrices are defined. The behavior of their elements, as functions of model parameters and other variables, must be well-understood. If these elements are defined by using DATA steps, you can validate their behavior by running these DATA steps outside of the CSSM procedure. In particular, note the following:
 - The transition matrix $\mathbf{T}$ (or $\mathbf{T}_t$ in the time-varying case) determines the explosiveness characteristics of the model; it must be well-behaved for all parameters.
 - The disturbance covariances $\mathbf{Q}_t$ must be positive semidefinite for all parameters.
 - If the system matrices in the state equation, such as the transition matrix $\mathbf{T}_t$ or the disturbance covariance $\mathbf{Q}_t$, are time-varying and the data contain replicate observations (observations with the same ID value), check that the elements of these matrices do not vary during replicate observations. This follows from the fact that the underlying state does not vary during replications (see the state equation in the section “[State Space Model and Notation](#)” on page 760 and the section “[Types of Sequence Data](#)” on page 762).

Convergence Problems

As explained in the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769, the model parameters are estimated by nonlinear optimization of the likelihood. This process is not guaranteed to succeed. For some data sets, the optimization algorithm can fail to converge. Nonconvergence can result from a number of causes, including flat or ridged likelihood surfaces and ill-conditioned data. It is also possible for the algorithm to converge to a point that is not the global optimum of the likelihood.

If you experience convergence problems, consider the following:

- Data that are extremely large or extremely small can adversely affect results because of the internal tolerances used during the filtering steps of the likelihood calculation. Rescaling the data can improve stability.
- Whenever possible, parameterize the disturbance variances in the model on the exponential scale. For illustrations of parameterizing disturbance variances in this manner, see “[Example 14.12: Long-Term Temperature Trends Based on a Multivariate Model](#)” on page 864 and “[Example 14.14: A Factor Model for Now-Casting the US Economy](#)” on page 876.
- Examine your model for redundancies in the included components and regressors. The components or regressors that are nearly collinear to each other can cause the optimization process to become unstable.
- Lack of convergence can indicate model misspecification such as unidentifiable model or a violation of the normality assumption.

Computer Resource Requirements

The computing resources required for the CSSM procedure depend on several factors. The memory requirement for the procedure is largely dependent on the number of observations to be processed and the size of the state vector underlying the specified model. If n denotes the sample size and m denotes the size of the state vector, the memory requirement for the smoothing phase of the Kalman filter is of the order of $6 \times 8 \times n \times m^2$ bytes, ignoring the lower-order terms. If the smoothed component estimates are not needed, then the memory requirement is of the order of $6 \times 8 \times (m^2 + n)$ bytes. Besides m and n , the computing time for the parameter estimation depends on the size of the parameter vector θ and how many likelihood evaluations are needed to reach the optimum.

Default Displayed Output

The default printed output produced by the CSSM procedure contains the following information:

- brief information about the input data set, including the data set name and label
- summary statistics for the response variables in the model, including the names of the variables, the total number of observations and the number of missing observations, the smallest and largest measurements, and the mean and standard deviation

- information about the index variable, including the index value of the first and the last observation, the maximum difference between the successive index values, the number of distinct index values, and the categorization of the data into regular, regular with replication, or irregular types
- estimates of the regression parameters if the model contains any predictors, including their standard errors, t statistics, and p -values
- convergence status of the likelihood optimization process if any parameters are estimated
- estimates of the free parameters at the end of the model-fitting phase, including the parameter estimates and their approximate standard errors
- the likelihood-based goodness-of-fit statistics, including the full likelihood, the sum of squares of residuals normalized by their standard errors, and the information criteria: AIC, AICC, HQIC, BIC, and CAIC
- summary of most significant additive outliers

Displayed Output

The CSSM procedure assigns a name to each table it creates. You can use these names to refer to the table when you use the Output Delivery System (ODS) to select tables and create output data sets. These names are listed in [Table 14.9](#).

Table 14.9 ODS Tables Produced by PROC CSSM

ODS Table Name	Description	Statement	Option
Tables That Summarize the Model Information			
ModelSummary	Summary information about the underlying state space model		Default
IdInformation	Summary information about the ID variable		Default
ResponseInfo	Summary information about the response variables		Default
StateSummary	Summary information about the model state vector	PROC CSSM	STATEINFO
DiffuseStateSummary	Summary information about the diffuse initial state	PROC CSSM	STATEINFO
Tables Related to Model Parameters and the Likelihood			
ConvergenceStatus	Convergence status of the estimation process		Default
RegressionEstimates	Estimates of the regression parameters	MODEL	Default
StateRegressionEstimates	Estimates of the state regression parameters	STATE	W

Table 14.9 *continued*

ODS Table Name	Description	Statement	Option
FixedStateEstimates	Estimates of time-invariant, non-stochastic state subsections		Default
NamedParameterEstimates	Estimates of the parameters specified in the PARMS statement	PARMS	Default
ParameterEstimates	Estimates of the unknown elements in the model system matrices		Default
DisturbanceCovariance	Estimate of the disturbance covariance	STATE	PRINT=COV
InitialCovariance	Estimate of the initial state covariance	STATE	PRINT=COV1
ARCoefficient	Estimate of the autoregressive coefficient matrix	STATE	PRINT=AR
MACoefficient	Estimate of the moving-average coefficient matrix	STATE	PRINT=MA
TransitionMatrix	Estimate of the state transition matrix	STATE	PRINT=T
FitSummary	Summary of the likelihood-based fit-statistics		Default
InformationCriteria	Likelihood-based information criteria		Default
Tables Related to Series and Component Forecasts			
Forecasts	Series forecasts	MODEL	PRINT=FILTER
SmoothedResponse	Smoothed series values	MODEL	PRINT=SMOOTH
FilteredComponent	Component forecasts	COMPONENT	PRINT=FILTER
SmoothedComponent	Smoothed component	COMPONENT	PRINT=SMOOTH
Tables Related to Outlier Detection and Model Quality			
AOSummary	Summary of additive outliers	Default	Default
ElementTrendBreakSummary	Elementwise trend break summary	TREND	CHECKBREAK
OverallTrendBreakSummary	Overall trend break summary	TREND	CHECKBREAK(OVERALL)
StateElementBreakSummary	Elementwise state break summary	STATE	CHECKBREAK
OverallStateBreakSummary	Overall state break summary	STATE	CHECKBREAK(OVERALL)
MaximalShockSummary	Summary of maximal state shocks	OUTPUT	MAXSHOCK
PRESS	Prediction error sum of squares	OUTPUT	PRESS

Table 14.9 *continued*

ODS Table Name	Description	Statement	Option
Tables Related to Simulation Smoothing			
SimulationSummary	Summary of simulation smoothing specifications	Default	Default

ODS Graphics

Statistical procedures use ODS Graphics to create graphs as part of their output. ODS Graphics is described in detail in the “Statistical Graphics Using ODS” chapter in *SAS/STAT User’s Guide*.

Before you create graphs, ODS Graphics must be enabled (for example, by using the ODS GRAPHICS ON statement). For more information about enabling and disabling ODS Graphics, see the section “Enabling and Disabling ODS Graphics” in that chapter.

The overall appearance of graphs is controlled by ODS styles. Styles and other aspects of using ODS Graphics are discussed in the section “A Primer on ODS Statistical Graphics” in that chapter.

You can refer to every graph produced through ODS Graphics with a name. The names of the graphs that PROC CSSM generates are listed in [Table 14.10](#), along with the required statements and options.

Table 14.10 ODS Graphs Produced by PROC CSSM

ODS Graph Name	Description	Statement	Option
Graphs for One-Step-Ahead Residual Analysis			
ResidualNormalityPlot	Normality check	PROC CSSM	PLOTS=RESIDUAL(NORMAL)
ResidualHistogram	Residual histogram	PROC CSSM	PLOTS(UNPACK)=RESIDUAL
ResidualQQPlot	Residual Q-Q plot	PROC CSSM	PLOTS(UNPACK)=RESIDUAL
StdResidualPlot	Time series plot of standardized residuals	PROC CSSM	Default
Graphs Related to Outlier Detection and Structural Break			
PredErrorNormalityPlot	Normality check	PROC CSSM	PLOTS=AO(NORMAL)
PredErrorHistogram	Prediction error histogram	PROC CSSM	PLOTS(UNPACK)=AO
PredErrorQQPlot	Prediction error Q-Q plot	PROC CSSM	PLOTS(UNPACK)=AO
StdPredErrorPlot	Time series plot of standardized additive-outlier statistics	PROC CSSM	PLOTS=AO(STD)
MaximalShockPlot	Time series plot of maximal state shock chi-square statistics	PROC CSSM	PLOTS=MAXSHOCK

OUT= Data Set

You can use the OUT= option in the OUTPUT statement to store the series and component forecasts that are produced by PROC CSSM. Which columns are included in the data set depends on the model specification. The model can have one or more response variables, a variety of components that appear in the MODEL statement, and components specified by the EVAL statement. The OUT= data set contains the one-step-ahead and full-sample estimates of the response variables, and all these components.

The following list describes the columns of the data set:

- the BY variables
- the ID variable
- the OBSINDEX variable, if specified by the OBSINDEX statement
- the response series (more than one in the multivariate case)
- the following columns associated with the response series (the wildcard * is replaced by the name of one of the response variables):
 - FORECAST_* contains the one-step-ahead predicted values and the multistep forecasts of the response series.
 - RESIDUAL_* contains the difference between the actual and forecast values.
 - StdErr_* contains the standard error of prediction.
 - Lower_* and Upper_* contain the lower and upper forecast confidence limits.
 - Smoothed_* contains the smoothed values of the response variable.
 - StdErr_Smoothed_* contains standard errors of the smoothed values of the response variable.
 - AO_* contains the additive outlier estimate.
 - StdErr_AO_* contains the standard error of the additive outlier estimate.
- the following columns associated with the components (the wildcard * is replaced by the name of one of the components):
 - FORECAST_* contains the one-step-ahead predicted values and the multistep forecasts of the component.
 - StdErr_* contains the standard error of prediction.
 - Smoothed_* contains the smoothed values of the component.
 - StdErr_Smoothed_* contains standard errors of the smoothed values of the component.
 - Smoothed_Lower_* and Smoothed_Upper_* contain the lower and upper confidence limits of the smoothed component.
- the maximal state shock chi-square statistics at distinct time points (this column is present only if the MAXSHOCK option is used in the OUTPUT statement)

Confidence limits are not produced for the smoothed series values or for the component forecasts; they are produced for the smoothed components.

Examples: CSSM Procedure

Example 14.1: Bivariate Basic Structural Model

This example illustrates how you can use the CSSM procedure to analyze a bivariate time series. The following data set contains two variables, `f_KSI` and `r_KSI`, which are measured quarterly, starting the first quarter of 1969. The variable `f_KSI` represents the quarterly average of the log of the monthly totals of the front-seat passengers killed or seriously injured during the car accidents, and `r_KSI` represents a similar number for the rear-seat passengers. The data set has been extended at the end with eight missing values, which represent four quarters, to cause the CSSM procedure to produce model forecasts for this span.

```
data seatBelt;
input f_KSI r_KSI @@;
label f_KSI = "Front Seat Passengers Injured--log scale";
label r_KSI = "Rear Seat Passengers Injured--log scale";
date = intnx( 'quarter', '1jan1969'd, _n_-1 );
format date YYQS.;
datalines;
  6.72417 5.64654 6.81728 6.06123 6.92382 6.18190
  6.92375 6.07763 6.84975 5.78544 6.81836 6.04644
  7.00942 6.30167 7.09329 6.14476 6.78554 5.78212
  6.86323 6.09520 6.99369 6.29507 6.98344 6.06194
  6.81499 5.81249 6.92997 6.10534 6.96356 6.21298
  7.02296 6.15261 6.76466 5.77967 6.95563 6.18993
  7.02016 6.40524 6.87849 6.06308 6.55966 5.66084
  6.73627 6.02395 6.91553 6.25736 6.83576 6.03535
  6.52075 5.76028 6.59860 5.91208 6.70597 6.08029
  6.75110 5.98833 6.53117 5.67676 6.52718 5.90572
  6.65963 6.01003 6.76869 5.93226 6.44483 5.55616
  6.62063 5.82533 6.72938 6.04531 6.82182 5.98277
  6.64134 5.76540 6.66762 5.91378 6.83524 6.13387
  6.81594 5.97907 6.60761 5.66838 6.62985 5.88151
  6.76963 6.06895 6.79927 6.01991 6.52728 5.69113
  6.60666 5.92841 6.72242 6.03111 6.76228 5.93898
  6.54290 5.72538 6.62469 5.92028 6.73415 6.11880
  6.74094 5.98009 6.46418 5.63517 6.61537 5.96040
  6.76185 6.15613 6.79546 6.04152 6.21529 5.70139
  6.27565 5.92508 6.40771 6.13903 6.37293 5.96883
  6.16445 5.77021 6.31242 6.05267 6.44414 6.15806
  6.53678 6.13404 . . . . .
run;
```

These data have been analyzed in Durbin and Koopman (2012, chap. 8, sec. 3). The analysis presented here is similar. To simplify the illustration, the monthly data have been converted to quarterly data and two predictors (the number of kilometers traveled and the real price of petrol) are excluded from the analysis. You can also use PROC CSSM to carry out the more elaborate analysis in Durbin and Koopman (2012).

One of the original reasons for studying these data was to assess the effect on `f_KSI` of the enactment of a seat-belt law in February 1983 that compelled the front seat passengers to wear seat belts. A simple graphical inspection of the data (not shown here) reveals that `f_KSI` and `r_KSI` do not show a pronounced

upward or downward trend but do show seasonal variation, and that these two series seem to move together. Additional inspection also shows that the seasonal effect is relatively stable throughout the data span. These considerations suggest the following model for $y = (f_KSI, r_KSI)$:

$$y_t = \begin{pmatrix} X_t \\ 0 \end{pmatrix} \beta + \mu_t + \zeta_t + \xi_t$$

All the terms on the right-hand side of this equation are assumed to be statistically independent. These terms are as follows:

- The predictor X_t (defined as Q1_83_Shift later in the program) denotes a variable that is 0 before the first quarter of 1983, and 1 thereafter. X_t is supposed to affect only f_KSI (the first element of y); it represents the enactment of the seat-belt law of 1983.
- μ_t denotes a bivariate random walk. It is supposed to capture the slowly changing level of the vector y_t . To capture the fact that f_KSI and r_KSI move together (that is, they are co-integrated), the covariance of the disturbance term of this random walk is assumed to be of lower than full rank.
- ζ_t denotes a bivariate trigonometric seasonal term. In this model, it is taken to be fixed (that is, the seasonal effects do not change over time).
- ξ_t denotes a bivariate white noise term, which captures the residual variation that is unexplained by the other terms in the model.

The preceding model is an example of a (bivariate) basic structural model (BSM). The following statements load the seatBelt data into the table mylib.seatBelt:

```
data mylib.seatBelt;
  set seatBelt;
run;
```

The following statements specify and fit this model to f_KSI and r_KSI :

```
proc cssm data=mylib.seatBelt stateinfo;
  id date interval=quarter;
  Q1_83_Shift = (date >= '1jan1983'd);
  state error(2) type=WN cov(g) print=cov;
  component wn1 = error[1];
  component wn2 = error[2];
  state level(2) type=RW cov(rank=1) print=cov;
  component rw1 = level[1];
  component rw2 = level[2];
  state season(2) type=season(length=4);
  component s1 = season[1];
  component s2 = season[2];
  model f_KSI = Q1_83_Shift rw1 s1 wn1 / print=(smooth);
  model r_KSI = rw2 s2 wn2;
  eval f_KSI_sa = rw1 + Q1_83_Shift;
  output out=mylib.For1;
run;
```

The PROC CSSM statement specifies the input table, `mylib.seatBelt`. The use of the STATEINFO option in the PROC CSSM statement produces additional information about the model state vector and its diffuse initial state. The optional ID statement specifies an index variable, `date`. The INTERVAL=QUARTER option in the ID statement indicates that the measurements were collected on a quarterly basis. Next, a programming statement defines `Q1_83_Shift`, the predictor that represents the enactment of the seat-belt law of 1983. It is used later in the MODEL statement for `f_KSI`. Separate STATE statements specify the terms μ_t , ξ_t , and ζ_t because they are statistically independent. Each model that governs them (white noise for ξ_t , random walk for μ_t , and trigonometric seasonal for ζ_t) can be specified by using the TYPE= option of the STATE statement. When you use the TYPE= option, you can use the COV option to specify the information about the disturbance covariance in the state transition equation. The other details, such as the transition matrix specification and the specification of A_1 in the initial condition, are inferred from the TYPE= option. The use of PRINT=COV in the STATE statement causes the estimated disturbance covariance to be printed. For ξ_t (a white noise), A_1 is zero and $Q_t = Q$ for all $t \geq 1$, where Q is specified by the COV option. For μ_t and ζ_t the initial condition is fully diffuse—that is, A_1 is an identity matrix of appropriate order and $Q_1 = 0$. The total diffuse dimension of this model, $(d + k)$, is $9 = 8 + 1$ as a result of one predictor, `Q1_83_Shift`, and two fully diffuse state subsections, μ_t and ζ_t . The components in the model are defined by suitable linear combinations of these different state subsections. The program statements define the model as follows:

- **state error(2) type=WN cov(g);** defines ξ_t as a two-dimensional white noise, named `error`, with the covariance of general form. Then two COMPONENT statements define `wn1` and `wn2` as the first and second elements of `error`, respectively.
- **state level(2) type=RW cov(rank=1);** defines μ_t as a two-dimensional random walk, named `level`, with covariance of general form whose rank is restricted to 1. Then two COMPONENT statements define `rw1` and `rw2` as the first and second elements of `level`, respectively.
- **state season(2) type=season(length=4);** defines ζ_t as a two-dimensional trigonometric seasonal of season length 4, named `season`, with zero covariance—signified by the absence of the COV option. Then two COMPONENT statements define `s1` and `s2` as appropriate linear combinations of `season` so that `s1` represents the seasonal for `f_KSI` and `s2` represents the seasonal for `r_KSI`. Because TYPE=SEASON in the STATE statement, the COMPONENT statement appropriately interprets `component s1 = season[1];` as `s1` being a dot product: $(1\ 0\ 0\ 0\ 1\ 0) * \text{season}$. For more information, see the section “Multivariate Season” on page 783.
- **model f_KSI = Q1_83_Shift rw1 s1 wn1;** defines the model for `f_KSI`, and **model r_KSI = rw2 s2 wn2;** defines the model for `r_KSI`.

The CSSM procedure fits the model and reports the parameter estimates, their approximate standard errors, and the likelihood-based goodness-of-fit measures by default. In order to output the one-step-ahead and full-sample estimates of the components in the model, you can either use the PRINT= options in the MODEL statement and the respective COMPONENT statements or you can specify an output data set in the OUTPUT statement. In addition, you can use the EVAL statement to define specific linear combinations of the underlying state that should also be estimated. The statement `eval f_KSI_sa = rw1 + Q1_83_Shift;` is an example of one such linear combination. It defines `f_KSI_sa`, a linear combination that represents the seasonal adjustment of `f_KSI`. The output data set, `mylib.For1` (named in the OUTPUT statement) contains estimates of all the model components in addition to the estimate of `f_KSI_sa`.

The model summary table, shown in [Output 14.1.1](#), provides basic model information, such as the dimension of the underlying state equation ($m = 10$), the diffuse dimension of the model ($(d + k) = 9$), and the number of parameters (5) in the model parameter vector θ .

Output 14.1.1 Bivariate Basic Structural Model

Model Summary	
Number of Model Equations	2
State Dimension	10
Dimension of the Diffuse Initial Condition	9
Number of Parameters	5

Additional details about the role of different components in forming the model state and its diffuse initial condition are shown in [Output 14.1.2](#) and [Output 14.1.3](#). They show that the 10-dimensional model state vector is made up of subsections that are associated with error and level (each of dimension 2) and season (of dimension 6). Similarly, the nine-dimensional diffuse vector in the initial condition is made up of subsections that correspond to level, season, and the regression variable, Q1_83_Shift. Note that error does not contribute to the diffuse initial vector because it has a fully nondiffuse initial state.

Output 14.1.2 Bivariate Basic Structural Model State Vector Summary

State Vector Composition	
Subsection	Dimension
error	2
level	2
season	6

Output 14.1.3 Bivariate Basic Structural Model Initial Diffuse State Vector Summary

Diffuse Initial State Composition	
Subsection	Dimension
level	2
season	6
Q1_83_Shift	1

The index variable information is shown in [Output 14.1.4](#).

Output 14.1.4 Index Variable Information

ID Variable Information					
Name	Start	End	Maximum Distinct		Type
			Delta	Values	
date	1969/1	1985/4	1.0000	68	Regular

[Output 14.1.5](#) provides simple summary information about the response variables. It shows that f_KSI and r_KSI have four missing values each and no induced missing values because the predictor in the model, Q1_83_Shift, has no missing values.

Output 14.1.5 Response Variable Summary

Response Variable Information							
		Number of Observations		Induced Missing		Standard Deviation	
Name	Total	Missing	Missing	Minimum	Maximum	Mean	Deviation
f_KSI	68	4	0	6.164	7.093	6.707	0.2062
r_KSI	68	4	0	5.556	6.405	5.973	0.1862

The regression coefficient of Q1_83_Shift, shown in [Output 14.1.6](#), is negative and is statistically significant. This is consistent with the expected drop in f_KSI after the enactment of the seat-belt law.

Output 14.1.6 Regression Coefficient of Q1_83_Shift

Regression Parameter Estimates					
		Standard			
Response	Regressor	Estimate	Error	t Value	Pr > t
f_KSI	Q1_83_Shift	-0.4084	0.0259	-15.74	<.0001

[Output 14.1.7](#) shows the estimates of the elements of θ . The five parameters in θ correspond to unknown elements that are associated with the covariance matrices in the specifications of error and level. Whenever a covariance specification is of a general form and is not defined by a user-specified variable list, it is internally parameterized as a product of its Cholesky root: $\text{Cov} = \text{Root Root}^T$. This ensures that the resulting covariance is positive semidefinite. The Cholesky root is constrained to be lower triangular, with positive diagonal elements. If rank constraints (such as the rank-one constraint on the covariance in the specification of level) are imposed, the number of free parameters in the Cholesky factor is reduced appropriately. For more information, see the section “[Covariance Parameterization](#)” on page 793. In view of these considerations, the five parameters in θ are a result of three parameters from the Cholesky root of error and two parameters that are associated with the Cholesky root of level.

Output 14.1.7 Parameter Estimates

Model Parameter Estimates					
				Standard	
Component	Type	Parameter	Estimate	Error	t Value
error	Disturbance Covariance	RootCov[1, 1]	0.0361	0.0074	4.91
error	Disturbance Covariance	RootCov[2, 1]	0.0338	0.0113	2.99
error	Disturbance Covariance	RootCov[2, 2]	0.0462	0.0047	9.84
level	Disturbance Covariance	RootCov[1, 1]	0.0375	0.0084	4.45
level	Disturbance Covariance	RootCov[2, 1]	0.0223	0.0057	3.92

[Output 14.1.8](#) shows the resulting covariance estimate of error after multiplying the Cholesky factors.

Output 14.1.8 White Noise Covariance Estimate

Disturbance Covariance for error		
	Col1	Col2
Row1	0.00131	0.00122
Row2	0.00122	0.00328

Similarly, [Output 14.1.9](#) shows the covariance estimate of level disturbance. Note that because of the rank-one constraint, the determinant of this matrix is 0.

Output 14.1.9 Covariance Estimate of the Random Walk Disturbance

Disturbance Covariance for level		
	Col1	Col2
Row1	0.00141	0.000837
Row2	0.000837	0.000497

[Output 14.1.10](#) shows the likelihood computation summary. This table is produced by using the fitted model to carry out the filtering operation on the data. For more information, see the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769.

Output 14.1.10 Likelihood Computation Summary of the Fitted Model

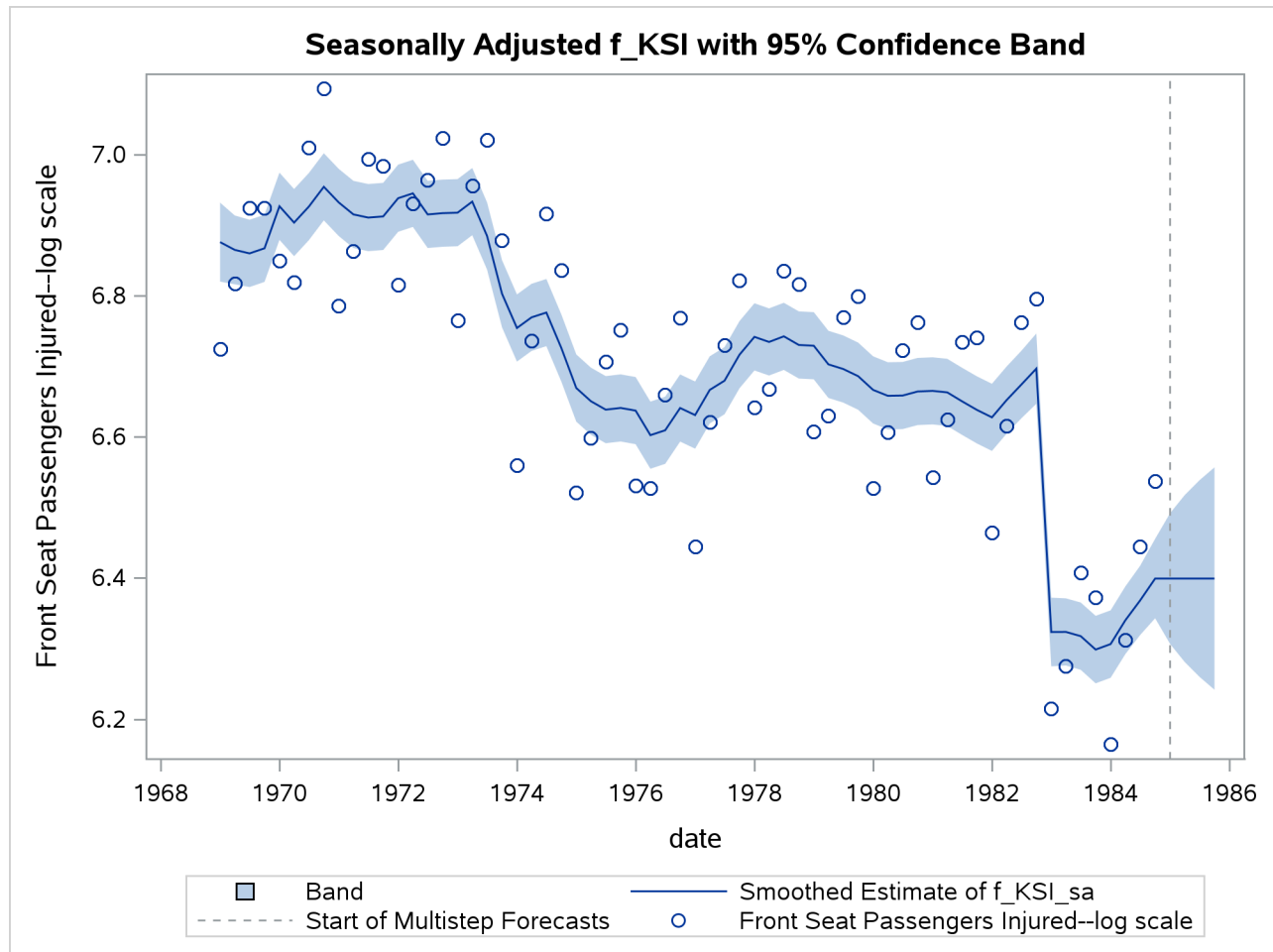
Likelihood Computation Summary	
Nonmissing Response Values Used	128
Estimated Parameters	5
Initialized Diffuse State Elements	9
Normalized Residual Sum of Squares	119.0000
Diffuse Log Likelihood	166.1575
Profile Log Likelihood	199.9116

The output data set, `mylib.For1`, specified in the `OUTPUT` statement contains one-step-ahead and full-sample estimates of all the model components and the user-specified components (defined by the `EVAL` statement). Their standard errors and the upper and lower confidence limits (by default, 95%) are also produced.

The following statements use the `mylib.For1` data set to produce a time series plot of the seasonally adjusted `f_KSI`:

```
proc sgplot data=mylib.For1;
  title "Seasonally Adjusted f_KSI with 95% Confidence Band";
  band x=date lower=smoothed_lower_f_KSI_sa
    upper=smoothed_upper_f_KSI_sa ;
  series x=date y=smoothed_f_KSI_sa;
  refline '1jan1985'd / axis=x lineattrs=(pattern=shortdash)
    LEGENDLABEL= "Start of Multistep Forecasts"
    name="Forecast Reference Line";
  scatter x=date y=f_KSI ;
run;
```

The generated plot is shown in [Output 14.1.11](#).

Output 14.1.11 Plot of Seasonally Adjusted f_KSI 

Example 14.2: Random-Effects and Autoregressive Models for Panel Data

This example shows how you can use the CSSM procedure to specify and fit the two-way random-effects model and the autoregressive model to analyze a panel of time series. The fitting of dynamic panel model for such data is illustrated in “[Example 14.11: Dynamic Panel Model for the Cigar Data](#)” on page 859. These (and a few other) model types can also be fitted by the PANEL procedure, a SAS/ETS procedure that is specially designed to efficiently handle the cross-sectional time series data. However, because of the differences in their model fitting algorithms, generally the parameter estimates and other fit statistics produced by the CSSM and PANEL procedures do not match. The CSSM procedure always uses the (restricted) maximum likelihood for parameter estimation. The estimation method used by the PANEL procedure depends on the model type and the particular estimation options.

The cross-sectional data, Cigar, that are used in the section “[Getting Started: CSSM Procedure](#)” on page 722 are reused in this example. The output shown here is less extensive than the output shown in that section. The main emphasis of this example is how you can specify the two-way random effects model and the autoregressive model in the CSSM procedure.

According to the two-way random effects model, the cigarette sales, $lsales$, can be described by the following equation:

$$lsales_{i,t} = \mu + lprice \beta_1 + lndi \beta_2 + lpimin \beta_3 + \zeta_i + \eta_t + \epsilon_{i,t}$$

This model represents $lsales$ in region i and in year t as a sum of an overall intercept μ , the regression effects due to $lprice$, $lndi$, and $lpimin$, a zero-mean, random effect ζ_i associated with region i , a zero-mean, random effect η_t associated with year t , and the observation noise $\epsilon_{i,t}$. The region-specific random effects ζ_i and the year-specific random effects η_t are assumed to be independent, Gaussian sequences with variances σ_ζ^2 and σ_η^2 , respectively. In addition, they are assumed to be independent of the observation noise, which is also assumed to be a sequence of independent, zero-mean, Gaussian variables with variance σ_ϵ^2 .

The following statements load the Cigar data into the table `mylib.Cigar`:

```
data mylib.Cigar;
  set Cigar;
run;
```

You can specify and fit this model by using the following statements:

```
proc cssm data=mylib.Cigar;
  id year interval=year;
  obsindex region;
  parms s2g/ lower=(1.e-6);
  array RegionArray{46} region1-region46;
  do i=1 to 46;
    RegionArray[i] = (region=i);
  end;
  /* region-specific random effects */
  state zeta(46) T(I) cov1(I)=(s2g);
  component regionEffect = zeta * (RegionArray);
  /* year-specific random effect */
  state eta(1) type=wn cov(D);
  component timeEffect = eta[1];
  irregular wn;
  intercept = 1.0;
  model lsales = intercept lprice lndi lpimin
    timeEffect regionEffect wn;
run;
```

The `PARMS` statement defines `s2g`, a parameter that is restricted to be positive and is used later as the variance parameter for the region effect. Similarly the 46-dimensional array, `RegionArray`, of region-specific dummy variables is defined to be used later. The state subsection `zeta` corresponds to ζ , which is the 46-dimensional vector of region-specific, zero-mean, random effects. The component `regionEffect` extracts the proper element of ζ by using the array `RegionArray`. A constant column, `intercept`, is defined to be used later as an intercept term. The component `timeEffect` corresponds to η_t , and `wn` specifies the observation noise ϵ_{it} . Finally the `MODEL` statement defines the model. Some of the tables that are produced by running these statements are shown in [Output 14.2.1](#) through [Output 14.2.5](#).

The model summary, shown in [Output 14.2.1](#), shows that the model is defined by one `MODEL` statement, the dimension of the underlying state vector is 47 (because ζ is 46-dimensional and η_t is one-dimensional), the diffuse dimension is 4 (because of the four predictors in the model), and there are three parameters to be estimated.

Output 14.2.1 Two-Way Random-Effects Model: Model Summary

Model Summary	
Number of Model Equations	1
State Dimension	47
Dimension of the Diffuse Initial Condition	4
Number of Parameters	3

Output 14.2.2 provides the likelihood information about the fitted model.

Output 14.2.2 Two-Way Random-Effects Model: Likelihood Summary

Likelihood Computation Summary	
Nonmissing Response Values Used	1380
Estimated Parameters	3
Initialized Diffuse State Elements	4
Normalized Residual Sum of Squares	1376
Diffuse Log Likelihood	1459
Profile Log Likelihood	1471

Output 14.2.3 shows the regression estimates.

Output 14.2.3 Two-Way Random-Effects Model: Regression Estimates

Regression Parameter Estimates					
		Standard			
Response	Regressor	Estimate	Error	t Value	Pr > t
Isales	intercept	2.7976	0.1136	24.62	<.0001
Isales	lprice	-0.9029	0.0365	-24.73	<.0001
Isales	Indi	0.5922	0.0246	24.08	<.0001
Isales	lpimin	0.1266	0.0398	3.18	0.0015

The ML estimate of s2g, a parameter specified in the PARMS statement, is shown in Output 14.2.4. It corresponds to σ_{ξ}^2 , the variance of the region effect.

Output 14.2.4 Two-Way Random-Effects Model: Estimate of σ_{ξ}^2

Estimates of Named Parameters			
		Standard	
Parameter	Estimate	Error	t Value
s2g	0.0241	0.0051	4.70

Output 14.2.5 Variance Estimates of η_t and ϵ_{it}

Model Parameter Estimates					
		Standard			
Component	Type	Parameter	Estimate	Error	t Value
eta	Disturbance Covariance	Cov[1, 1]	0.0007	0.0003	2.58
wn	Irregular	Variance	0.0057	0.0002	25.40

The estimates of the other unknown parameters in the model are shown in [Output 14.2.5](#). It shows the estimate of the variance of the irregular component ω and the estimate of the variance of the time effect η_t .

The remainder of this example describes how you can specify and fit the following first-order vector autoregressive model to the cigarette data:

$$\begin{aligned} \text{lsales}_{i,t} &= \mu + \text{lprice} \beta_1 + \text{ln di} \beta_2 + \text{lpimin} \beta_3 + \zeta_t[i] \\ \zeta_t &= \Phi \zeta_{t-1} + \eta_t \end{aligned}$$

This model represents lsales in region i and in year t as a sum of an overall intercept μ , the regression effects due to lprice , ln di , and lpimin , and the i th element of a vector error term $\zeta_t[i]$. The multidimensional error sequence ζ_t is assumed to follow a first-order autoregression with a diagonal autoregressive coefficient matrix Φ and with a multivariate, white noise sequence η_t as its disturbance sequence. The covariance matrix of η_t , Σ , is assumed to be dense. Note that the dimension of the vectors ζ_t is the same as the number of cross sections in the study (the number of regions in this example). Therefore, even for a relatively modest panel study, the total number of parameters to be estimated can get quite large. Therefore, in this example only the first three regions are considered in the analysis. The following statements specify and fit this model to the Cigar data set:

```
proc cssm data=mylib.Cigar;
  where region <= 3;
  id year interval=year;
  obsindex region;
  array RegionArray{3} region1-region3;
  do i=1 to 3;
    RegionArray[i] = (region=i);
  end;
  state zeta(3) type=varma(p(d)=1) cov(g) print=(ar cov);
  component eta = zeta*(RegionArray);
  intercept = 1.0;
  model lsales = intercept lprice ln di lpimin eta;
run;
```

The vectors ζ_t are specified in the STATE statement. The TYPE= specification signifies that the three-dimensional state subsection, zeta , follows a vector AR(1) model with a diagonal transition matrix and a disturbance covariance of a general form. The PRINT=(AR COV) option causes the CSSM procedure to print the estimated AR coefficient matrix, Φ , and the disturbance error covariance Σ , respectively. The COMPONENT statement defines the appropriate error contribution (named eta), $\zeta_t[i]$. [Output 14.2.6](#) shows the estimated regression coefficients, [Output 14.2.7](#) shows the estimate of Φ , and [Output 14.2.8](#) shows the estimate of Σ :

Output 14.2.6 Autoregressive Model: Regression Estimates

Regression Parameter Estimates					
Response	Regressor	Estimate	Standard Error	t Value	Pr > t
lsales	intercept	3.6857	0.3961	9.31	<.0001
lsales	lprice	-0.2356	0.0833	-2.83	0.0047
lsales	ln di	0.1969	0.0774	2.54	0.0110
lsales	lpimin	0.0737	0.0995	0.74	0.4588

Output 14.2.7 Estimate of the AR Coefficient Φ

AR Coefficient for zeta			
	Col1	Col2	Col3
Row1	0.9257	0	0
Row2	0	0.9840	0
Row3	0	0	0.9601

Output 14.2.8 Estimate of the Disturbance Covariance Σ

Disturbance Covariance for zeta			
	Col1	Col2	Col3
Row1	0.000911	0.000342	0.000361
Row2	0.000342	0.00222	0.000172
Row3	0.000361	0.000172	0.000923

Example 14.3: Backcasting, Forecasting, and Interpolation

This example illustrates how you can do model-based extrapolation—backcasting, forecasting, or interpolation—of a response variable. All you need is to appropriately augment the input data set with the relevant ID and predictor information and assign missing values to the response variable in these places. The following DATA step creates one such augmented data set by using a well-known data set that contains recordings of the Nile River water level measured between the years 1871 and 1970. Suppose you want to backcast the Nile water level for two years before 1871, forecast it for two years after 1970, and interpolate its value for the year 1921—for illustration purposes, this value is assumed to be missing in the available data set.

```
data Nile;
    input level @@;
    year = intnx( 'year', '1jan1869'd, _n_-1 );
    format year year4.;
    if year = '1jan1921'd then level=.;
datalines;
. .
1120 1160 963 1210 1160 1160 813 1230 1370 1140
995 935 1110 994 1020 960 1180 799 958 1140
1100 1210 1150 1250 1260 1220 1030 1100 774 840
874 694 940 833 701 916 692 1020 1050 969
831 726 456 824 702 1120 1100 832 764 821
768 845 864 862 698 845 744 796 1040 759
781 865 845 944 984 897 822 1010 771 676
649 846 812 742 801 1040 860 874 848 890
744 749 838 1050 918 986 797 923 975 815
1020 906 901 1170 912 746 919 718 714 740
. .
;
```

```
data mylib.Nile;
  set Nile;
run;
```

It is also known that for this time span the Nile water level can be reasonably modeled as a sum of a random walk trend, a level shift in the year 1899, and the observation error. The following statements fit this model to the data:

```
proc cssm data=mylib.Nile;
  id year interval=year;
  shift1899 = ( year >= '1jan1899'd );
  trend rw(rw);
  irregular wn;
  model level = shift1899 RW wn / print=smooth;
  output out=mylib.nileOut;
quit;
```

The model-based interpolated and extrapolated values of the Nile water level are shown in [Output 14.3.1](#), which is produced by using the PRINT=SMOOTH option in the MODEL statement.

Output 14.3.1 Interpolated and Extrapolated Nile Water Level

Full-Sample Prediction of Missing Values for level				
ID	Estimate	Standard Error	95% Confidence Limits	
1869	1097.7	130.2	842.5	1353
1870	1097.7	130.2	842.5	1353
1921	851.1	128.9	598.6	1104
1971	851.1	128.9	598.6	1104
1972	851.1	128.9	598.6	1104

Example 14.4: Smoothing of Repeated Measures for Longitudinal Data

This example of a repeated measures study is taken from Diggle, Liang, and Zeger (1994, p. 100). The data consist of body weights of 27 cows, measured at 23 unequally spaced time points over a period of approximately 22 months. Following Diggle, Liang, and Zeger (1994), one animal is removed from the analysis, one observation is removed according to their Figure 5.7, and the time is shifted to start at 0 and is measured in 10-day increments. The design is a 2×2 factorial, and the factors are the infection of an animal with *M. paratuberculosis* and whether the animal is receiving iron dosing. The data set contains five variables: cow assigns a unique identification number—from 1 to 26—to each cow in the study, tpoint denotes the time of the growth measurement, weight denotes the growth measurement, iron is a dummy variable that indicates whether the animal is receiving iron or not, and infection is a dummy variable that indicates whether the animal is infected or not. The goal of the study is to assess the effect of iron and infection—and their possible interaction—on weight. The following DATA steps create this data set:


```

data times;
  input time1-time23;
  datalines;
122 150 166 179 219 247 276 296 324 354 380 445
478 508 536 569 599 627 655 668 723 751 781
;

data Cows;
  if _n_ = 1 then merge times;
  array t{23} time1 - time23;
  array w{23} weight1 - weight23;
  input cow iron infection weight1-weight23 @@;
  do i=1 to 23;
    weight = w{i};
    tpoint = (t{i}-t{1})/10;
    output;
  end;
  keep cow iron infection tpoint weight;
  datalines;
1 0 0 4.7 4.905 5.011 5.075 5.136 5.165 5.298 5.323
      5.416 5.438 5.541 5.652 5.687 5.737 5.814 5.799
... more lines ...

```

The following DATA step adds ironInf, a grouping variable that is used later during the plotting of the results. In the next step, the data are sorted by the index variable, tpoint.

```

data Cows;
  set Cows;
  ironInf = "No Iron and No Infection";
  if iron=1 and infection=1 then ironInf = "Iron and Infection";
  else if iron=1 and infection=0 then ironInf = "Iron and No Infection";
  else if iron=0 and infection=1 then ironInf = "No Iron and Infection";
  else ironInf = "No Iron and No Infection";
  run;

proc sort data=Cows;
  by tpoint ;
run;

```

The following statements load the Cows data into the table mylib.Cows:

```

data mylib.Cows;
  set Cows;
  obsNo = _n_;
run;

```

To assess the effect of iron and infection on weight, the natural growth profile of the animals must also be accounted for. Here two alternate models for this problem are considered. The first model assumes that the observed weight of an animal is the sum of a common growth profile, which is modeled by a polynomial spline trend of order 2, the regression effects of iron and infection, and the observation error—modeled as white noise. An interaction term, for interaction between iron and infection, was found to be insignificant and is not included. In the second model, the common growth profile and the regression variables of the first

model are replaced by four environment specific growth profiles.

The following statements fit the first model:

```
proc cssm data=mylib.Cows;
  id tpoint;
  obsindex obsNo;
  trend growth(ps(2));
  irregular wn;
  model weight = iron infection growth wn;
  eval pattern = iron + infection + growth;
  output out=mylib.For;
quit;
```

Output 14.4.1 shows that the state dimension of this model is 2 (corresponding to the polynomial trend specification of order 2), the number of diffuse elements in the initial condition is 4 (corresponding to the trend and the two regressors iron and infection), and the number of unknown parameters is 2 (corresponding to the variance parameters of trend and irregular).

Output 14.4.1 Model1: Model Summary Information

Model Summary	
Number of Model Equations	1
State Dimension	2
Dimension of the Diffuse Initial Condition	4
Number of Parameters	2

Output 14.4.2 shows that the ID variable is irregularly spaced with replication.

Output 14.4.2 ID Variable Information

ID Variable Information					
Name	Start	End	Maximum Distinct		
			Delta	Values	Type
tpoint	0	65.900000	6.5000	23	Irregular with Replication

The estimated regression coefficients of iron and infection, shown in Output 14.4.3, are significant and negative. This implies that both iron and infection adversely affect the response variable, weight.

Output 14.4.3 Model 1: Regression Estimates

Regression Parameter Estimates					
Response	Regressor	Estimate	Standard		
			Error	t Value	Pr > t
weight	iron	-0.0748	0.0076	-9.82	<.0001
weight	infection	-0.1292	0.0086	-15.04	<.0001

The variance estimates of the trend component and the irregular component are shown in Output 14.4.4.

Output 14.4.4 Model 1: Estimates of Unnamed Parameters

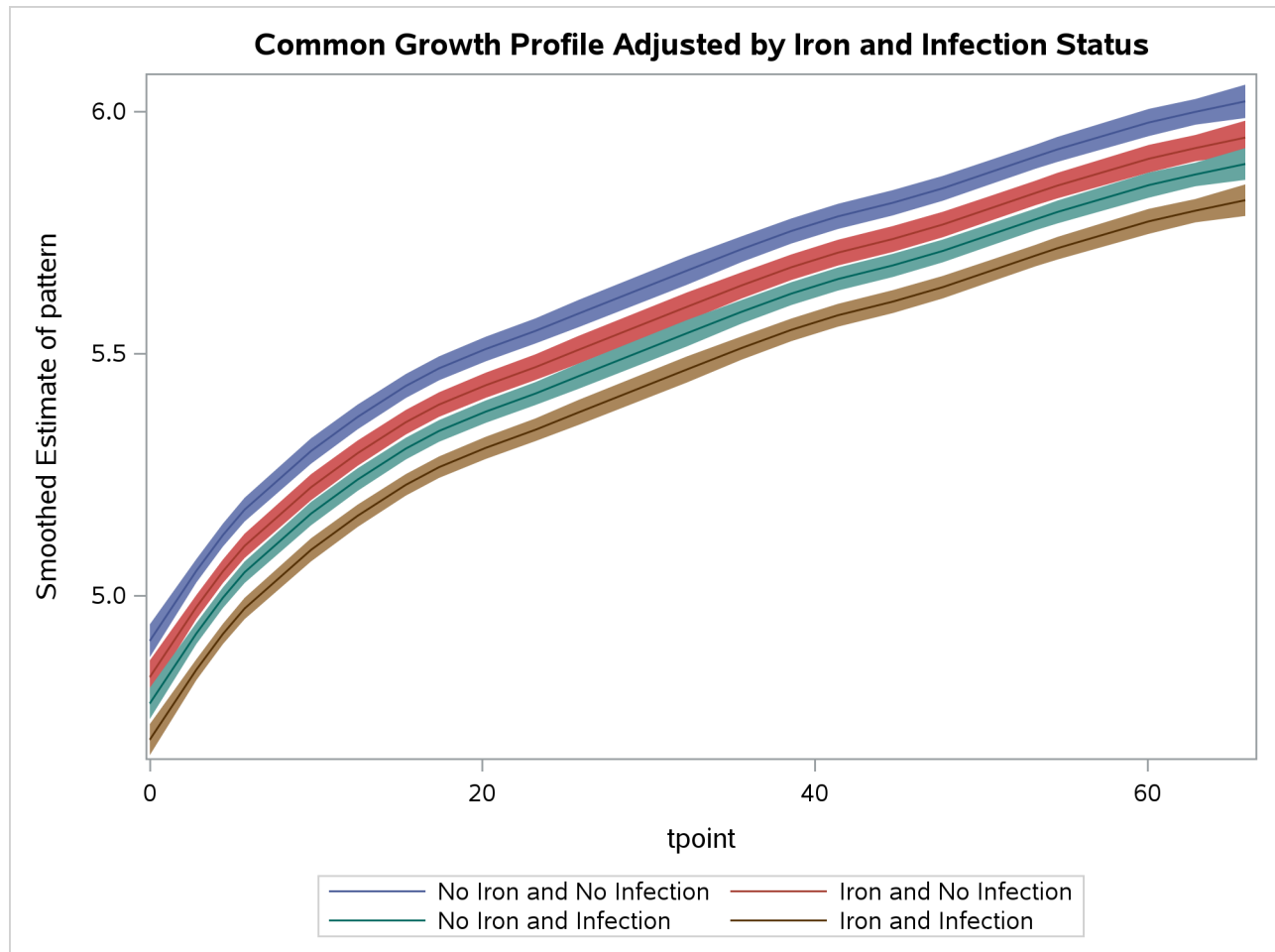
Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
growth	PS(2) Trend	Level Variance	0.0000	0.0000	1.80
wn	Irregular	Variance	0.0086	0.0005	17.06

After examining the model fit, it is useful to study how well the patterns implied by the model follow the data. `pattern`, defined by the `EVAL` statement, is a sum of the trend component and the regression effects. A graphical examination of the smoothed estimate of `pattern` is done next. The following `DATA` step merges the output data set specified in the `OUTPUT` statement, `mylib.For`, with the input data set, `mylib.Cows`. In particular, this adds `ironInf` (a grouping variable from `mylib.Cows`) to `mylib.For`.

```
data mylib.For;
  merge mylib.for mylib.Cows;
  by tpoint obsNo;
run;
```

The following statements produce the graphs of `smoothed_pattern`, grouped according to the environment condition (see [Output 14.4.5](#)). The plot clearly shows that the control group “No Iron and No Infection” has the best growth profile, while the worst growth profile is for the group “Iron and Infection.”

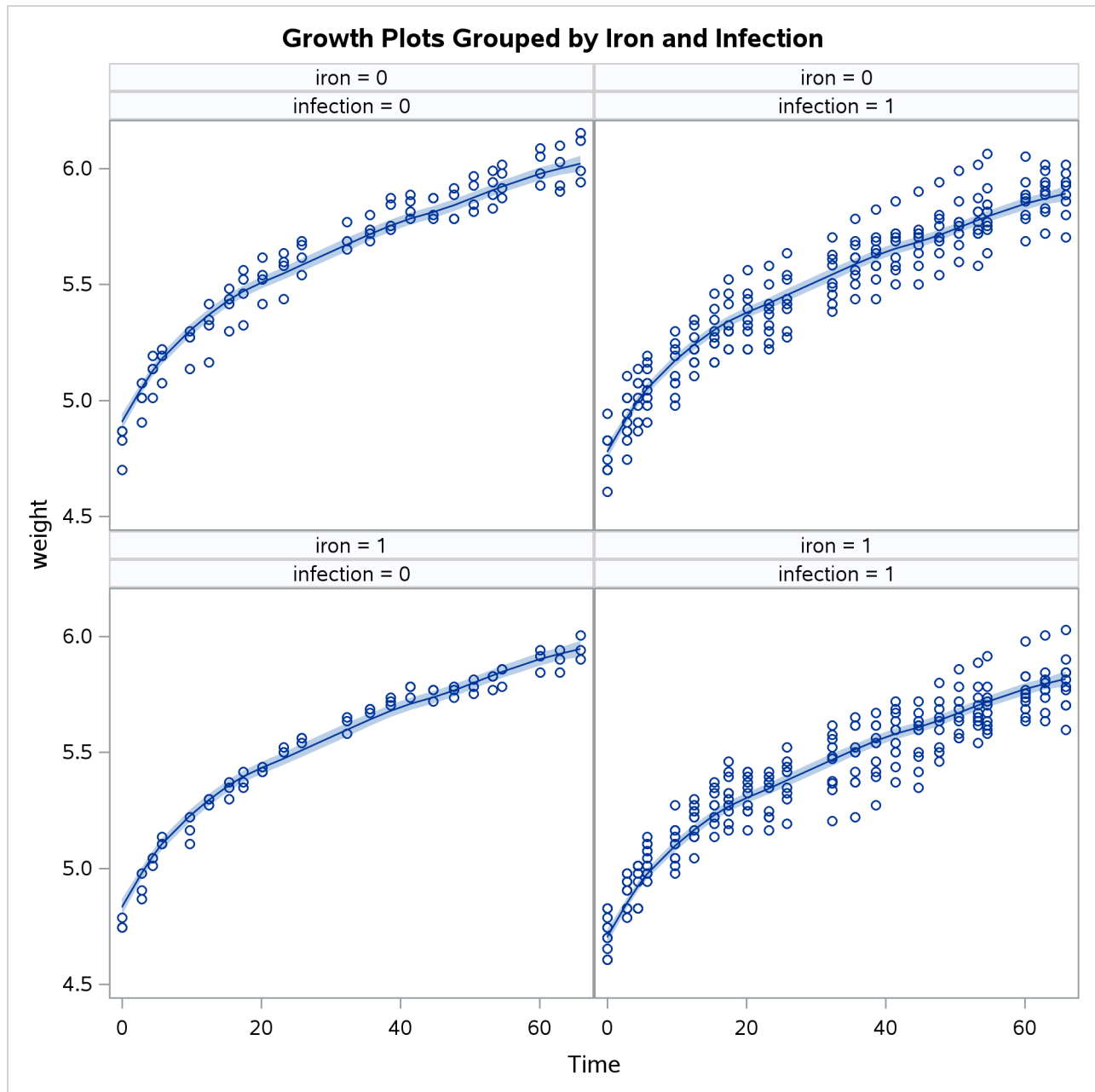
```
proc sgplot data=mylib.For noautolegend;
  title 'Common Growth Profile Adjusted by Iron and Infection Status';
  band x=tpoint lower=smoothed_lower_pattern
      upper=smoothed_upper_pattern / group=ironInf name="band";
  series x=tpoint y=smoothed_pattern / group=ironInf name="series";
  keylegend "series";
run;
```

Output 14.4.5 Model 1: Growth Profile Comparison with 95% Confidence Bands

The following statements produce a panel of plots that show how well `smoothed_pattern` follows the observed data:

```
proc sgpanel data=mylib.For noautolegend;
  title 'Growth Plots Grouped by Iron and Infection';
  label tpoint='Time' ;
  panelby iron infection / columns=2;
  band x=tpoint lower=smoothed_lower_pattern
        upper=smoothed_upper_pattern ;
  scatter x=tpoint y=weight;
  series x=tpoint y=smoothed_pattern ;
run;
```

Output 14.4.6 shows that the model fits the data reasonably well.

Output 14.4.6 Model 1: Smoothed Model Fit Lines

The following statements fit the second model. In this model separate polynomial trends are fit according to different settings of iron and infection by specifying an appropriate list of (dummy) variables in the **CROSS=** option of the trend specification.

```
proc cssm data=mylib.Cows;
  id tpoint;
  obsindex obsNo;
  a1 = (iron=1 and infection=1);
  a2 = (iron=1 and infection=0);
  a3 = (iron=0 and infection=1);
```

```

a4 = (iron=0 and infection=0);
trend growth(ps(2)) cross=(a1-a4);
irregular wn;
model weight = growth wn;
/* Define contrasts between a1 and other treatments */
comp a1Curve = growth_state_[1];
comp a2Curve = growth_state_[2];
comp a3Curve = growth_state_[3];
comp a4Curve = growth_state_[4];
eval contrast21 = a2Curve - a1Curve;
eval contrast31 = a3Curve - a1Curve;
eval contrast41 = a4Curve - a1Curve;
output out=mylib.for1;
quit;

```

As a result of the **CROSS=** option, the trend component growth is actually a sum of four separate trends that correspond to the different iron-infection settings. Denoting growth by μ_t and the four independent trends by $\mu_{1,t}$, $\mu_{2,t}$, $\mu_{3,t}$, and $\mu_{4,t}$,

$$\mu_t = a1 * \mu_{1,t} + a2 * \mu_{2,t} + a3 * \mu_{3,t} + a4 * \mu_{4,t}$$

where a1, a2, a3, and a4 are the dummy variables specified in the **CROSS=** option. This shows that, for any given setting (say, the one for a4) μ_t is simply the corresponding trend $\mu_{4,t}$. In addition, note the form of the **COMPONENT** statements that define the components a1Curve, a2Curve, a3Curve, and a4Curve. This form of the **COMPONENT** statement treats the state that is associated with growth, named growth_state_ by convention, as a state of nominal dimension 4—the number of variables in the **CROSS=** list. This, in turn, implies that a1Curve, which is defined as growth_state_[1], refers to $\mu_{1,t}$. These components are subsequently used in the **EVAL** statements to define contrasts between the trends—for example, contrast21 corresponds to the difference between the trends $\mu_{2,t}$ and $\mu_{1,t}$. The estimates of these components (a1Curve, a2Curve, ..., contrast41) are output to the data set mylib.For1 named in the **OUT=** option of the **OUTPUT** data set.

The model summary, shown in [Output 14.4.7](#), reflects the increased state dimension and the increased number of parameters.

Output 14.4.7 Model2: Model Summary Information

Model Summary	
Number of Model Equations	1
State Dimension	8
Dimension of the Diffuse Initial Condition	8
Number of Parameters	5

[Output 14.4.8](#) shows the parameter estimates for this model.

Output 14.4.8 Model2: Estimates of Unnamed Parameters (Partial Output)

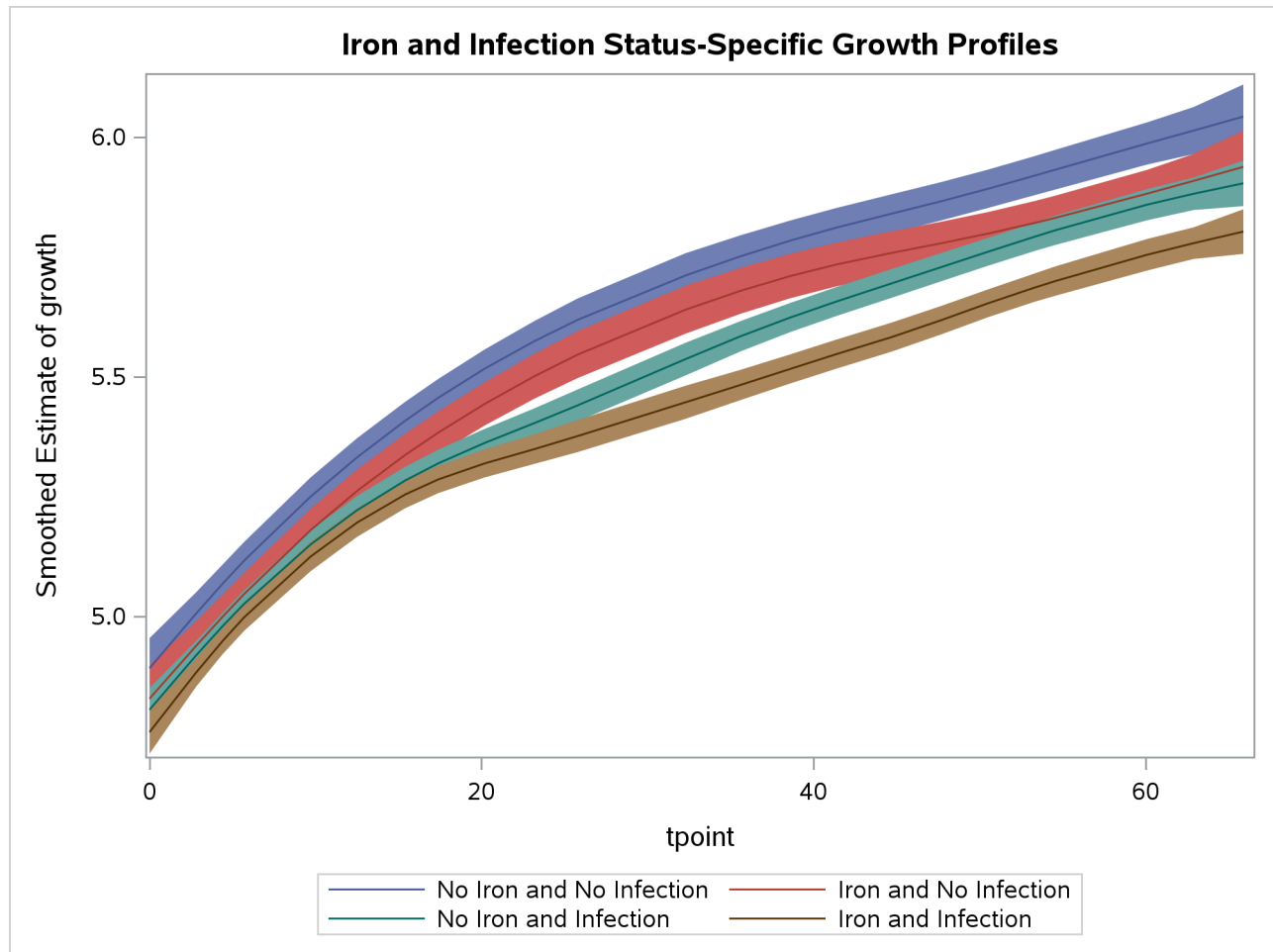
Component	Type	Parameter	Estimate	StdErr	tValue
growth(Cross = a1)	PS(2) Trend	Level Variance	0.0000	0.0000	1.87
growth(Cross = a2)	PS(2) Trend	Level Variance	0.0000	0.0000	2.29
growth(Cross = a3)	PS(2) Trend	Level Variance	0.0000	0.0000	2.14
growth(Cross = a4)	PS(2) Trend	Level Variance	0.0000	0.0000	2.49
wn	Irregular	Variance	0.0084	0.0005	16.84

Next, the smoothed estimate of trend (growth) is graphically studied. The following DATA step prepares the data for the grouped plots of smoothed_growth by merging mylib.For1 with the input data set Cows. As before, the reason is merely to include ironInf (the grouping variable).

```
data mylib.For1;
  merge mylib.For1 mylib.Cows;
  by tpoint obsNo;
run;
```

The following statements produce the graphs of smoothed μ_t for the desired settings (since the grouping variable ironInf exactly corresponds to these settings). Once again, the plot in [Output 14.4.9](#) clearly shows that the control group “No Iron and No Infection” has the best growth profile, while the worst growth profile is for the group “Iron and Infection.” However, unlike the first model, the profile curves are not merely shifted versions of a common profile.

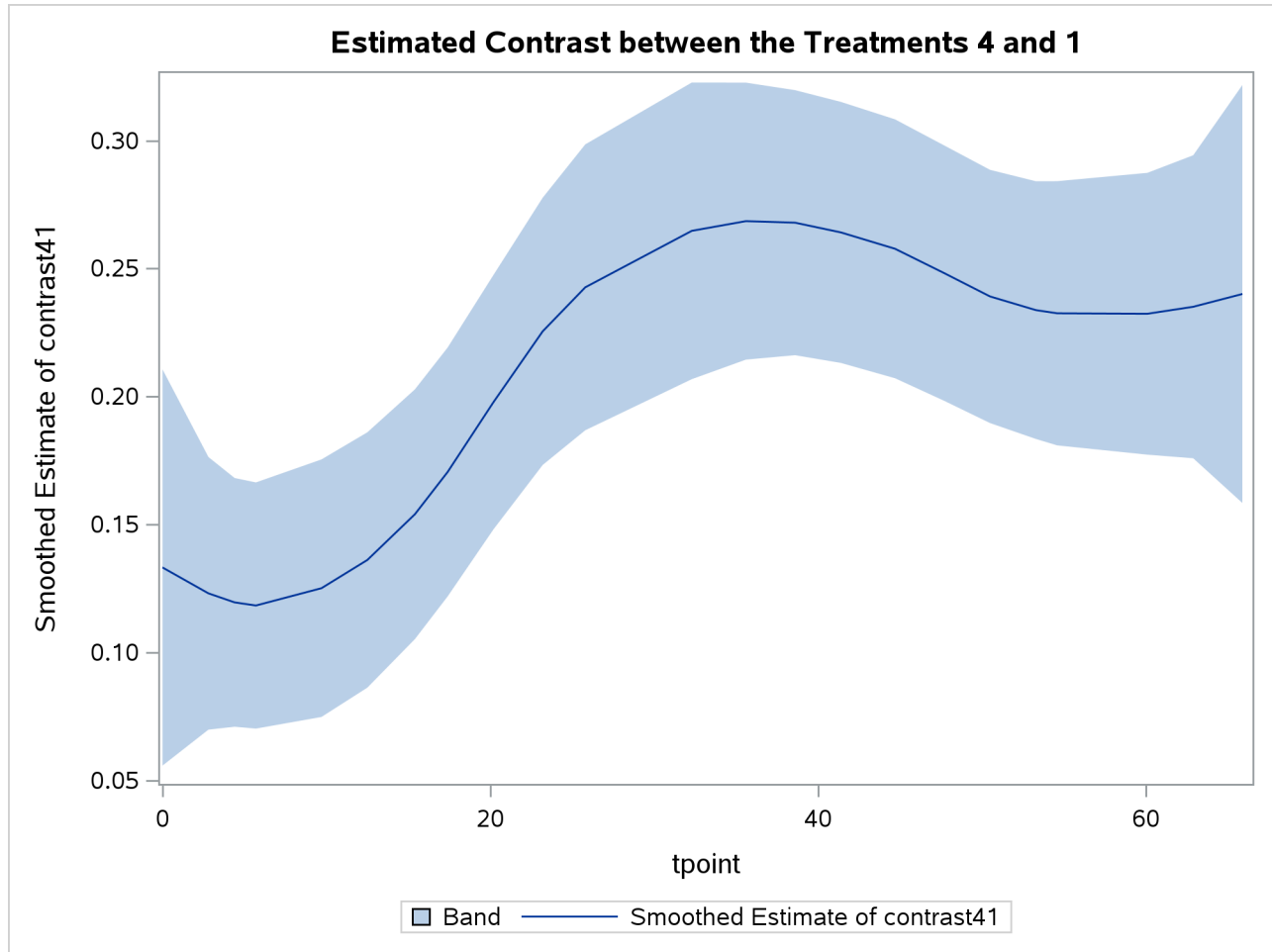
```
proc sgplot data=mylib.For1 noautolegend;
  title 'Iron and Infection Status-Specific Growth Profiles';
  band x=tpoint lower=smoothed_lower_growth
    upper=smoothed_upper_growth / group=ironInf name="band";
  series x=tpoint y=smoothed_growth / group=ironInf name="series";
  keylegend "series";
run;
```

Output 14.4.9 Model 2: Growth Profile Comparison with 95% Confidence Bands

The following statements produce the plot of smoothed $(\mu_{4,t} - \mu_{1,t})$ —contrast between the best and the worst growth profiles:

```
proc sgplot data=mylib.For1;
  title "Estimated Contrast between the Treatments 4 and 1 ";
  band x=tpoint lower=smoothed_lower_contrast41
    upper=smoothed_upper_contrast41;
  series x=tpoint y=smoothed_contrast41;
run;
```

Output 14.4.10 shows that the growth pattern of the control group “No Iron and No Infection” consistently remains above the growth pattern of the treatment group “Iron and Infection.”

Output 14.4.10 Estimated Contrast between the Treatments 4 and 1 with 95% Confidence Bands

Example 14.5: A User-Defined Trend Model

This example shows how to specify a continuous-time trend model discussed in Harvey (1989, chap. 9, sec. 9.2.1). This model is not one of the predefined trend models in the CSSM procedure. The system matrices that govern the two-dimensional state of this model are

$$\mathbf{T} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} h\sigma_1^2 + \frac{h^3\sigma_2^2}{3} & \frac{h^2\sigma_2^2}{2} \\ \frac{h^2\sigma_2^2}{2} & h\sigma_2^2 \end{bmatrix}$$

where $h = h_t = (\tau_{t+1} - \tau_t)$ denotes the difference between the successive time points, and the parameters σ_1^2 and σ_2^2 are called the level variance and the slope variance, respectively. The initial condition is fully diffuse. The trend component corresponds to the first element of this state vector. The second element of the state vector corresponds to the slope of this trend component. This model reduces to the polynomial spline model of order 2 if the level variance $\sigma_1^2 = 0$. (See the section “[Polynomial Spline Trend](#)” on page 778.)

The following statements specify a trend-plus-noise model to model the growth of cows in the previous example (“[Example 14.4: Smoothing of Repeated Measures for Longitudinal Data](#)” on page 814). The only cows that are considered are the ones that received iron and are infected.

```
data mylib.Cows;
  set Cows;
  obsNo = _n_;
run;

proc cssm data=mylib.Cows;
  where iron=1 and infection=1;
  id tpoint;
  obsindex obsNo;
  parms var1 var2 / lower=(1.e-8 1.e-8);
  array tMat{2,2};
  tMat[1,1] = 1;
  tMat[2,2] = 1;
  tMat[1,2] = _ID_DELTA_;
  array covMat{2,2};
  covMat[1,1] = var1*_ID_DELTA_ + var2*_ID_DELTA_**3/3;
  covMat[1,2] = var2*_ID_DELTA_**2/2;
  covMat[2,1] = covMat[1,2] ;
  covMat[2,2] = var2*_ID_DELTA_;
  state harveyLL(2) T(g)=(tMat) cov(g)=(covMat) a1(2);
  component trend = harveyLL[1];
  component slope = harveyLL[2];
  irregular wn;
  model weight = trend wn;
  output out=mylib.for;
run;
```

The program is easy to follow. The PARMS statement declares var1 and var2 as positive parameters, which correspond to σ_1^2 and σ_2^2 , respectively. The programming statements define arrays tMat and covMat, which later become the matrices **T** and **Q**, respectively. Note that the element tMat[2,1] is left unassigned, since it is a structural zero of **T** (see the section “[Sparse Transition Matrix Specification](#)” on page 765 for more information). Recall that the predefined variable _ID_DELTA_ contains the value of h_t , which is needed for defining the elements of **T** and **Q** (see the section “[ID Statement](#)” on page 743). The STATE statement defines the trend state vector, harveyLL, and the COMPONENT statement defines the trend component, trend, by selecting the first element of harveyLL. An additional COMPONENT statement defines the slope component, slope, as the second element of harveyLL. The slope component (which represents the cow’s growth rate) is not part of the observation equation; it is specified so that its estimate is output to For (the OUT= data set specified in the OUTPUT statement). The IRREGULAR statement defines the observation noise, and the MODEL statement defines the trend-plus-noise model.

The estimates of var1 and var2 are shown in [Output 14.5.1](#). It shows that the estimate of the level variance is nearly 0, implying that the fitted trend model is identical to the polynomial spline trend of order 2.

Output 14.5.1 Estimates of the Named Parameters

Estimates of Named Parameters			
Parameter	Estimate	Standard Error	t Value
var1	0.0000	0.0008	0.00
var2	0.0000	.	.

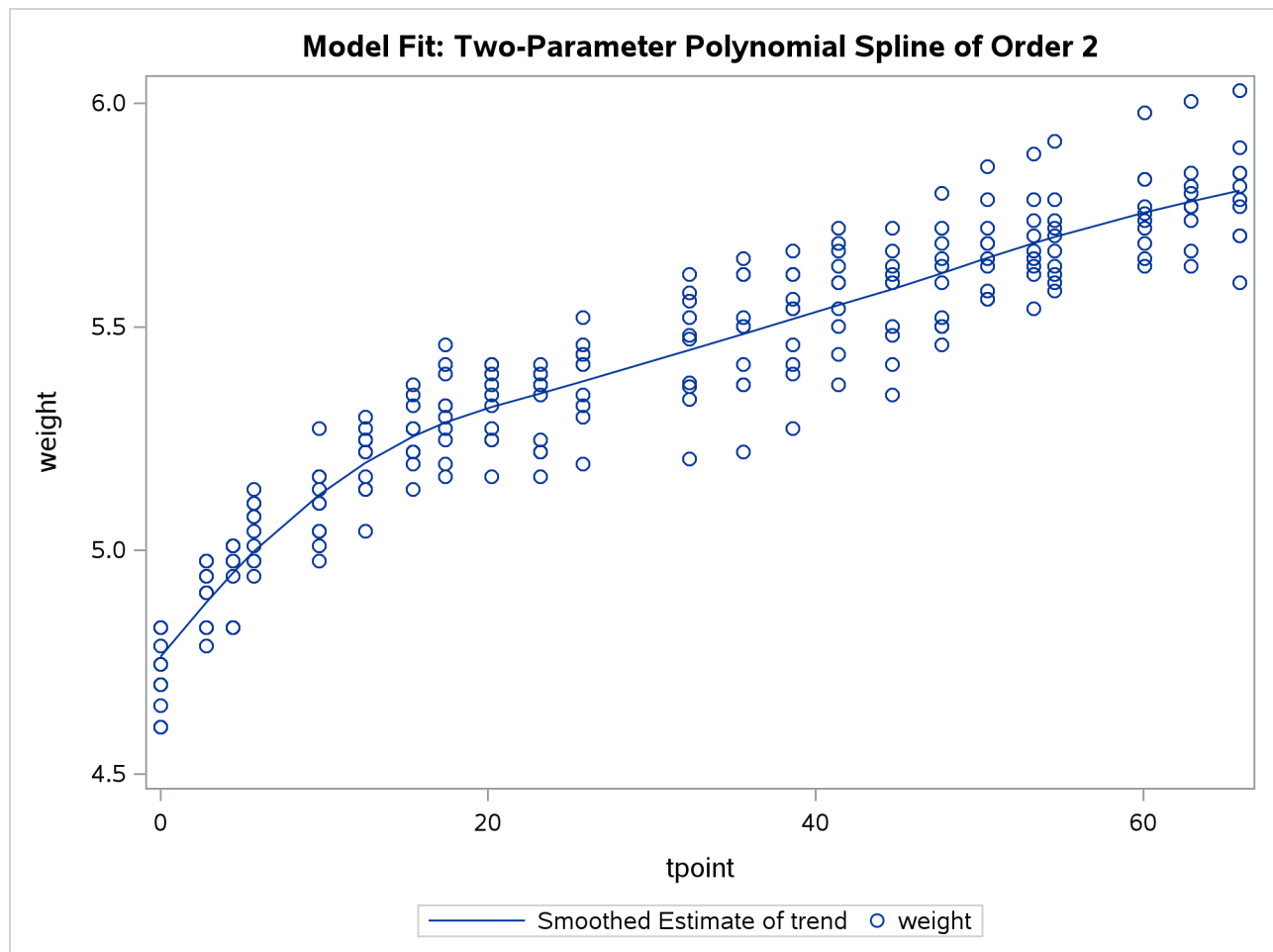
The estimate of the noise variance is shown in [Output 14.5.2](#).

Output 14.5.2 Estimates of the Unnamed Parameters

Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
wn	Irregular	Variance	0.0095	0.0009	10.49

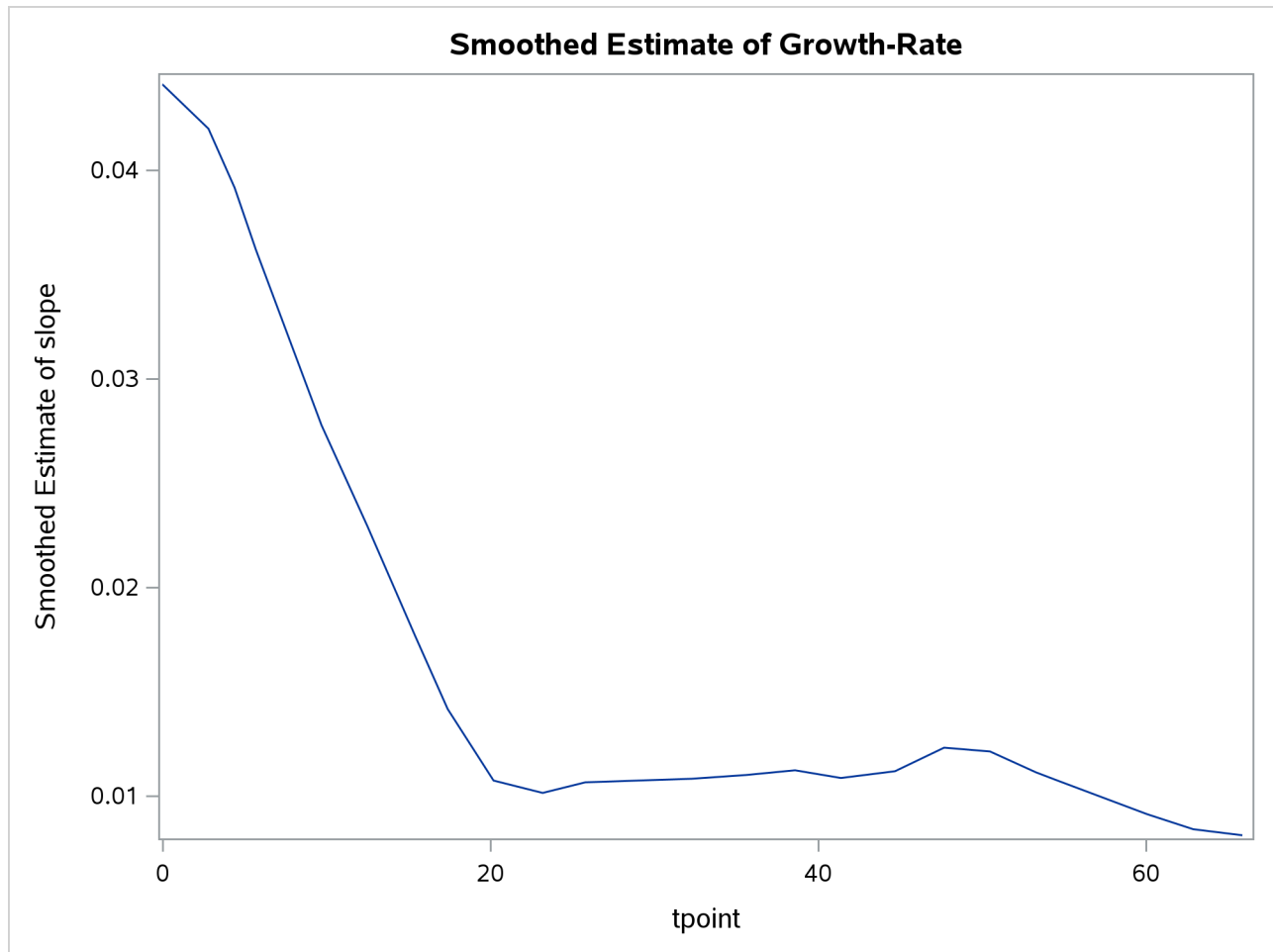
The following statements produce the plot of the fit of this trend model (shown in [Output 14.5.3](#)):

```
proc sgplot data=mylib.For;
  title "Model Fit: Two-Parameter Polynomial Spline of Order 2";
  series x=tpoint y=smoothed_trend;
  scatter x=tpoint y=weight;
run;
```

Output 14.5.3 A User-Defined Trend Model

The following statements produce the plot of the estimate of the slope component (shown in [Output 14.5.4](#)). This plot complements the preceding plot of trend; it shows the pattern of decline in the growth rate as the animals age.

```
proc sgplot data=mylib.For;
  title "Smoothed Estimate of Growth-Rate";
  series x=tpoint y=smoothed_slope;
run;
```

Output 14.5.4 Estimate of the slope Component

Example 14.6: Model with Multiple ARIMA Components

This example shows how you can fit the REGCOMPONENT models in Bell (2011) by using the CSSM procedure. The following DATA step generates the data used in the last example of this article (Example 6: “Modeling a Time Series with a Sampling Error Component”). The variable *y* in this data set contains monthly values of the VIP series (value of construction put in place), a US Census Bureau publication that measures the value of construction installed or erected at construction sites during a given month. The values of *y* are known to be contaminated with heterogeneous sampling errors; the variable *hwt* in the data set is a proxy for this sampling error in the log scale. The variable *hwt* is treated as a weight variable for the noise component in the model.

```
data Test;
  input y hwt;
  date = intnx('month', '01jan1997'd, _n_-1 );
  format date date.;
  logy = log(y);
  label logy = 'Log value of construction put in place';
  datalines;
```

```

115.2    0.042
110.4    0.042
111.5    0.067
127.9    0.122
150.0    0.129
149.5    0.135
139.5    0.152
144.6    0.168
176.0    0.173

... more lines ...

```

The following statements load the Test data into the table mylib.Test:

```

data mylib.Test;
  set Test;
run;

```

The article proposes the following model for the log VIP series:

$$\log(y) = \mu_t + hwt * \eta_t$$

where μ_t follows an $\text{ARIMA}(0,1,1) \times (0,1,1)_{12}$ model and η_t is a zero-mean, AR(2) error process. In addition, the article fixes the values of some of the model parameters to known values in order to use the known background information. The following statements specify the model in the article:

```

proc cssm data=mylib.Test;
  id date interval=month;
  parm var1=0.016565 / lower=1.e-8;
  trend airlineTrend(arma(d=1 sd=1 q=1 sq=1 s=12)) variance=var1;
  trend ar2Noise(arma(p=2)) cross=(hwt) ar=0.600 0.246 variance=0.34488;
  model logy = airlineTrend ar2Noise;
  output outfor=mylib.For;
run;

```

Output 14.6.1 Estimates of the MA Parameters in the airlineTrend Model

Model Parameter Estimates				
Component	Type	Parameter	Estimate	Standard Error t Value
airlineTrend	ARMA Trend	MA_1	0.4660	0.2798 1.67
airlineTrend	ARMA Trend	SMA_1	0.4298	0.3440 1.25

Output 14.6.2 Estimate of the Error Variance in the airlineTrend Model

Estimates of Named Parameters			
Parameter	Estimate	Standard Error	t Value
var1	0.0053	0.0031	1.74

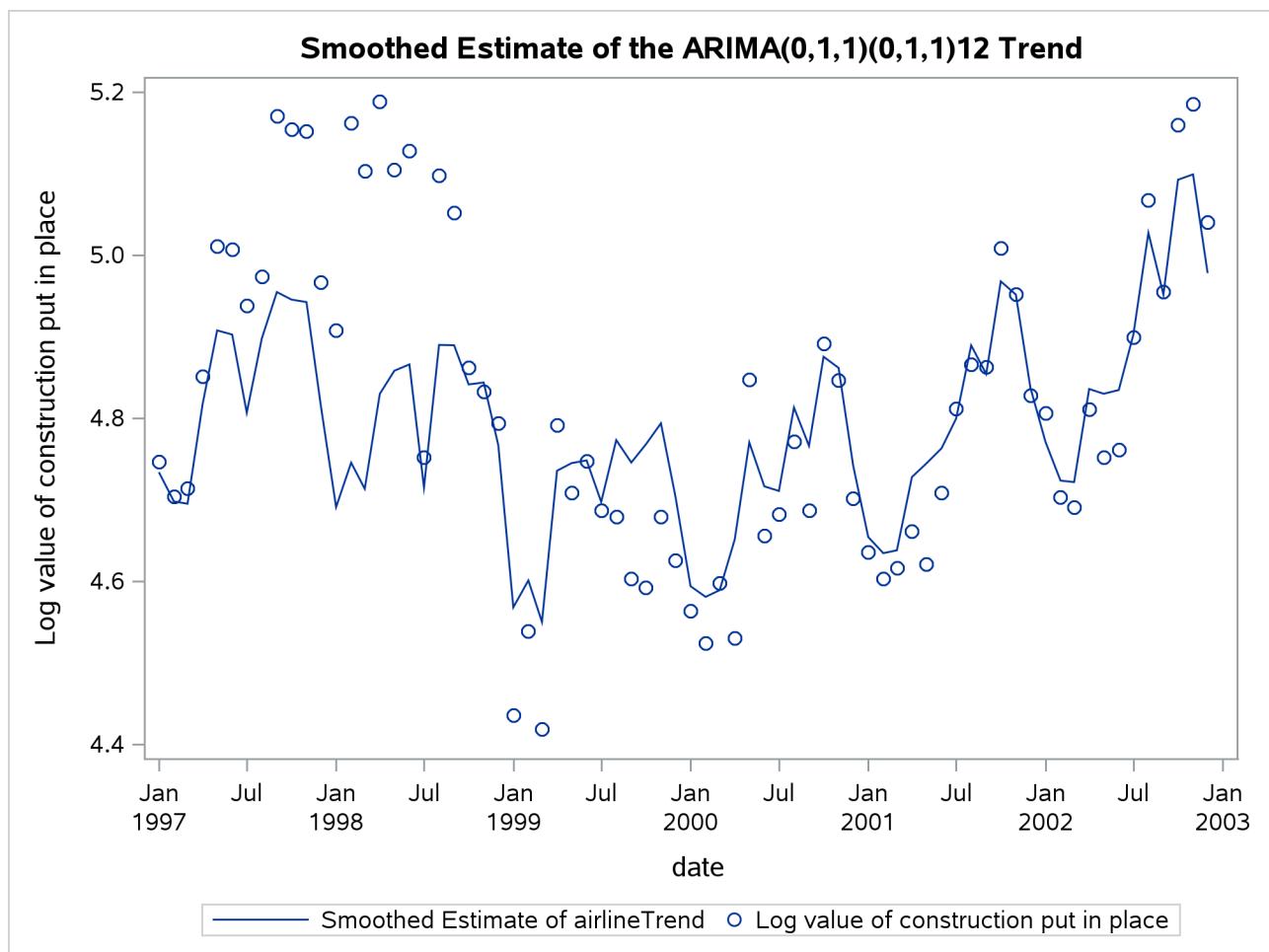
The $\text{ARIMA}(0,1,1) \times (0,1,1)_{12}$ trend μ_t is named airlineTrend and the zero-mean, AR(2) error process η_t is named ar2Noise. For more information about the ARIMA notation, see the [TREND](#) statement. The estimates

of model parameters are shown in [Output 14.6.1](#) and [Output 14.6.2](#). These estimates are quite close to the estimates given in the article, and the estimated trend and noise series are also very similar to those in the article.

The following statements produce the plot of the estimate of the `airlineTrend` component (shown in [Output 14.6.3](#)). This plot is very similar to the trend plot shown in the article (the article plots are in the antilog scale).

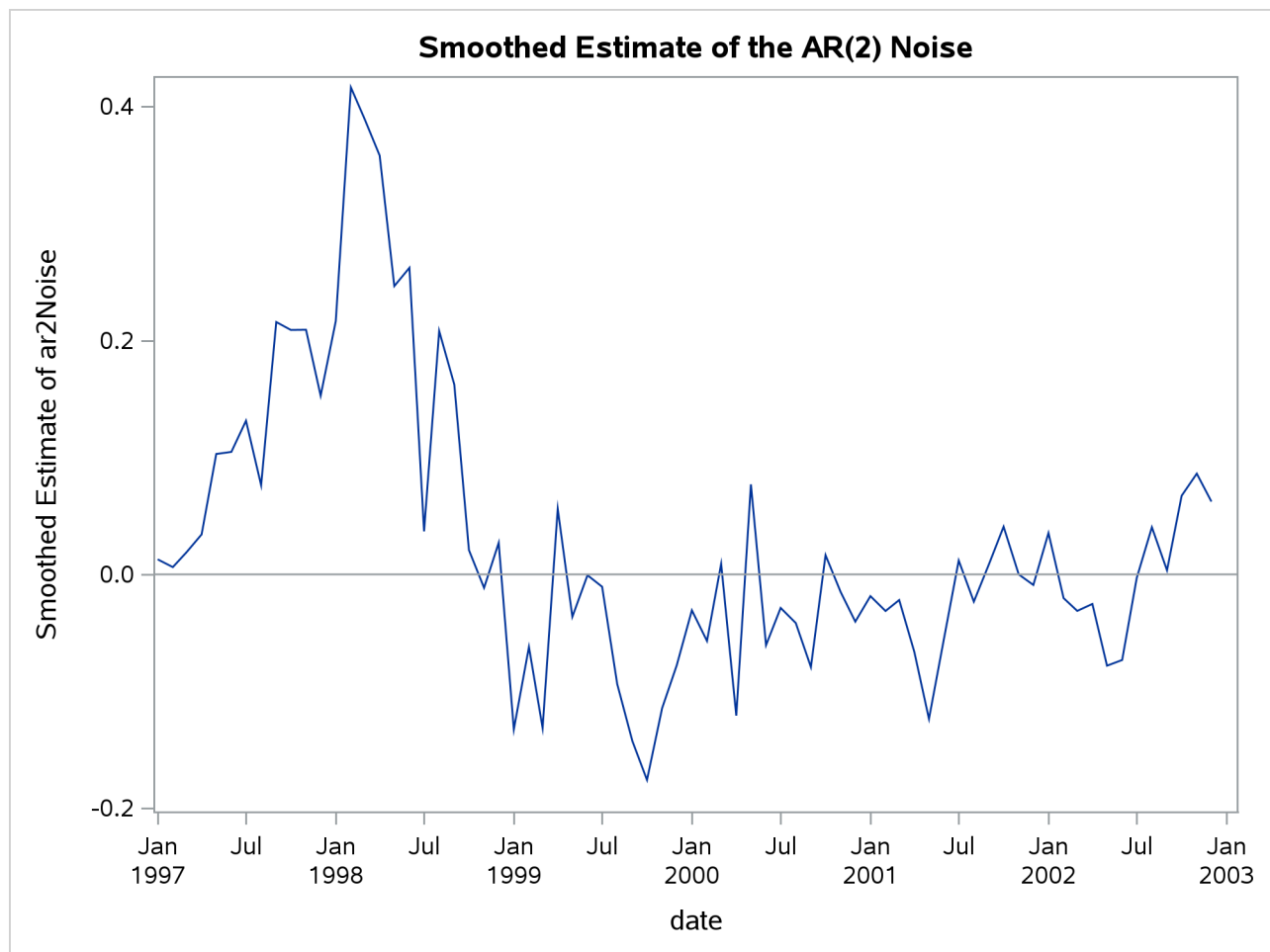
```
proc sgplot data=mylib.For;
  title "Smoothed Estimate of the ARIMA(0,1,1)(0,1,1)12 Trend";
  series x= date y=smoothed_airlineTrend;
  scatter x= date y=logy;
run;
```

Output 14.6.3 Estimate of the `airlineTrend` Component



The following statements produce the plot of the estimate of the `ar2Noise` component (shown in [Output 14.6.4](#)). This plot is also very similar to the noise plot shown in the article (once again, the article plots are in the antilog scale).

```
proc sgplot data=mylib.For;  
  title "Smoothed Estimate of the AR(2) Noise";  
  series x= date y=smoothed_ar2Noise;  
  refline 0;  
run;
```

Output 14.6.4 Estimate of the ar2Noise Component

Example 14.7: A Dynamic Factor Model for the Yield Curve

This example shows how you can fit a variant of the dynamic Nelson-Siegel (DNS) factor model discussed in Koopman, Mallee, and van der Wel (2010). Also see the example in Durbin and Koopman (2012, chap. 8, sect. 6). The following DATA step creates the yield-curve data set, `Dns`, that is used in Koopman, Mallee, and van der Wel. The data are monthly bond yields that were recorded between the start of 1970 and the end of 2000 for 17 bonds of different maturities; the maturities range from three months to 10 years (120 months). The variable `date` contains the observation date, `yield` contains the bond yield, `maturity` contains the associated bond maturity, and `mtype` contains an index (ranging from 1 to 17) that sequentially labels bonds of increasing maturity. The data have been extended for two more years by adding missing yields for the years 2001 and 2002, which causes the CSSM procedure to produce model forecasts for this span.

```
data Dns;
input date : date. yield maturity mtype;
format date date.;
datalines;
1-Jan-70      8.019      3      1
1-Jan-70      8.091      6      2
1-Jan-70      8.108      9      3

... more lines ...
```

In addition, suppose you are interested in extrapolating the fitted model to predict the yield of a hypothetical bond that has a maturity of 42 months and is not traded on the general exchange. The following DATA step creates the necessary missing values for this new bond, which is assigned the index of 18—that is, the value of `mtype` is 18:

```
data tmp1;
  set dns(keep=date);
  by date;
  if first.date then do;
    yield = .;
    maturity = 42;
    mtype = 18;
    output;
  end;
run;

proc append data=tmp1 base=dns; run;
proc sort data=dns;
by date;
run;
```

Suppose that $\theta_t(\tau)$ denotes the (idealized) yield at time t that is associated with a bond of maturity τ (in months). Even if time is not measured continuously and the bonds of only certain maturities are traded, $\theta_t(\tau)$ is treated as a smooth function of two continuous variables, time t and maturity τ . Koopman, Mallee, and van der Wel (2010) discuss a variety of models for $\theta_t(\tau)$, which is called the yield surface. One of these models depends on a positive, time-varying, scalar parameter λ_t and a time-varying three-dimensional vector

parameter β_t . This model can be described as follows:

$$\begin{aligned}\theta_t(\tau) &= \theta(\tau; \lambda_t, \beta_t) \\ &= \beta_{1t} + \beta_{2t} \left(\frac{1 - \exp(-\lambda_t \tau)}{\lambda_t \tau} \right) + \beta_{3t} \left(\frac{1 - \exp(-\lambda_t \tau)}{\lambda_t \tau} - \exp(-\lambda_t \tau) \right)\end{aligned}$$

This model is a dynamic version of a static model discussed in Nelson and Siegel (1987), where λ_t and β_t are time invariant. For fixed time period t , the three terms in this model have relatively simple interpretation. The first term β_{1t} can be thought of as the overall yield level because it does not depend on τ , the bond maturity. It can also be thought of as the long term yield because as $\tau \uparrow \infty$ the other two terms vanish; the coefficients of both β_{2t} and β_{3t} converge to 0 as $\tau \uparrow \infty$ (recall that λ_t is positive). Next, note that as $\tau \downarrow 0$ the coefficient of β_{2t} in the second term converges to 1 while that of β_{3t} in the third term converges to 0; therefore the second term can be thought of as a correction to the overall yield that is associated with the short term bonds. Finally, note that the coefficient of β_{3t} in the third term is a unimodal function of τ that decays monotonically to 0 as $\tau \downarrow 0$ and as $\tau \uparrow \infty$; therefore the third term is associated with the medium term bond yields. It is postulated that the observed yield, denoted by $y_t(\tau)$, is a noisy version of this unobserved (true) yield $\theta_t(\tau)$. The observed yield can be modeled as

$$\begin{aligned}y_t(\tau) &= \theta(\tau; \lambda_t, \beta_t) + \epsilon_{t,\tau} \\ &= \beta_{1t} + \beta_{2t} \left(\frac{1 - \exp(-\lambda_t \tau)}{\lambda_t \tau} \right) + \beta_{3t} \left(\frac{1 - \exp(-\lambda_t \tau)}{\lambda_t \tau} - \exp(-\lambda_t \tau) \right) + \epsilon_{t,\tau} \\ (\beta_t - \mu) &= \Phi(\beta_{t-1} - \mu) + \eta_t\end{aligned}$$

where $\epsilon_{t,\tau}$ are zero-mean, independent, Gaussian variables with variance σ_τ^2 , and η_t is a three-dimensional, Gaussian white noise. That is, β_t is a VAR(1) process with mean vector μ . The remainder of this example explains how to use the CSSM procedure to fit this model to the yield data in the Dns data set.

Suppose that variables Z1, Z2, and Z3 are defined as the coefficients of β_{1t} , β_{2t} , and β_{3t} , respectively. That is,

$$\begin{aligned}Z1 &= 1 \\ Z2 &= \frac{1 - \exp(-\lambda_t \tau)}{\lambda_t \tau} \\ Z3 &= \frac{1 - \exp(-\lambda_t \tau)}{\lambda_t \tau} - \exp(-\lambda_t \tau)\end{aligned}$$

In this case,

$$\theta_t(\tau) = Z1 * \beta_{1t} + Z2 * \beta_{2t} + Z3 * \beta_{3t}$$

Let $\zeta_t = \beta_t - \mu$. Then ζ_t is a zero-mean VAR(1) process and $\beta_t = \zeta_t + \mu$. In particular,

$$\begin{aligned}\theta_t(\tau) &= Z1 * \beta_{1t} + Z2 * \beta_{2t} + Z3 * \beta_{3t} \\ &= Z1 * \zeta_{1t} + Z2 * \zeta_{2t} + Z3 * \zeta_{3t} + Z1 * \mu_1 + Z2 * \mu_2 + Z3 * \mu_3\end{aligned}$$

This shows that the model for $y_t(\tau)$ can be cast into a state space form with the following observation equation:

$$y_t(\tau) = \mathbf{Z}\zeta_t + \mathbf{Z}\mu + \epsilon_{t,\tau}$$

The underlying six-dimensional state vector α_t is formed by joining the two independent subvectors, ζ_t (which is a zero-mean, VAR(1) process) and the constant mean vector μ . That is, $\alpha_t = (\zeta_{1t} \zeta_{2t} \zeta_{3t} \mu_1 \mu_2 \mu_3)'$.

Note that the variables Z2 and Z3 depend on the time varying parameter λ_t , which is unknown. λ_t is assumed to be a smooth and positive function of time t . In what follows λ_t is represented as an exponential of a cubic spline—a B-spline—in time with four evenly spaced interior knots between January 1970 and December 2002. A cubic spline with four interior knots can be represented as a sum of seven (number of knots + spline degree + 1) B-spline basis functions, $c_{1t}, c_{2t}, \dots, c_{7t}$, for example. More specifically, λ_t can be expressed as

$$\lambda_t = \exp(v1 * c_{1t} + \dots + v7 * c_{7t})$$

for some parameters $v1, v2, \dots, v7$ and the B-spline basis functions (of time) $c_{1t}, c_{2t}, \dots, c_{7t}$. Thus, the variables Z2 and Z3 become known functions of time, except for the parameters $v1, v2, \dots, v7$, which are estimated from the data. The following statements augment the Dns data set with the B-spline basis columns in two steps. First a data set that contains the basis columns, c1–c7, is created by using the BSPLINE function in the IML procedure. This data set is then merged with the Dns data set.

```
proc iml;
  use dns;
  read all var {date} into x;
  bsp = bspline(x, 2, ., 4);
  create spline var{c1 c2 c3 c4 c5 c6 c7};
  append from bsp;
quit;
data dns;
  merge dns spline;
run;
```

The following statements load the Dns data into the table mylib.Dns:

```
data mylib.Dns;
  set Dns;
run;
```

The following statements use the CSSM procedure to perform the model fitting and forecasting calculations. The variance of the observation equation disturbance for the hypothetical bond (mtype = 18) is taken to be the average of the neighboring bonds (mtype = 10 and 11), whose maturities are 36 and 48 months, respectively.

```
proc cssm data=mylib.Dns optimizer(technique=activeset) like=marginal;
  id date interval=month;
  obsindex mtype;

  /* Time-varying parameter lambda */
  parms v1-v7;
  lambda = exp(v1*c1 + v2*c2 + v3*c3 + v4*c4
              + v5*c5 + v6*c6 + v7*c7);

  /* Observation equation disturbance -- separate variance for each maturity */
  parms signal-signal17 / lower=1.e-4;
  array s_array(17) signal-signal17;
  do i=1 to 17;
    if (mtype=i) then sigma = s_array[i];
  end;
  if (mtype=18) then sigma = (signal10+signal11)/2;
```

```

irregular wn variance=sigma;

/* Variables Z1, Z2, Z3 needed in the observation equation */
Z1= 1.0;
tmp = lambda*maturity;
Z2 = (1-exp(-tmp))/tmp;
Z3 = ( 1-exp(-tmp)-tmp*exp(-tmp) )/tmp;

/* Zero-mean VAR(1) factor zeta and the associated component */
state zeta(3) type=VARMA(p=1) cov(g) print=(cov ar);
comp zetaComp = (Z1-Z3)*zeta;

/* Constant mean vector mu and the associated component */
state mu(3) type=rw;
comp muComp = (Z1-Z3)*mu;

/* Observation equation */
model yield = muComp zetaComp wn;

/* Various components defined only for output purposes */
eval yieldSurface = muComp + zetaComp;

comp zeta1 = zeta[1];
comp zeta2 = zeta[2];
comp zeta3 = zeta[3];
comp mu1 = mu[1];
comp mu2 = mu[2];
comp mu3 = mu[3];

comp z2zeta = (Z2)*zeta[2];
comp z3zeta = (Z3)*zeta[3];
comp z2Mu = (Z2)*mu[2];
comp z3Mu = (Z3)*mu[3];

eval beta1 = mu1 + zeta1;
eval beta2 = mu2 + zeta2;
eval beta3 = mu3 + zeta3;

eval shortTem = z2zeta + z2Mu;
eval medTerm = z3zeta + z3Mu;

/* output the component estimates and the forecasts */
output out=mylib.dnsFor pdv;
run;

```

The ACTIVESET optimization technique is used for parameter estimation since it is computationally more efficient in this example. The mean vector μ (recall that $\mathbf{beta}_t = \mathbf{zeta}_t + \mu$) is specified as a three-dimensional random walk with zero disturbance covariance (signified by the absence of COV= option). The model specification part of the program ends with the MODEL statement; the subsequent COMP and EVAL statements define some useful linear combinations of the underlying state. Their estimates are computed after the model fit is completed and are output to the output data set mylib.dnsFor. The mylib.dnsFor data set also contains all the program variables and the parameters defined in the PARMS statement because the

OUTPUT statement contains the PDV option.

Output 14.7.1 shows the estimated mean vector (μ). It shows that the mean long-term yield is 7.64. Output 14.7.2 shows the estimates of v_1 – v_7 (used for defining time-varying λ_t) and the maturity specific observation variances. Output 14.7.3 shows the estimate of the VAR(1) transition matrix Φ , and Output 14.7.4 shows the associated disturbance covariance matrix Σ . The model fit summary is shown in Output 14.7.5.

Output 14.7.1 Estimate of the Mean Vector (μ)

Estimates of Fixed State Effects					
State	Element		Standard		
	Index	Estimate	Error	t Value	Pr > t
mu	1	7.8645	1.1408	6.89	<.0001
mu	2	-1.3153	0.4924	-2.67	0.0076
mu	3	-0.2854	0.3176	-0.90	0.3688

Output 14.7.2 Estimates of v_1 – v_7 and Observation Variances

Estimates of Named Parameters			
Parameter	Estimate	Standard	
		Error	t Value
v1	-1.2429	0.3015	-4.12
v2	-2.9077	0.1112	-26.15
v3	-1.8756	0.0680	-27.57
v4	-2.3168	0.0779	-29.75
v5	-3.1985	0.1015	-31.50
v6	-1.6574	0.3157	-5.25
v7	-4.5601	1.5675	-2.91
sigma1	0.0533	0.0046	11.66
sigma2	0.0031	0.0008	3.72
sigma3	0.0087	0.0007	11.67
sigma4	0.0110	0.0009	12.17
sigma5	0.0087	0.0008	11.44
sigma6	0.0060	0.0006	10.56
sigma7	0.0052	0.0005	10.58
sigma8	0.0054	0.0005	10.91
sigma9	0.0056	0.0005	11.26
sigma10	0.0064	0.0006	11.43
sigma11	0.0103	0.0008	12.17
sigma12	0.0074	0.0007	10.98
sigma13	0.0111	0.0009	11.67
sigma14	0.0120	0.0011	11.36
sigma15	0.0125	0.0012	10.73
sigma16	0.0216	0.0019	11.64
sigma17	0.0275	0.0023	11.97

Output 14.7.3 Transition Matrix, Φ , Associated with ζ

AR Coefficient for zeta			
	Col1	Col2	Col3
Row1	0.9974	0.0322	-0.0261
Row2	-0.0290	0.9304	0.0596
Row3	0.0230	0.0640	0.7664

Output 14.7.4 Estimated Disturbance Covariance of ζ

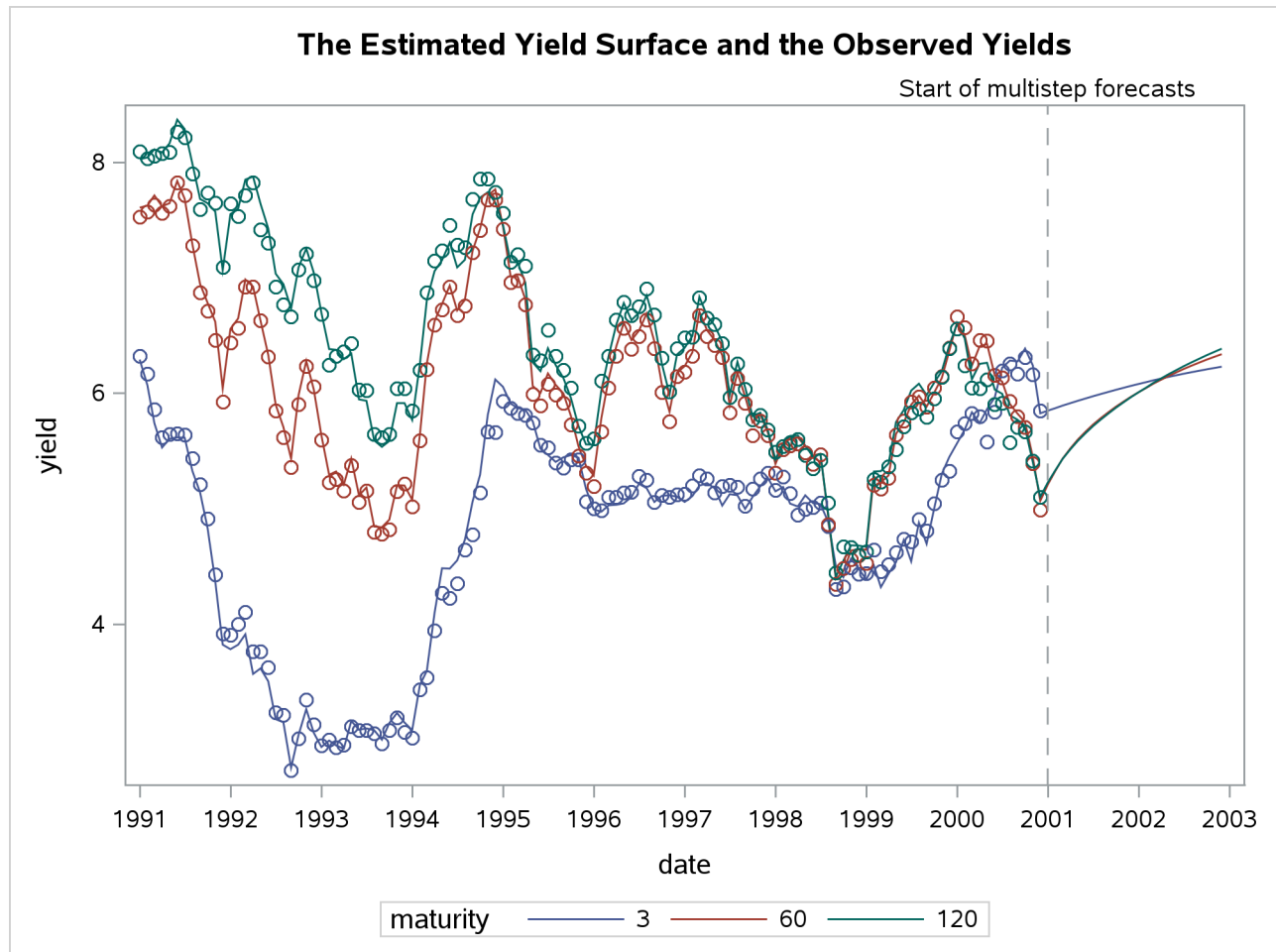
Disturbance Covariance for zeta			
	Col1	Col2	Col3
Row1	0.1029	-0.0210	0.0847
Row2	-0.0210	0.3531	0.00588
Row3	0.0847	0.00588	1.0794

Output 14.7.5 Likelihood Computation Summary for the DNS Factor Model

Likelihood Computation Summary	
Nonmissing Response Values Used	6324
Estimated Parameters	39
Initialized Diffuse State Elements	3
Normalized Residual Sum of Squares	6321
Diffuse Log Likelihood	3561
Profile Log Likelihood	3560
Marginal Log Likelihood	3570

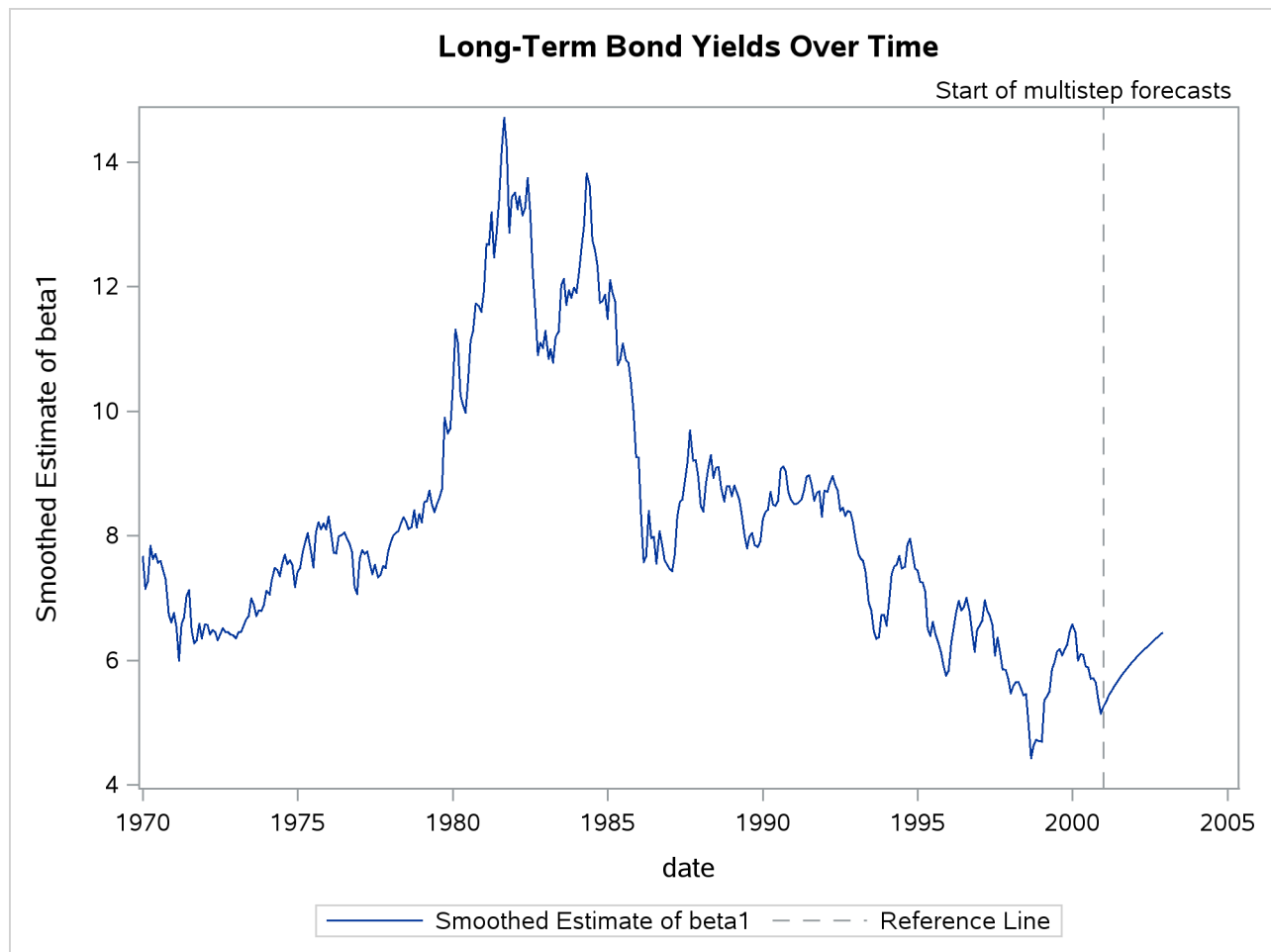
The following statements produce the time series plots of the smoothed estimate of the idealized bond yield ($\theta_t(\tau)$) for bonds with maturities 30, 60, and 120 months (shown in [Output 14.7.6](#)). To simplify the display, the plots exclude the time span prior to 1991.

```
proc sgplot data= mylib.dnsFor;
  title "The Estimated Yield Surface and the Observed Yields ";
  where maturity in (3 60 120) and date >= '31dec1990'd;
  series x=date y=smoothed_yieldSurface / group=maturity;
  scatter x=date y=yield / group=maturity;
  refline '31dec2000'd / axis=x lineattrs=GraphReference(pattern = Dash)
    name="RefLine" label="Start of multistep forecasts";
run;
```

Output 14.7.6 Smoothed Estimate of $\theta_t(\tau)$ for $\tau = 3, 60, 120$ 

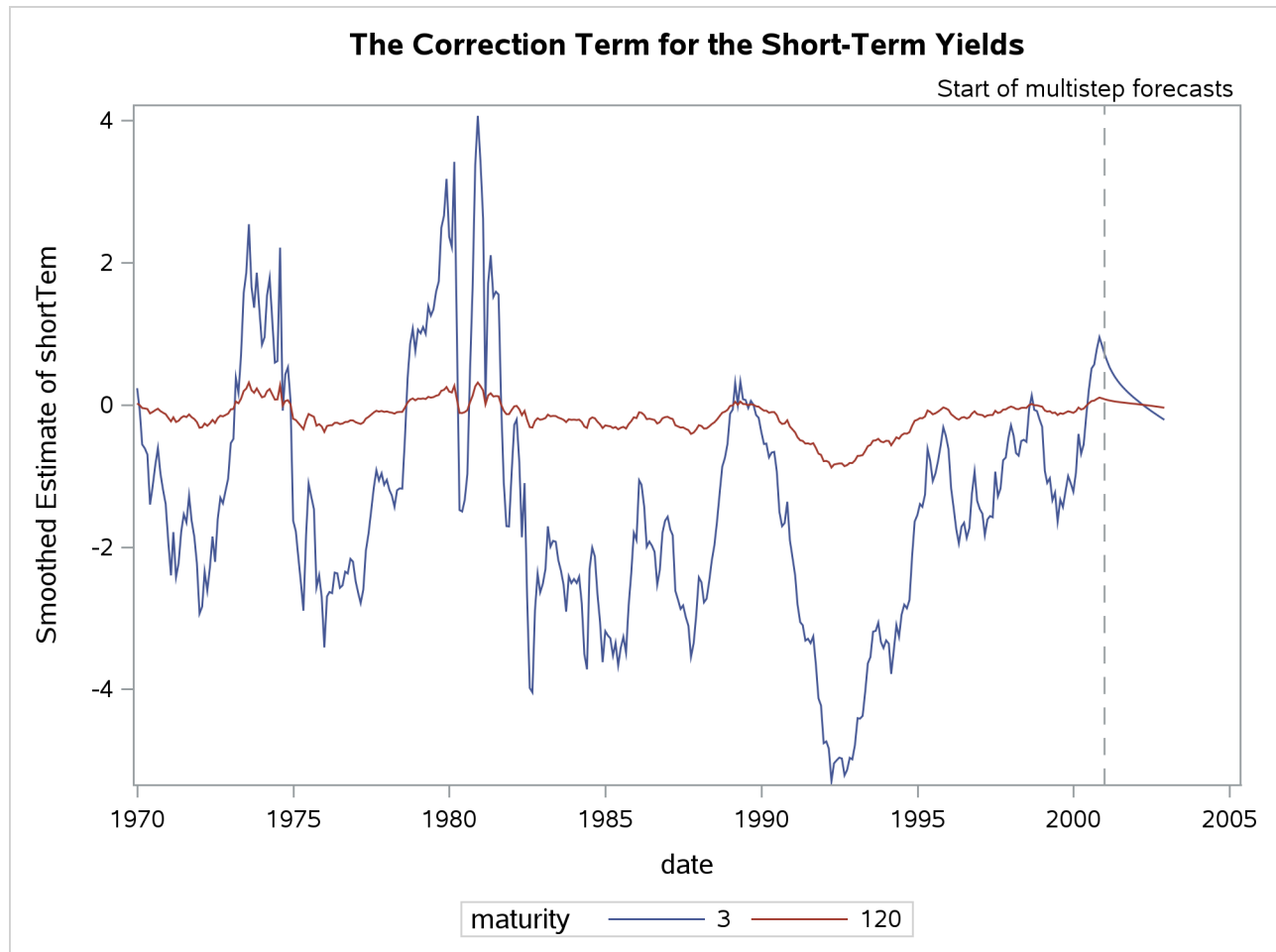
The plots indicate that the DNS model is a reasonable description of the yield data. Similar plots (not shown here) for other maturities also indicate the adequacy of the DNS model. The following statements produce the time series plot of the smoothed estimate of β_{1t} , the long-term bond yield (shown in [Output 14.7.7](#)) :

```
proc sgplot data=mylib.dnsFor;
  title "Long-Term Bond Yields Over Time ";
  series x=date y=smoothed_beta1 ;
  refline '31dec2000'd / axis=x lineattrs=GraphReference(pattern = Dash)
    name="RefLine" label="Start of multistep forecasts";
run;
```

Output 14.7.7 Smoothed Estimate of β_{1t} , the Long-Term Yield

Similarly, [Output 14.7.8](#), which is produced by the following statements, shows the smoothed estimate of the correction to the overall yield that is provided by the second term ($Z_2 * \beta_{2t}$) for maturities of 3 months and 120 months. As expected, the correction for the (long-term) maturity of 120 months is negligible compared to the (short-term) maturity of 3 months.

```
proc sgplot data=mylib.dnsFor;
  title "The Correction Term for the Short-Term Yields ";
  where maturity in (3 120);
  series x=date y=smoothed_shortTem / group=maturity;
  refline '31dec2000'd / axis=x lineattrs=GraphReference(pattern = Dash)
    name="RefLine" label="Start of multistep forecasts";
run;
```


Output 14.7.8 Smoothed Estimate of $Z_2 * \beta_{2t}$, the Correction Term for the Short-Term Yields

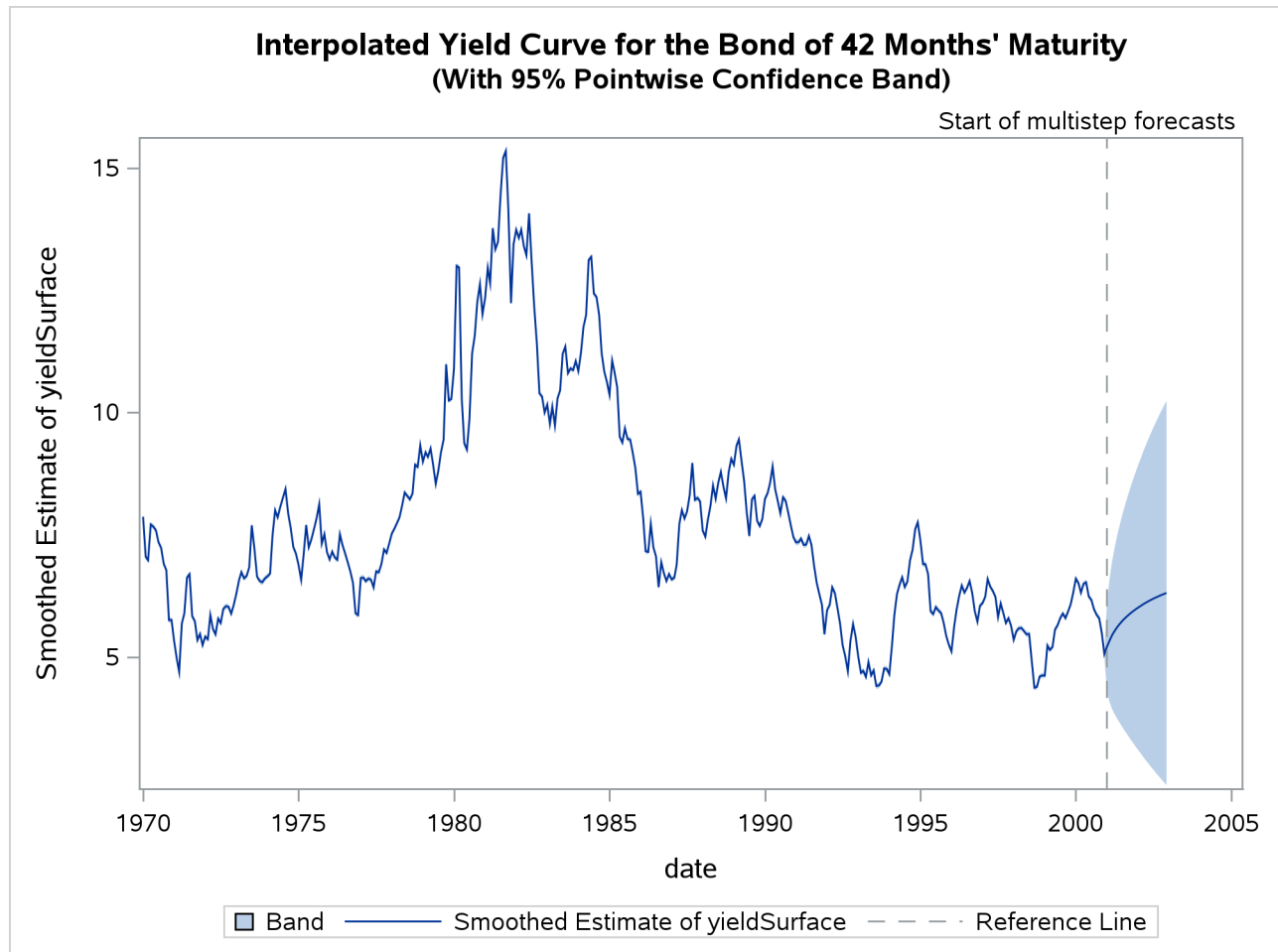
The following statements create plots that show the estimated yield for the hypothetical bond whose maturity is 42 months:

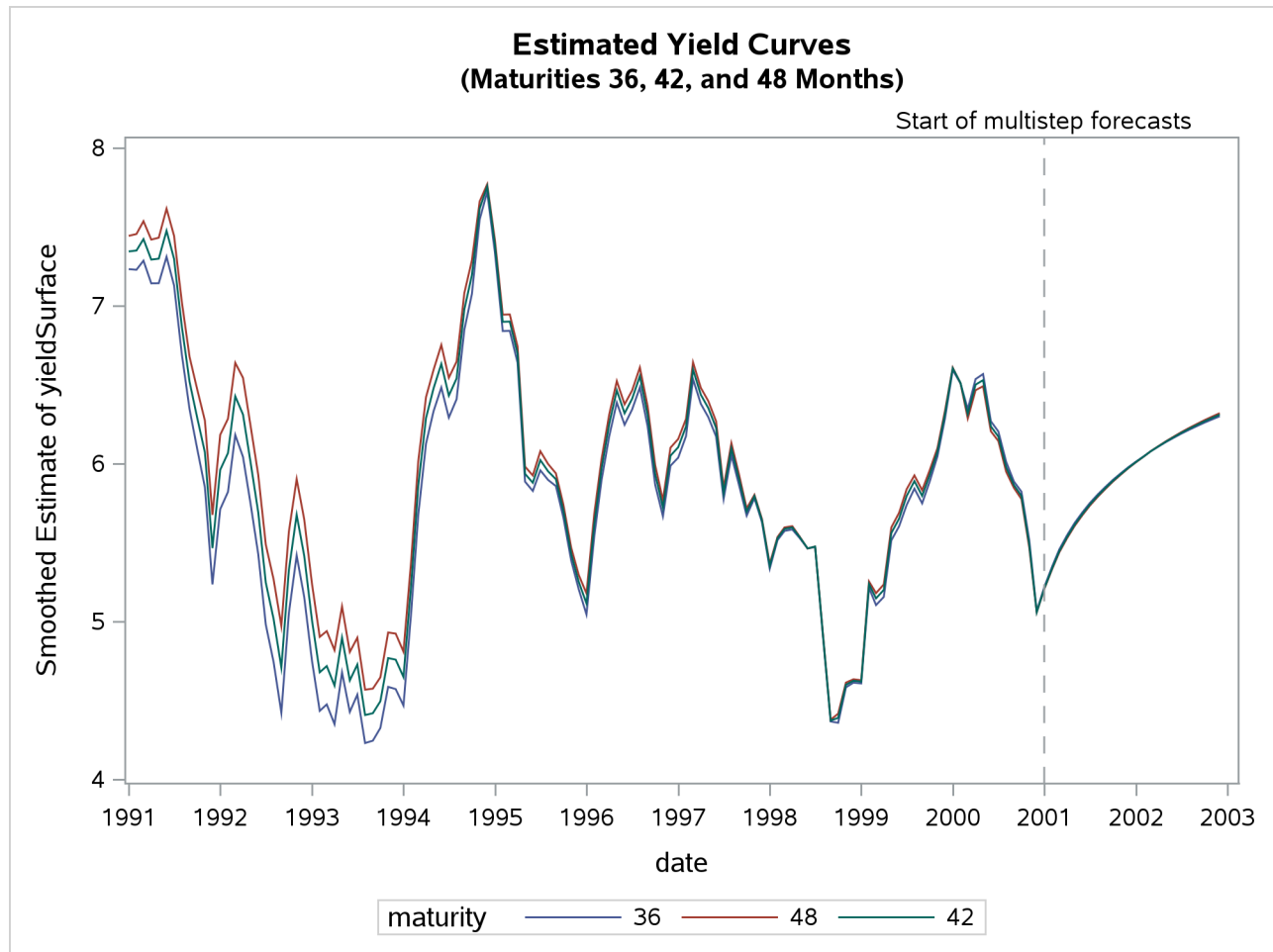
```
proc sgplot data=mylib.dnsFor;
  title "Interpolated Yield Curve for the Bond of 42 Months' Maturity";
  title2 "(With 95% Pointwise Confidence Band)";
  where maturity in (42);
  band x=date lower=smoothed_lower_yieldSurface upper=smoothed_upper_yieldSurface;
  series x=date y=smoothed_yieldSurface;
  refline '31dec2000'd / axis=x lineattrs=GraphReference(pattern = Dash)
    name="RefLine" label="Start of multistep forecasts";
run;

proc sgplot data= mylib.dnsFor;
  title "Estimated Yield Curves";
  title2 "(Maturities 36, 42, and 48 Months)";
  where maturity in (36 42 48) and date >= '31dec1990'd;
  series x=date y=smoothed_yieldSurface / group=maturity;
  refline '31dec2000'd / axis=x lineattrs=GraphReference(pattern = Dash)
    name="RefLine" label="Start of multistep forecasts";
run;
```

Output 14.7.9 shows the interpolated yield curve with a pointwise 95% confidence band. In the historical period, the confidence band appears too tight, mostly because of graphical scaling.

Output 14.7.9 Interpolated Yield Curve for 42 Months' Maturity



Output 14.7.10 Estimated Yield Curves

Output 14.7.10 shows the estimates of $\theta_t(\tau)$ for $\tau = 36, 42$, and 48 months. As expected, the estimated $\theta_t(42)$ lies between the estimates of $\theta_t(36)$ and $\theta_t(48)$.

Example 14.8: Diagnostic Plots and Structural Break Analysis

This example provides information about the diagnostic plots that the CSSM procedure produces. In addition, a simple illustration of structural break analysis is also provided. For additional examples of structural break analysis, see Selukar (2017). The following plots are available in the CSSM procedure:

- a panel of two plots—a histogram and a Q-Q plot—for the normality check of the one-step-ahead residuals $v_{t,i}$. A separate panel is produced for each response variable.
- a time series plot of standardized residuals, one per response variable
- a panel of two plots—a histogram and a Q-Q plot—for the normality check of the prediction errors $AO_{t,i}$. A separate panel is produced for each response variable.
- a time series plot of standardized prediction errors, one per response variable

- a time series plot of maximal state shock chi-square statistics

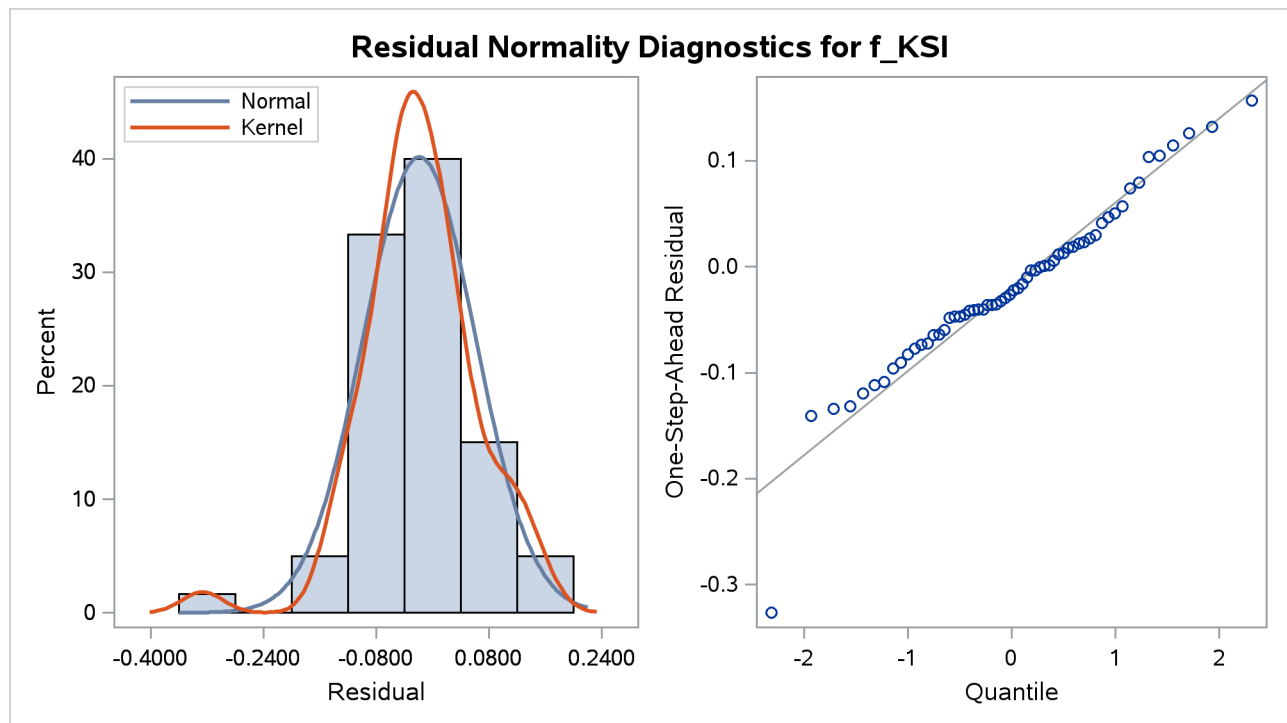
All these plots are used primarily for model diagnostics. In this example, the automobile seat-belt data that are discussed in “[Example 14.1: Bivariate Basic Structural Model](#)” on page 803 are revisited. In “[Example 14.1: Bivariate Basic Structural Model](#)” on page 803, the question under consideration is whether the data show evidence of the effectiveness of the seat-belt law that was introduced in the first quarter of 1983. An intervention variable, `Q1_83_Shift`, was used in the model to measure the effect of this law on the drivers and front-seat passengers who were killed or seriously injured in car accidents (`f_KSI`). In the current example, the analysis of these data begins without the knowledge of this seat-belt law. In effect, the same model is fitted without the use of the intervention variable `Q1_83_Shift`.

```
data mylib.seatBelt;
  set seatBelt;
run;
```

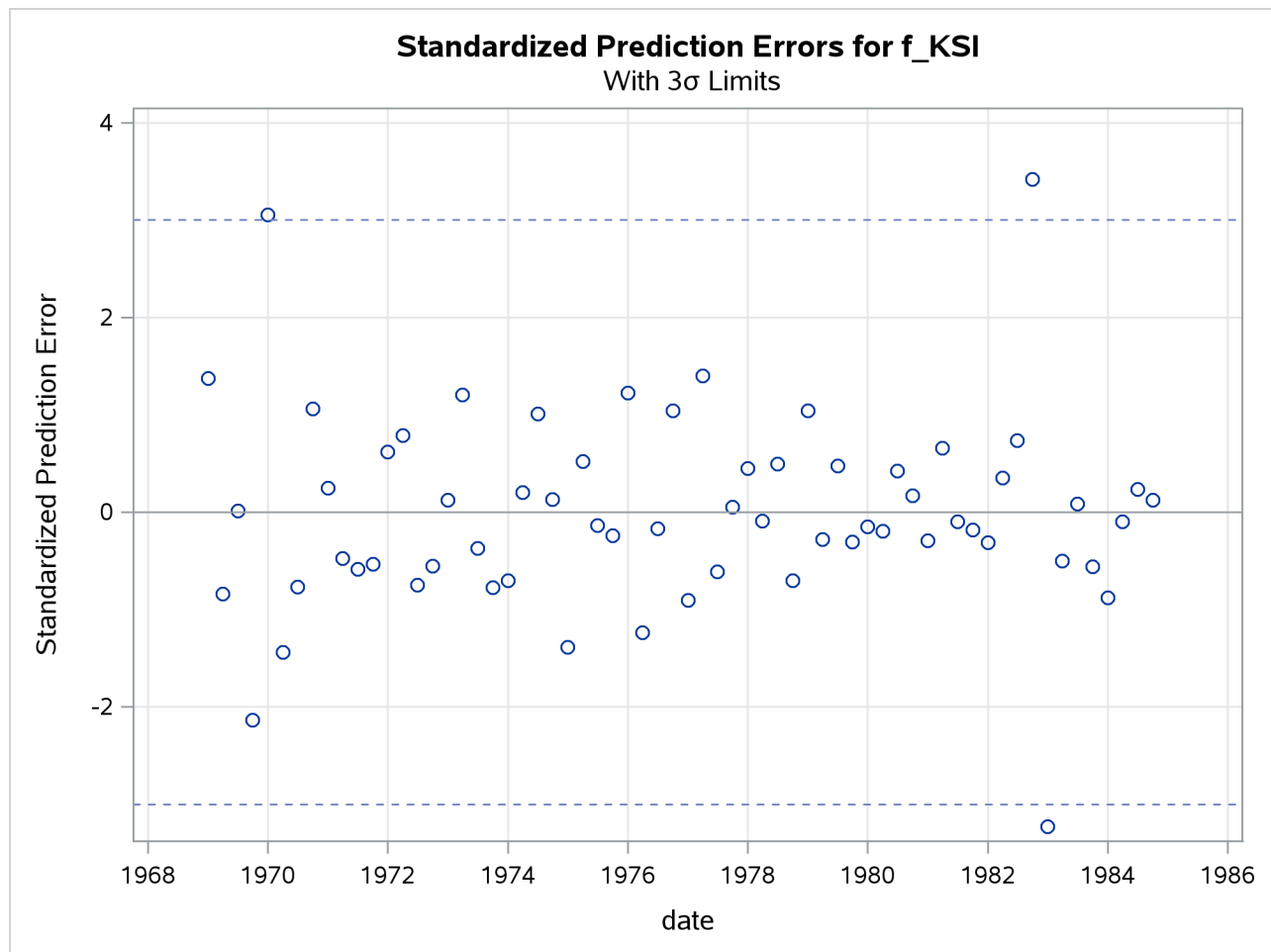
The following statements specify the model (without the intervention variable):

```
proc cssm data= mylib.seatBelt optimizer(tech=interiorpoint) plots=all;
  id date interval=quarter;
  state error(2) type=WN cov(g);
  component wn1 = error[1];
  component wn2 = error[2];
  state level(2) type=RW cov(rank=1) checkbreak;
  component rw1 = level[1];
  component rw2 = level[2];
  state season(2) type=season(length=4);
  component s1 = season[1];
  component s2 = season[2];
  model f_KSI = rw1 s1 wn1;
  model r_KSI = rw2 s2 wn2;
run;
```

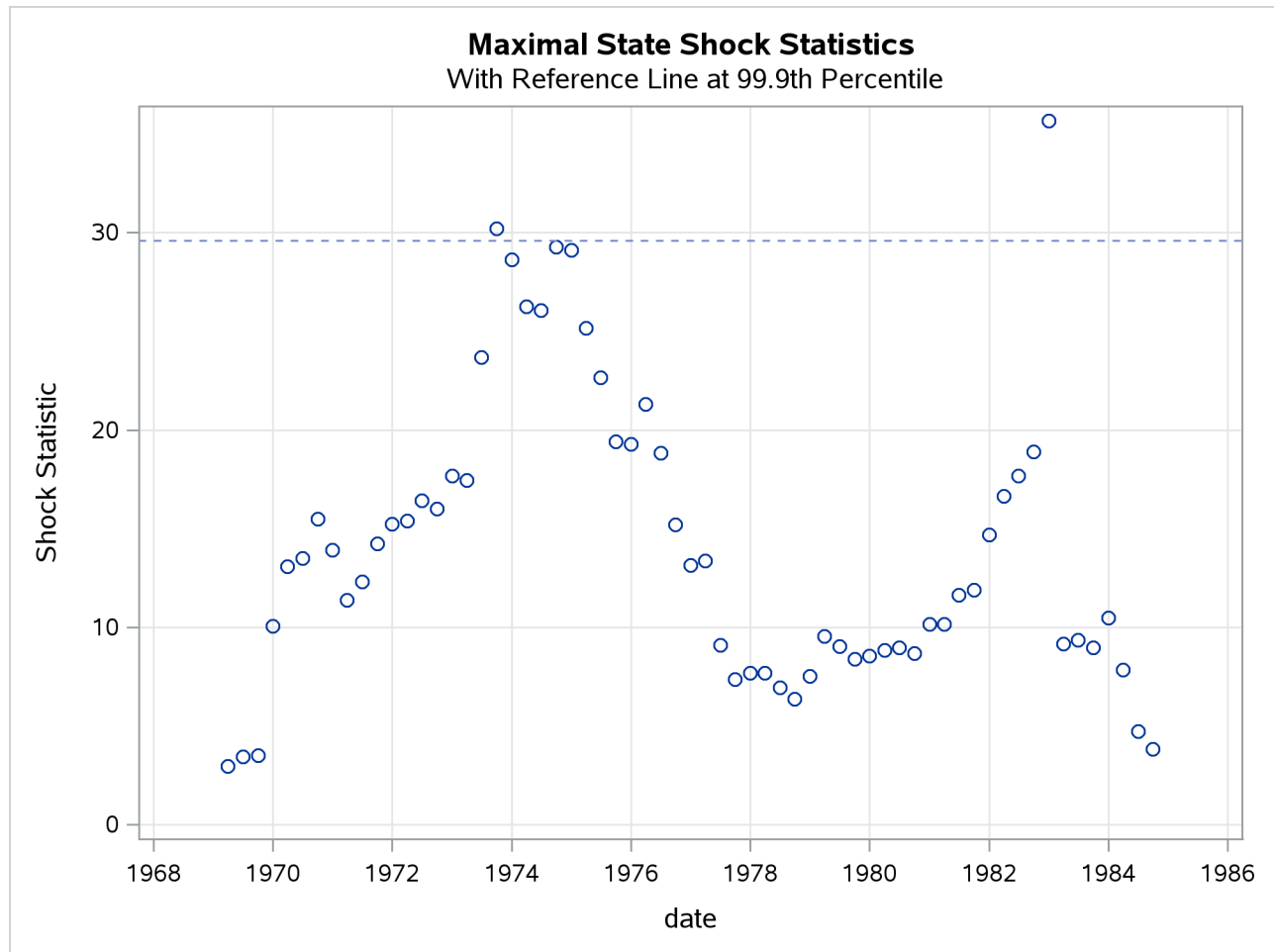
The `PLOTS=ALL` option in the `PROC CSSM` statement turns on all the plotting options. Because there are two response variables, nine plots in total are produced: a separate set of four plots—two residual and two prediction error—is produced for `f_KSI` and `r_KSI`, and one maximal shock plot is produced. Only three of these plots are shown here. [Output 14.8.1](#) shows the normality check for the one-step-ahead residuals for `f_KSI`. It shows some evidence of lack of normality.

Output 14.8.1 Normality Check of One-Step-Ahead Residuals for f_KSI 

Output 14.8.2 shows the time series plot of standardized prediction errors for f_KSI . It identifies some extreme observations (additive outliers): two near 1983 and one near 1970.

Output 14.8.2 Time Series Plot of Standardized Prediction Errors for f_KSI 

Output 14.8.3 shows the time series plot of maximal shock statistics. This plot can be very informative in showing the temporal locations of the structural changes in the overall observation-generation process (treating the fitted model as the reference). It can indicate locations of shifts in the process level or shifts in other characteristics, such as its slope. The precise nature of the shift (whether the shift occurs in the level or in some other aspects) can be determined by using the CHECKBREAK option in the appropriate STATE and TREND statements (as is done in the STATE statement in this example that defines the bivariate state level). In this example, the maximal shock statistics plot indicates two locations—the last quarter of 1973 and the first quarter of 1983—as likely locations for the structural breaks that are associated with the traffic accident process. These are indeed reasonable findings, because the last quarter of 1973 (beginning in October 1973) is associated with the start of the oil crisis that severely curtailed worldwide automobile traffic, and the first quarter of 1983 is associated with the introduction of the seat-belt law that might have improved the safety of drivers and front-seat passengers. In addition, **Output 14.8.4** shows the summary of most likely break locations for the bivariate state level. It identifies a break in the first element of level (which corresponds to the drivers and front-seat passengers) in the first quarter of 1983.

Output 14.8.3 Time Series Plot of Maximal Shock Statistics**Output 14.8.4** Elementwise Break Summary for the Bivariate State: level

Elementwise Break Summary for level			
Element			
ID	Index	Z Value	Pr > z
1983/1	1	-5.85	<.0001

The following statements fit a revised model that accounts for the break in the first element of level by introducing a dummy variable, Q1_83_Pulse, in the state equation:

```
ods output ElementStateBreakDetails=stateBreak;
proc cssm data= mylib.seatBelt optimizer(tech=interiorpoint) plots=all;
  id date interval=quarter;
  Q1_83_Pulse = (date = '1jan1983'd);
  zero = 0;
  state error(2) type=WN cov(g);
  component wn1 = error[1];
  component wn2 = error[2];
```

```

state level(2) type=RW cov(rank=1) W(g)=(Q1_83_Pulse zero)
  checkbreak print=breakdetail;
component rw1 = level[1];
component rw2 = level[2];
state season(2) type=season(length=4);
component s1 = season[1];
component s2 = season[2];
model f_KSI = rw1 s1 wn1;
model r_KSI = rw2 s2 wn2;
run;

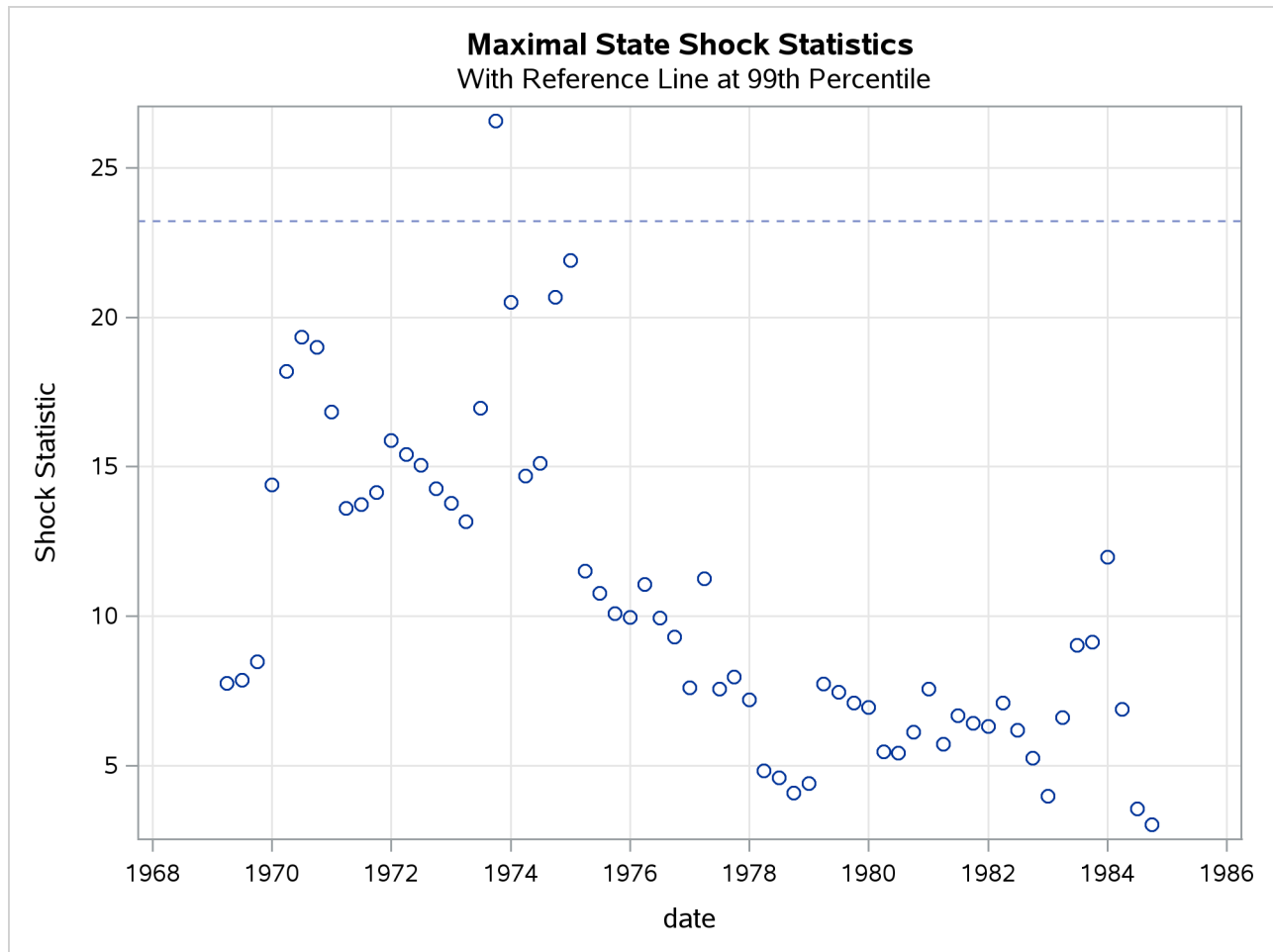
```

Note that using Q1_83_Pulse in the definition of level is equivalent to using Q1_83_Shift in the MODEL statement for f_KSI in “[Example 14.1: Bivariate Basic Structural Model](#)” on page 803. [Output 14.8.5](#) shows the estimated change in the first element of the state level, which is the same as the estimated level shift shown in [Output 14.1.6](#) (this is not surprising, because these two models are statistically equivalent).

Output 14.8.5 Estimate of the Regression Coefficient of Q1_83_Pulse

Estimate of the State Equation Regression Vector					
State	Element Index	Estimate	Standard Error	t Value	Pr > t
level	1	-0.4084	0.0259	-15.74	<.0001

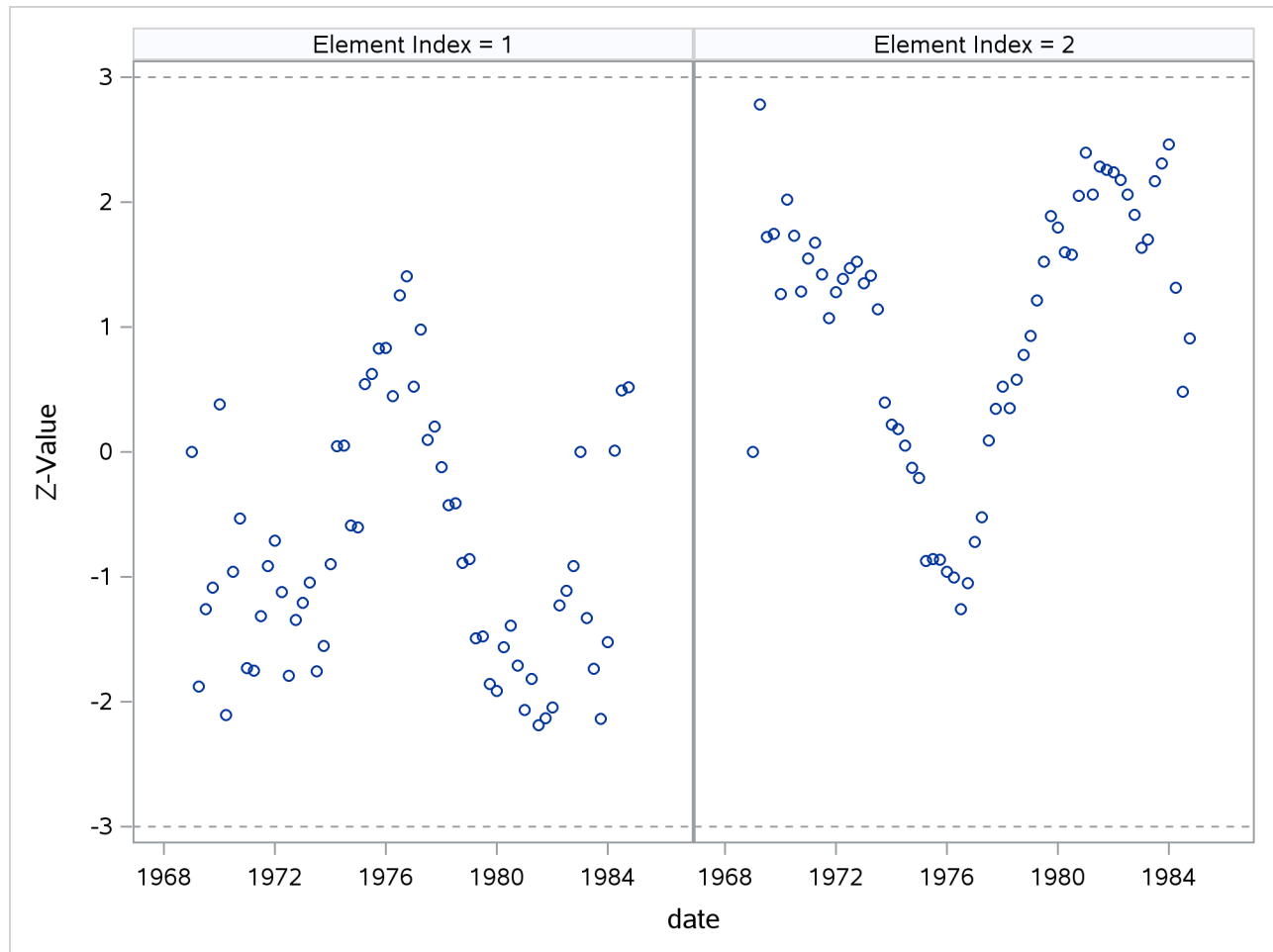
In the preceding CSSM procedure statements, the CHECKBREAK option is used along with the PRINT=BREAKDETAIL option, which produces a table that contains the break statistics at every distinct time point (this table, in turn, is captured in the output data set stateBreak for later use). [Output 14.8.6](#) shows the time series plot of maximal shock statistics for this revised model. As expected, the plot no longer shows the first quarter of 1983 as a structural break location. It continues to show the last quarter of 1973 as a structural break location, because the fitted model does not try to explicitly account for this shift.

Output 14.8.6 Time Series Plot of Maximal Shock Statistics for the Model with Q1_83_Pulse

Note that the reference line in [Output 14.8.3](#) is drawn at the 99.9th percentile, whereas the reference line in [Output 14.8.6](#) is drawn at the 99th percentile. The reference line location in the maximal state shock chi-square statistics plot is based on the points in the plot. A reference line is drawn at percentile 80, 90, 99, or 99.9 based on the largest maximal shock statistic that is shown.

The detailed information in the data set `stateBreak` can be used to further investigate the possibility of significant breaks in the trend in and around 1973. The following statements produce scatter plots for the break statistics for both the drivers and front passengers and the rear passengers (reference lines are also drawn at -3 and 3 to check for extreme Z values):

```
proc sgpanel data=stateBreak;
  panelby elementIndex;
  scatter x=time y=zValue;
  refline 3 / axis=y lineattrs=(pattern=shortdash) noclip;
  refline -3 / axis=y lineattrs=(pattern=shortdash) noclip;
run;
```

Output 14.8.7 Elementwise Structural Break Statistics for level

The resulting graph, shown in [Output 14.8.7](#), shows possible breaks in the second element—rear side passengers—around 1969. In general, however, the evidence of breaks in the elements of level is not very strong. This means that you must look elsewhere to explain the extreme point in [Output 14.8.6](#).

Example 14.9: Variable Bandwidth Smoothing for Longitudinal Data

The data for this example, taken from Givens and Hoeting (2005, chap. 11, Example 11.8), contain two variables, x and y . The variable y represents noisy evaluation of an unknown smooth function at x . The data are sorted by x .

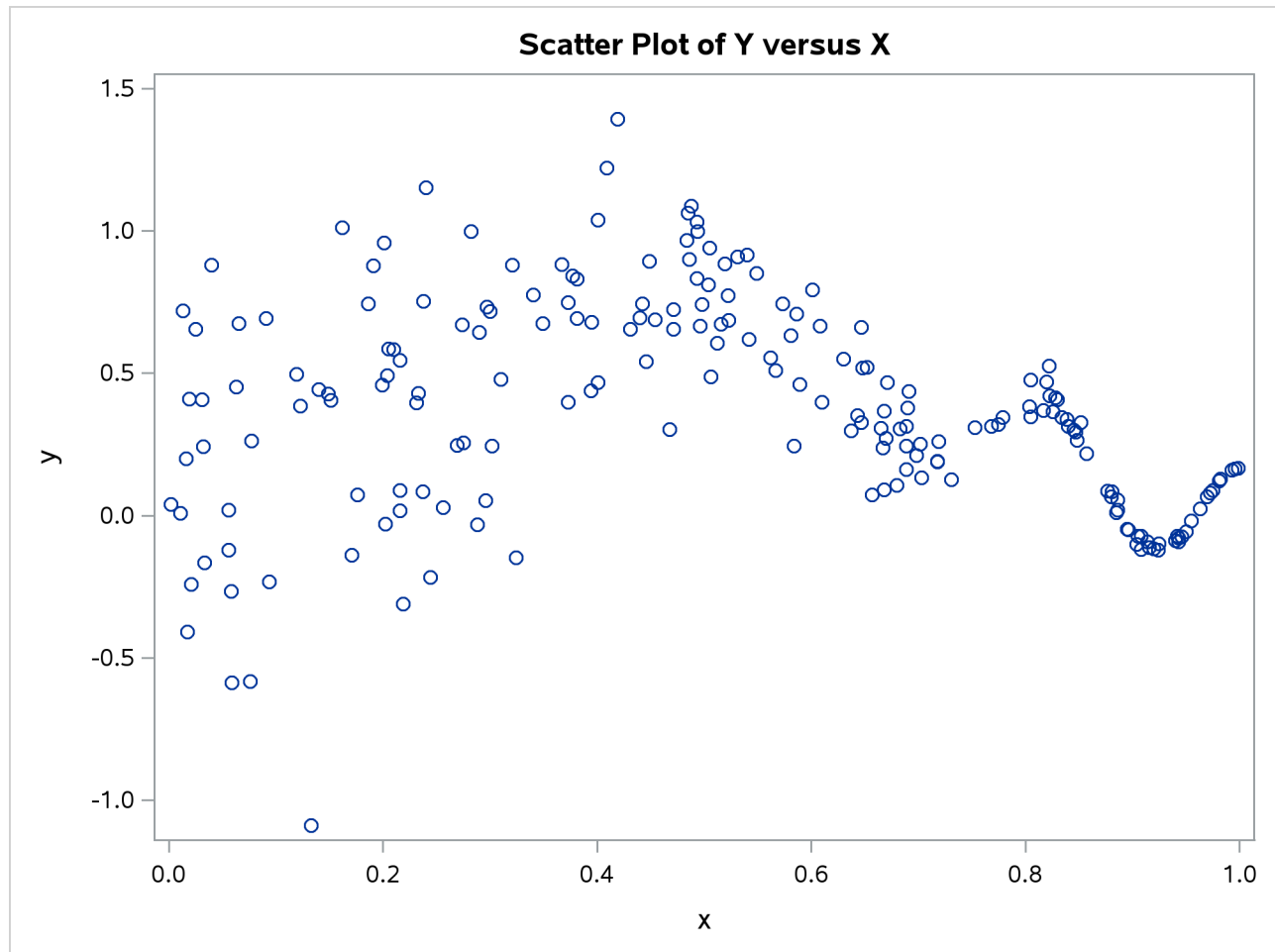
```
data Difficult;
input x y;
datalines;
0.002 0.040
0.011 0.009
0.013 0.719
0.016 0.199
0.017 -0.409
```

... more lines ...

Output 14.9.1 shows the scatter plot of y against x that is generated by the following statements:

```
proc sgplot data=Difficult;
  title "Scatter Plot of Y versus X";
  scatter x=x y=y ;
run;
```

Output 14.9.1 Scatter Plot of Y versus X



The plot clearly shows that the variance of y values varies considerably over the range of x values—the variance is larger for x values around 0.2 and gets increasingly smaller as the x values get closer to 1. Givens and Hoeting (2005) discuss the difficulties of extracting a smooth pattern from such data. Consider the following model for y :

$$y(x) = \mu_x + \epsilon_x$$

where μ_x is a smooth trend component and ϵ_x is the observation noise with variance, $h(x)$, which changes with x : $\epsilon_x \sim N(0, h(x))$. It is known (Durbin and Koopman 2012, chap. 3, sect. 9 and chap. 8, sect. 5) that modeling the trend μ_x as a polynomial smoothing spline (for example, the way the growth curves are

modeled in “[Example 14.4: Smoothing of Repeated Measures for Longitudinal Data](#)” on page 814) and taking the variance function of the observation noise ϵ_x a constant results in a trend estimate that can be termed a fixed-bandwidth-smoother. The optimal bandwidth turns out to be a function of the signal-to-noise ratio: the ratio of the observation noise variance and the disturbance variance of the trend component. On the other hand, allowing the variance function of the observation noise to change with the x values results in a trend estimate that can be termed a variable-bandwidth-smoother. The rest of this example shows how to use the CSSM procedure to create a data-dependent variance function $h(x)$ and to extract the associated (variable-bandwidth) smooth trend from such data. Suppose that the (unknown) variance function $h(x)$ can be approximated as

$$h(x) = \exp\left(\sum_{i=1}^7 v_i \text{SplineBasis}_i(x)\right)$$

where $v_i, i = 1, 2, \dots, 7$ are unknown parameters and $\text{SplineBasis}_i(x), i = 1, 2, \dots, 7$, are the full set of cubic spline basis functions (B-splines) with four evenly spaced internal knots between the range of x values—essentially, four equispaced points between 0.0 and 1.0. Note that the number of basis functions in the full set, 7, is the sum of the number of internal knots, 4, and the degree of the polynomial, 3. The following statements create a data set, Combined, that contains the variables x and y , along with the desired spline basis functions (col1–col7) that are created by using the BSPLINE function in PROC IML:

```
proc iml;
  use difficult;
  /* read x and y from difficult into temp */
  read all var _num_ into temp;
  x = temp[,1];
  /* generate B-spline basis for a cubic spline
     with 4 evenly spaced internal knots in the x-range */
  bsp = bspline(x, 2, ., 4);
  Combined = temp || bsp;
  /* create a merged data set with x, y, and
     spline basis columns */
  create Combined var {x y col1 col2 col3 col4 col5 col6 col7};
  append from Combined;
quit;
```

The following statements load the Combined data into the table mylib.Combined:

```
data mylib.Combined;
  set Combined;
  index = _n_;
run;
```

The following statements specify and fit the desired model to the data:

```
proc cssm data=mylib.Combined opt(tech=dbldog);
  id x;
  obsindex index;
  /* parameters needed to define h(x) */
  parms v1-v7;
  /* defining h(x) */
  evar = exp(v1*col1 + v2*col2 + v3*col3 + v4*col4
            + v5*col5 + v6*col6 + v7*col7);
  /* defining the polynomial spline trend */
```

```

trend trend(ps(2));
/* defining the observation noise with variance h(x) */
irregular wn variance=evar;
model y = trend wn;
output out=mylib.For pdv;
run;

```

Output 14.9.2 shows the estimates of $v_i, i = 1, 2, \dots, 7$, and Output 14.9.3 shows the estimate of the disturbance variance associated with the polynomial spline trend that is specified in the TREND statement.

Output 14.9.2 Estimates of v1–v7

Estimates of Named Parameters			
Parameter	Estimate	Standard	
		Error	t Value
v1	-3.3024	1.5007	-2.20
v2	-0.8260	0.6187	-1.33
v3	-2.2335	0.4526	-4.93
v4	-3.1302	0.4117	-7.60
v5	-4.3056	0.4146	-10.38
v6	-6.9008	0.5884	-11.73
v7	-19.5136	2.3058	-8.46

Output 14.9.3 Estimate of the Disturbance Variance Associated with the Trend

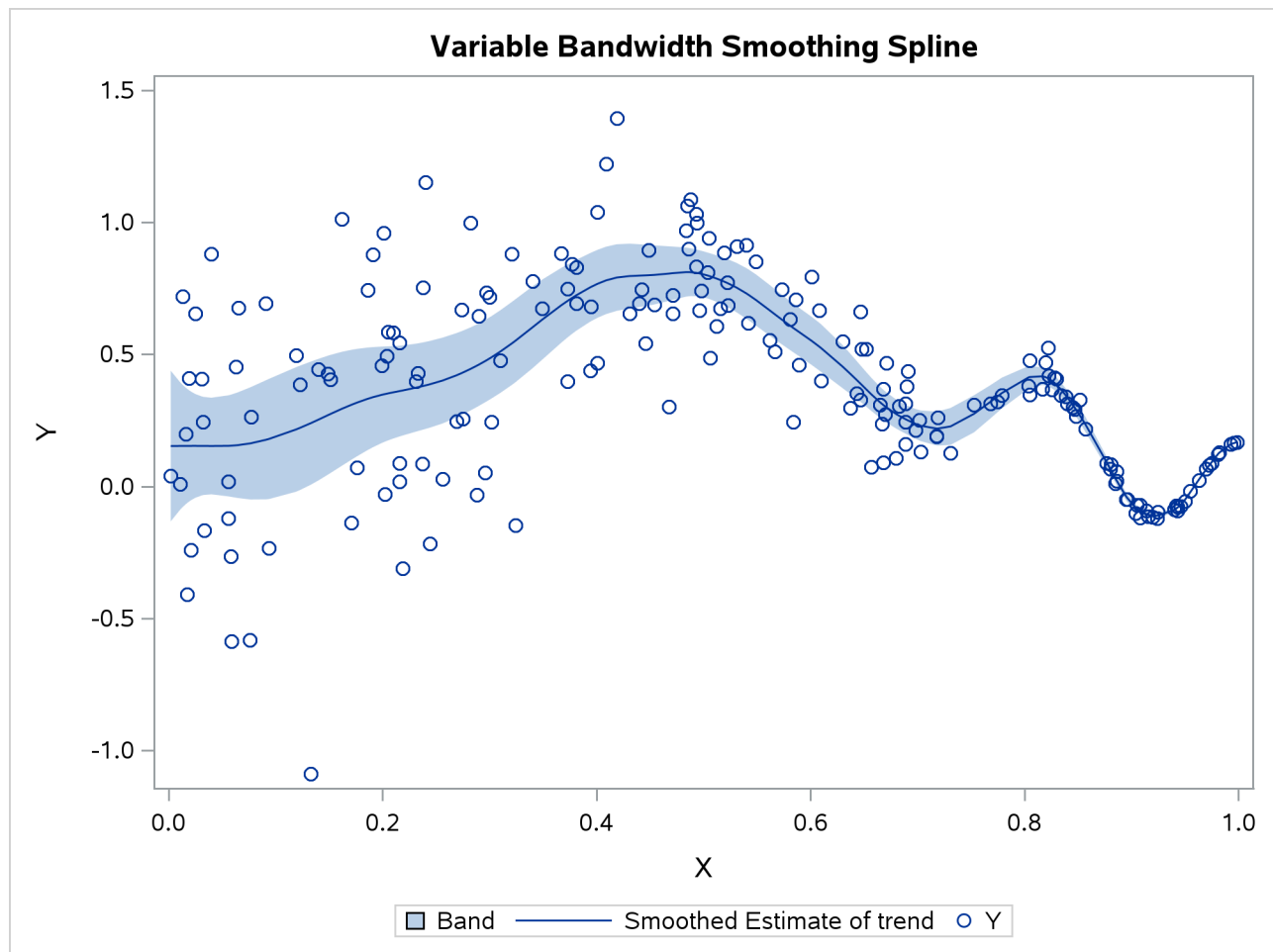
Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard	
				Error	t Value
trend	PS(2) Trend	Level Variance	338.6239	110.3375	3.07

The following statements produce a plot, shown in Output 14.9.4, of the fitted trend with 95% confidence band:

```

proc sgplot data=mylib.For;
  title "Variable Bandwidth Smoothing Spline";
  band x=x lower=smoothed_lower_trend
      upper=smoothed_upper_trend ;
  series x=x y=smoothed_trend;
  scatter x=x y=y;
run;

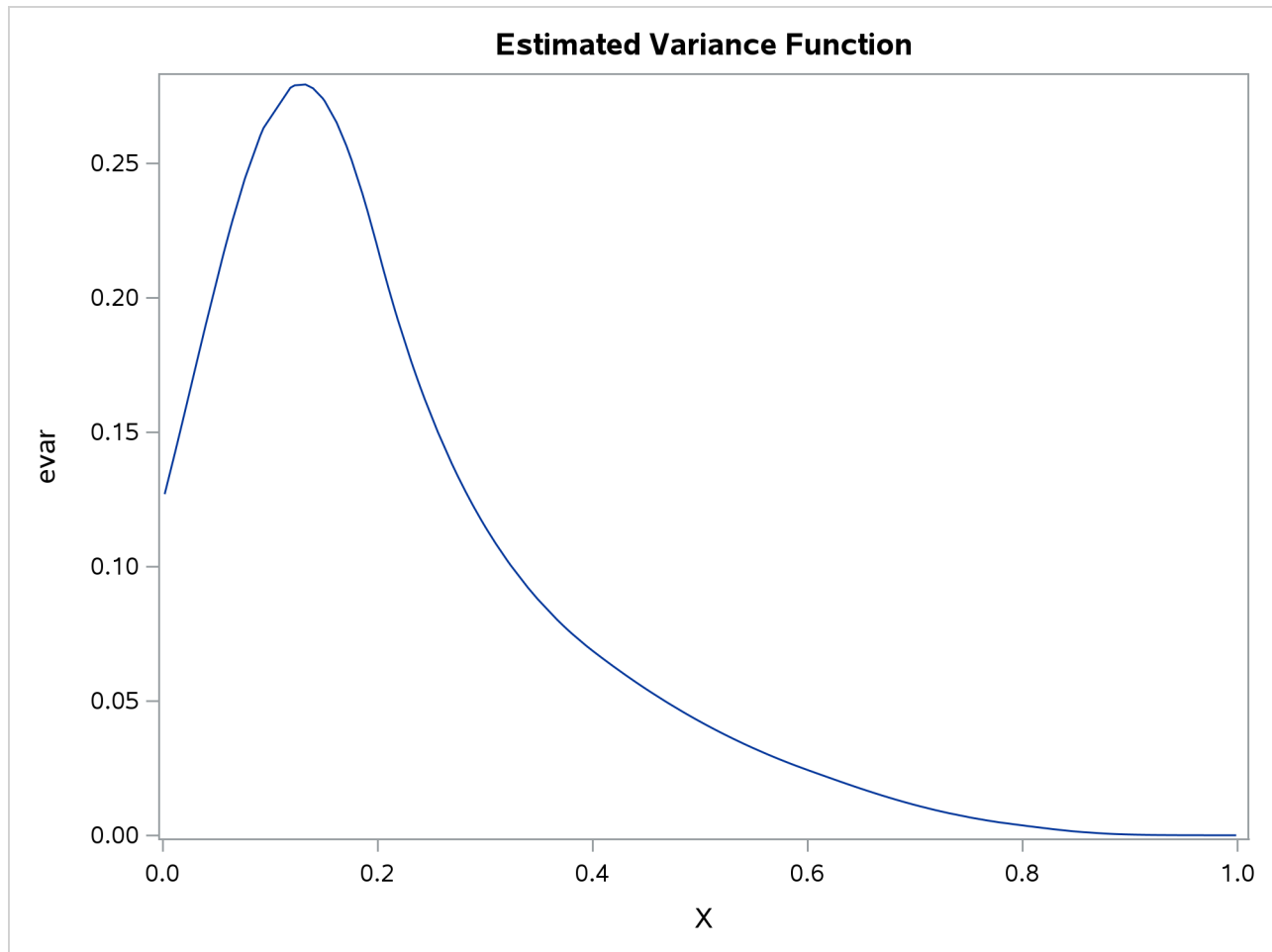
```

Output 14.9.4 Fitted Trend with 95% Confidence Band

Clearly the fitted curve tracks the data quite well. Lastly, [Output 14.9.5](#) (produced by using the following statements) shows the estimated variance function $h(x)$.

```
proc sgplot data=mylib.For;
  title "Estimated Variance Function";
  series x=x y=evar;
run;
```

As expected, the curve attains its peak at an x value around 0.18 and decays to nearly 0 as x values reach 1.0.

Output 14.9.5 Estimated Variance Function $h(x) = \exp(\sum_{i=1}^7 \hat{v}_i \text{SplineBasis}_i(x))$ 

Example 14.10: A Transfer Function Model for the Gas Furnace Data

This example describes how you can include components in your model that follow a transfer function model. Transfer function models, a generalization of distributed lag models, are useful for capturing the contributions from lagged values of the predictor series. Box and Jenkins popularized ARIMA models with transfer function inputs in their famous book (Box and Jenkins 1976). This example shows how you can specify an ARIMA model that is suggested in that book to analyze the data collected in an experiment at a chemical factory. The data set, called Series J by Box and Jenkins, contains sequentially recorded measurements of two variables: x , the input gas rate, and y , the output CO_2 . For the output CO_2 , Box and Jenkins suggest the model

$$y_t = \mu + f_t + \zeta_t$$

where μ is the intercept, ζ_t is a zero-mean noise term that follows a second-order autoregressive model (that is, $\zeta_t \sim \text{AR}(2)$), and f_t follows a transfer function model

$$f_t = \frac{(\gamma_1 B^3 + \gamma_2 B^4 + \gamma_3 B^5)}{(1 - \delta B)} x_t$$

The model for f_t is specified by using the ratio of two polynomials in the backshift operator B . Alternatively, this model can also be described as follows:

$$f_t = \delta f_{t-1} + \gamma_1 x_{t-3} + \gamma_2 x_{t-4} + \gamma_3 x_{t-5}$$

In this alternate form, it is easy to see that the equation for f_t can also be seen as a state evolution equation for a one-dimensional state with a (1×1) transition matrix δ and state regression variables x_{t-3} , x_{t-4} , and x_{t-5} (lagged values of x). This state equation has no disturbance term.

The following statements define the data set `Seriesj`. The variables `x3`, `x4`, and `x5`, which denote the appropriately lagged values of `x`, are also created. These variables are used later in the `STATE` statement that is used to specify f_t .

```
data Seriesj;
  input x y @@;
  label x = 'Input Gas Rate'
        y = 'Output CO2';
  x3 = lag3(x);
  x4 = lag4(x);
  x5 = lag5(x);
  obsIndex = _n_;
  label obsIndex = 'Observation Index';
datalines;
-0.109 53.8 0.000 53.6 0.178 53.5 0.339 53.5
0.373 53.4 0.441 53.1 0.461 52.7 0.348 52.4
0.127 52.2 -0.180 52.0 -0.588 52.0 -1.055 52.4

... more lines ...
```

The following statements load the `Seriesj` data into the table `mylib.Seriesj`:

```
/* Load data into mylib */
data mylib.Seriesj;
  set Seriesj(firstobs=6);
run;
```

The following CSSM procedure statements carry out the modeling of y , the output CO_2 , according to the preceding model:

```
proc cssm data=mylib.Seriesj;
  id obsIndex;
  parms delta /lower=-0.9999 upper=0.9999;
  state tfstate(1) T(g)=(delta) W(g)=(x3 x4 x5) a1(1) checkbreak;
  comp tfinput = tfstate[1];
  trend ar2(arma(p=2)) ;
  intercept = 1;
  model y = intercept tfinput ar2 ;
  eval modelCurve = intercept + tfinput;
  forecast out=mylib.For;
run;
```

The coefficient of the denominator polynomial, δ , is specified in a `PARMS` statement. It is constrained to be less than 1 in magnitude, which ensures that the transfer function term does not have explosive growth. The transfer function model for f_t is specified in a `STATE` statement that defines a one-dimensional state

named `tfstate`. In this statement, the transition matrix (which contains only one element, δ) is specified by using the `T(g)=` option; the state regression variables `x3`, `x4`, and `x5` are specified by using the `W(g)=` option; and the `CHECKBREAK` option is used to turn on the search for unexpected changes in the behavior of f_t . The `COV` option is absent from this `STATE` statement because the disturbance term is absent from the state equation for f_t . Moreover, because nothing can be assumed about the initial condition of this state equation, it is taken to be diffuse (as signified by the `A1` option). Note that the first five observations of the input data set, `Seriesj`, are excluded from the analysis to ensure that the state regression variables `x3`, `x4`, and `x5` do not contain any missing values. The component that is associated with `tfstate`, named `tfinput`, is specified in a `COMPONENT` statement that follows the `STATE` statement. A zero-mean, second-order autoregressive noise term, named `ar2`, is specified by using a `TREND` statement. Next, a constant regression variable, `intercept`, is defined to be used in the `MODEL` statement to capture the intercept term μ . Finally, the model specification is completed by specifying the response variable, `y`, and the three right-hand terms in the `MODEL` statement. Next, an `EVAL` statement is used to specify a component, `modelCurve`, which is the sum of the intercept and the transfer function input ($\mu + f_t$). The `modelCurve` component represents the structural part of the model and is defined only for output purposes: its estimate is output (along with the estimates of other components) to the data set that is specified in the `OUT=` option of the `OUTPUT` statement.

Note that the modeling of output CO_2 according to this model is also illustrated in “Example 7.3: Model for Series J Data from Box and Jenkins” (Chapter 7, *SAS/ETS User’s Guide*) of the PROC ARIMA documentation (see Chapter 7, “ARIMA Procedure” (*SAS/ETS User’s Guide*)). The ARIMA procedure handles the computation of the transfer function f_t slightly differently than the way it is estimated by the CSSM procedure. However, despite this algorithmic difference in the modeling procedures, for this example the estimated parameters agree quite closely (barring the sign conventions that are used to specify the model parameters).

Output 14.10.1 shows the estimate of μ , the intercept in the model. Output 14.10.2 shows the estimate of δ , the coefficient in the denominator polynomial of the transfer function. Output 14.10.3 shows the regression estimates of the state regression variables `x3`, `x4`, and `x5` (which correspond to the coefficients of the numerator polynomial). Output 14.10.4 shows the estimates of the parameters of the AR(2) noise term.

Output 14.10.1 Estimate of μ

Regression Parameter Estimates					
		Standard			
Response	Regressor	Estimate	Error	t Value	Pr > t
y	intercept	53.3594	0.1449	368.20	<.0001

Output 14.10.2 Estimate of δ , the Coefficient of the Denominator Polynomial in the Transfer Function

Estimates of Named Parameters			
		Standard	
Parameter	Estimate	Error	t Value
delta	0.5482	0.0398	13.78

Output 14.10.3 Regression Estimates of the State Regression Variables x3, x4, and x5

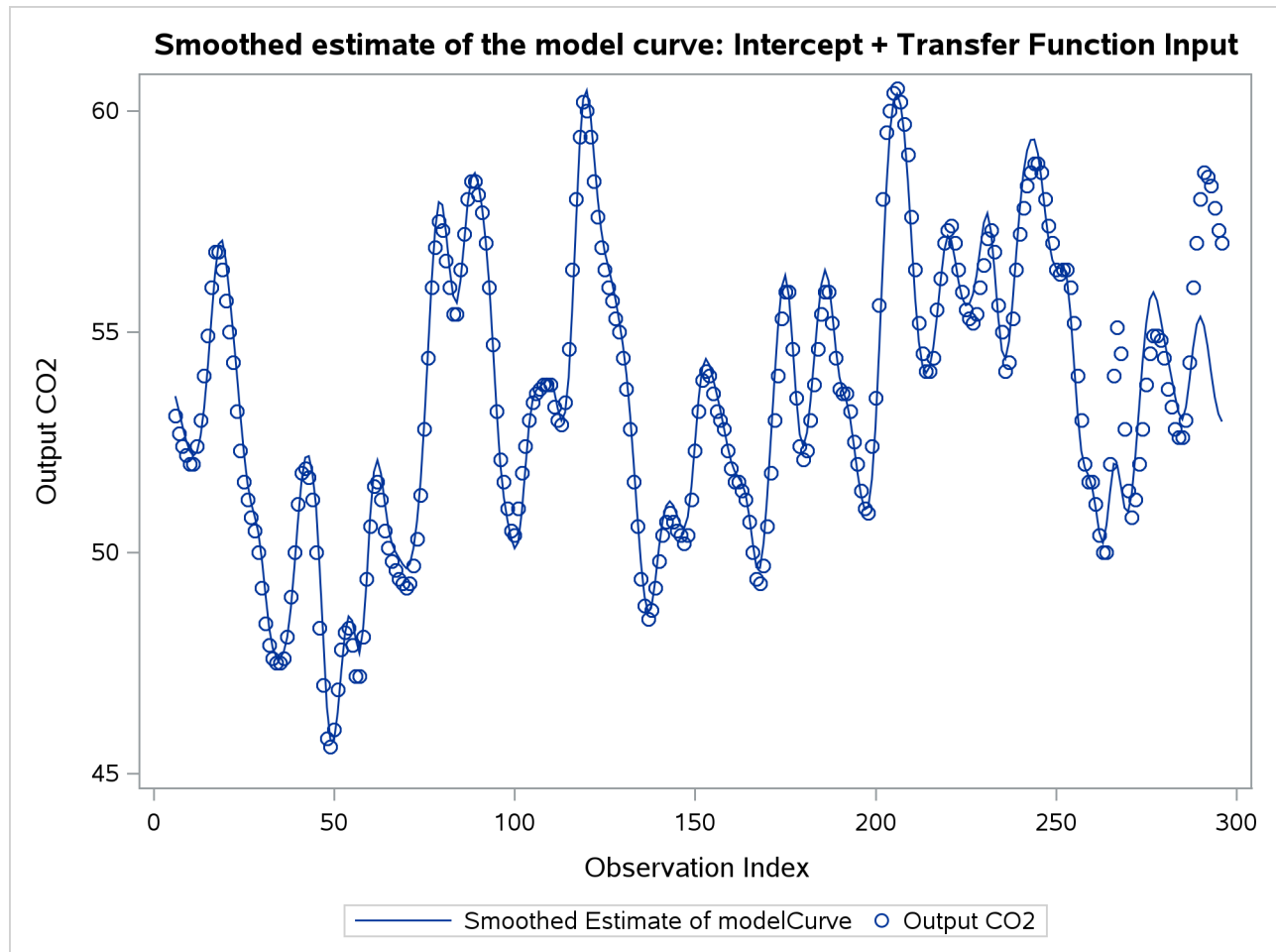
Estimate of the State Equation Regression Vector					
State	Element		Standard		
	Index	Estimate	Error	t Value	Pr > t
tfstate	1	-0.5299	0.0743	-7.13	<.0001
tfstate	2	-0.3801	0.1022	-3.72	0.0002
tfstate	3	-0.5194	0.0743	-6.99	<.0001

Output 14.10.4 Estimates of the Parameters of the Autoregressive Term

Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
ar2	ARMA Trend	Error Variance	0.0581	0.0049	11.96
ar2	ARMA Trend	AR_1	1.5319	0.0470	32.60
ar2	ARMA Trend	AR_2	-0.6291	0.0497	-12.65

The following statements produce a time series plot of the estimate of modelCurve—that is, the estimate of the structural part of the model ($\mu + f_t$)—along with the scatter plot of the observed values of the output CO₂. The plot, shown in [Output 14.10.5](#), seems to indicate that the model captures the relationship between the input gas rate and the output CO₂ quite well, at least up to the observation index 250.

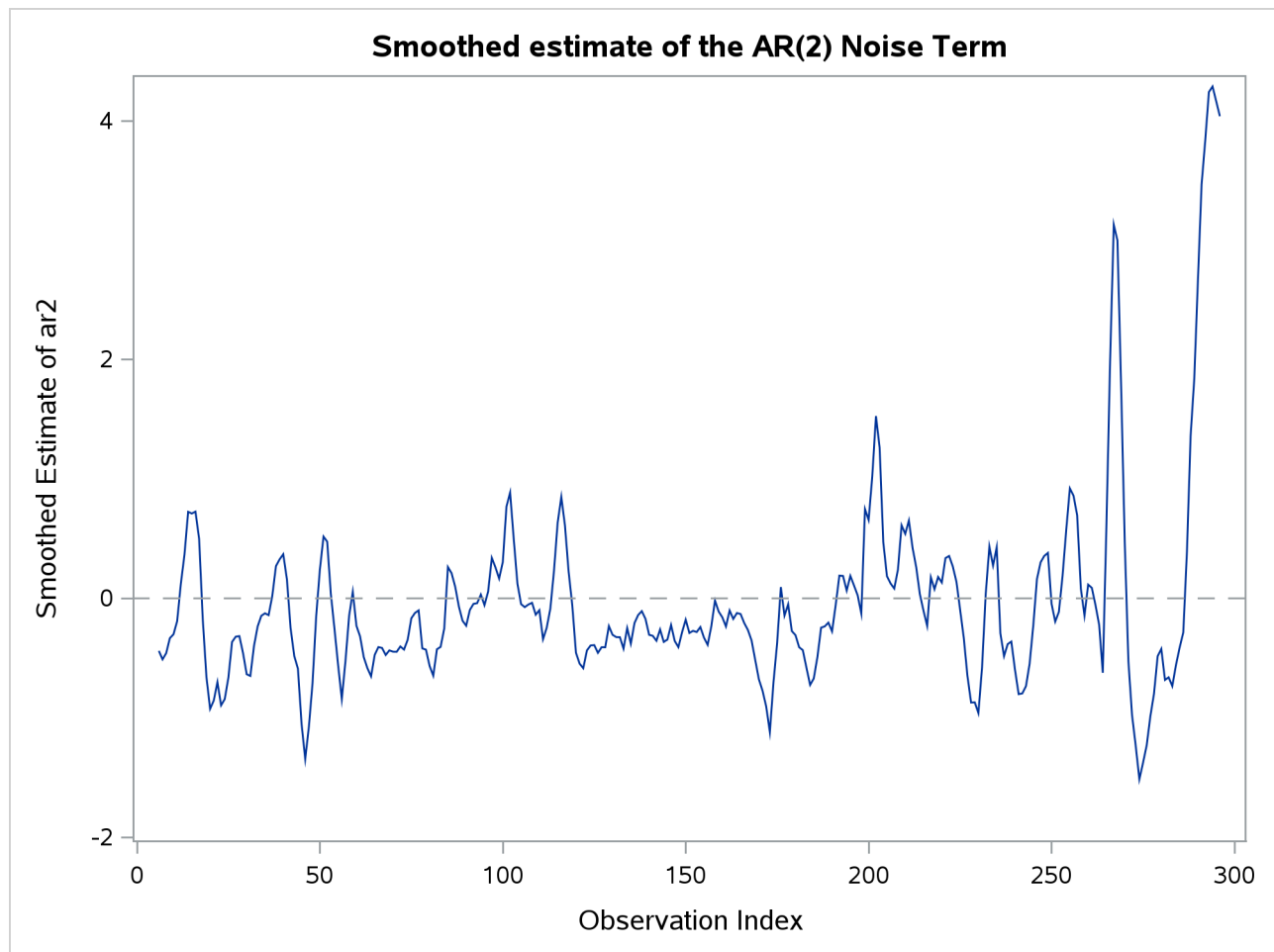
```
proc sgplot data=mylib.For;
  title "Smoothed estimate of the model curve: Intercept + Transfer Function Input";
  series x=obsIndex y=smoothed_modelCurve;
  scatter x=obsIndex y=y;
run;
```

Output 14.10.5 Smoothed Estimate of $\mu + f_t$ 

The plot shown in [Output 14.10.6](#) shows the estimate of the noise term, ar_2 , which is produced by the following statements:

```
proc sgplot data=mylib.For;
  title "Smoothed estimate of the AR(2) Noise Term";
  series x=obsIndex y=smoothed_ar2;
  refline 0 / axis=y lineattrs=GraphReference(pattern = Dash);
run;
```

If you compare the scales of these two plots, it appears that the noise term is relatively small and that most of the variation in output CO₂ can be explained by the structural part of the model. It does, however, appear that the model fit deteriorates toward the latter part of the sample. The structural break analysis summary, shown in [Output 14.10.7](#), indicates strong evidence of structural break in the behavior of f_t at or near the observation index 264. Obviously, this type of structural break analysis can be quite useful in industrial quality-control applications.

Output 14.10.6 Smoothed Estimate of the AR(2) Noise Term**Output 14.10.7** Summary of Breaks in f_t

Elementwise Break Summary for tfstate			
Element			
ID	Index	Z Value	Pr > z
264.000000	1	-5.03	<.0001
199.000000	1	4.62	<.0001
198.000000	1	-3.15	0.0016

Example 14.11: Dynamic Panel Model for the Cigar Data

This example shows how you can use the CSSM procedure to specify and fit the so-called dynamic panel model, which is commonly used to analyze a panel of time series. Suppose that a panel of time series $y_{t,i}$ follows the model

$$y_{t,i} = \rho y_{(t-1),i} + \mu_i + \beta X_{t,i} + \zeta_t + \epsilon_{t,i}$$

where t denotes the time index (for example, $t = 1, \dots, T$); i denotes the panel index (for example, $i = 1, \dots, P$); ρ is the autoregression coefficient; μ_i denote the panel-specific intercepts; $X_{t,i}$ are observations on a regression variable with regression coefficient β (the same for all panels); ζ_t are unobserved, random time effects; and $\epsilon_{t,i}$ are the observation errors. The sequences ζ_t and $\epsilon_{t,i}$ are assumed to be independent, zero-mean Gaussian variables with variances σ_1^2 and σ_0^2 , respectively. This is an example of a dynamic panel model that contains one regressor variable. It is easy to formulate this model equation as a state equation with state α_t of size P —the number of panels. Taking $y_{t,i} = \alpha_t[i]$, it is easy to see that the states α_t evolve according to the equation

$$\alpha_{t+1} = \mathbf{T}\alpha_t + \mathbf{W}_{t+1}\beta + \eta_{t+1}$$

where $\mathbf{T} = \rho \mathbf{I}_P$ (a P -dimensional, diagonal matrix with all its diagonal elements equal to ρ); $\mathbf{W}_t = (\mathbf{X}_t \ \mathbf{I}_P)$ is a $P \times (1 + P)$ -dimensional matrix (in a block form) of state regression variables, where the first block is a column that includes all the values $X_{t,i}$ that are associated with a given time index (t) and the second block is a P -dimensional identity matrix; $\beta = (\beta \ \mu_1, \dots, \mu_P)'$ is the $(1 + P)$ -dimensional column vector of regression coefficients; and $\eta_t = (\zeta_t + \epsilon_{t,1}, \dots, \zeta_t + \epsilon_{t,P})'$ is a P -dimensional column vector of all the disturbances that are associated with time index t . Because ζ_t and $\epsilon_{t,i}$ are independent, the covariance matrix of η_t —for example, $\mathbf{Q}_t$ —is easy to calculate: $\mathbf{Q}_t[i, i] = \sigma_0^2 + \sigma_1^2$ and, for $i \neq j$, $\mathbf{Q}_t[i, j] = \sigma_1^2$. This formulation can be easily extended to multiple regression variables, such as r variables, by appropriately modifying the term that is associated with the state regression variables— $\mathbf{W}_t\beta$: the new $\mathbf{W}_t$ matrix becomes $P \times (r + P)$ -dimensional and the new regression vector β becomes $(r + P)$ -dimensional.

The cross-sectional data, Cigar, that are used in the section “Getting Started: CSSM Procedure” on page 722 are reused in this example. In order to use the CSSM procedure to perform the dynamic panel model-based analysis, the input data set must be reorganized so that it contains the variables that form the $P \times (r + P)$ -dimensional matrix $\mathbf{W}_t$. For the Cigar data, the number of panels $P = 46$ (the number of regions considered in the study), and the number of regression variables $r = 3$. Therefore, the input data set needs to be augmented by $46 * (3 + 46) = 2,254$ variables that constitute the matrix $\mathbf{W}_t = (\mathbf{X}_t \ \mathbf{I}_{46})$ —the first 46×3 -dimensional block $\mathbf{X}_t$ contains the values of the three regression variables, lprice, lndi, and lpimin, at a given time index (a particular year in this case). The following DATA steps accomplish this task in two steps. In the first step, the raw data that form the rows of the Cigar data set are read into a temporary data set, Tmp, such that all $6*46 = 276$ values that are associated with a given year (values of six variables—year, region, lsales, lprice, lndi, and lpimin for 46 panels in a given year) are read in a single row that consists of 276 columns. In the second step, the final input data set is formed by rearranging Tmp so that it contains the necessary variables in the proper order—year (the time index), region (the panel index), lsales (the response variable), and the variables that form the 46×49 -dimensional $\mathbf{W}$ matrix (w1, ..., w2254).

```
data Tmp;
    input u1-u276;
datalines;
63 1 4.54223 3.35341 7.3514 3.26194
```

```
63 2 4.82831 3.17388 7.5729 3.21487
63 3 4.63860 3.29584 7.3000 3.25037
```

```
... more lines ...
```

```
data cigar(keep=year region lsales w1-w2254);
  array wmat{46, 49} w1-w2254;
  array ivar{46, 6} u1-u276;
  set tmp;
  year = intnx( 'year', '1jan63'd, u1-63 );
  format year year.;
  do i=1 to 46;
    region = ivar[i, 2];
    lsales = ivar[i, 3];
    do j=1 to 46;
      do k=1 to 49;
        wmat[j,k] = 0;
        if k = j+3 then wmat[j,k] = 1;
        if k=1 then wmat[j,k] = ivar[j, 4];
        if k=2 then wmat[j,k] = ivar[j, 5];
        if k=3 then wmat[j,k] = ivar[j, 6];
      end;
    end;
    output;
  end;
run;
```

The following statements load the Cigar data into the table mylib.Cigar:

```
/* Load data into mylib */
data mylib.Cigar;
  set Cigar;
run;
```

The following statements specify and fit the dynamic panel model:

```
proc cssm data=mylib.Cigar opt (tech=dbldog maxiter=75);
  id year interval=year;
  obsindex region;
  parms rho / lower=-0.9999 upper=0.9999;
  parms sigma0 sigma1 / lower=1.e-8;
  array RegionArray{46} region1-region46;
  do i=1 to 46;
    RegionArray[i] = (region=i);
  end;
  array cov{46,46};
  do i=1 to 46;
    do j=1 to 46;
      if(i=j) then cov[i,j] = sigma0 + sigma1;
      else cov[i,j] = sigma1;
    end;
  end;
  state panelState(46) T(I)=(rho) W(g)=(w1-w2254)
    cov(g)=(cov) al(46) checkbreak;
```

```

    comp dynPanel = (RegionArray)*panelState;
    model lsales = dynPanel;
    output out=mylib.for1 press;
run;

```

The estimates of the regression coefficients and the regional intercepts, which are all statistically significant, are shown in [Output 14.11.1](#). In particular, the estimated coefficients of `lprice`, `lndi`, and `lpimin`, are -0.26 , 0.13 , and 0.07 , respectively.

Output 14.11.1 Estimates of β_1 , β_2 , β_3 and the Regional Intercepts

Estimate of the State Equation Regression Vector					
State	Element Index	Estimate	Standard Error	t Value	Pr > t
panelState	1	-0.2627	0.0178	-14.79	<.0001
panelState	2	0.1340	0.0130	10.30	<.0001
panelState	3	0.0748	0.0198	3.78	0.0002
panelState	4	0.4265	0.0581	7.35	<.0001
panelState	5	0.3825	0.0605	6.32	<.0001
panelState	6	0.4425	0.0582	7.61	<.0001
panelState	7	0.3471	0.0631	5.50	<.0001
panelState	8	0.3686	0.0635	5.81	<.0001
panelState	9	0.4357	0.0614	7.10	<.0001
panelState	10	0.3753	0.0655	5.73	<.0001
panelState	11	0.4249	0.0606	7.01	<.0001
panelState	12	0.4185	0.0604	6.92	<.0001
panelState	13	0.3824	0.0602	6.35	<.0001
panelState	14	0.3942	0.0644	6.12	<.0001
panelState	15	0.4154	0.0626	6.64	<.0001
panelState	16	0.3961	0.0610	6.49	<.0001
panelState	17	0.3765	0.0618	6.10	<.0001
panelState	18	0.4528	0.0608	7.44	<.0001
panelState	19	0.4316	0.0586	7.36	<.0001
panelState	20	0.4357	0.0601	7.25	<.0001
panelState	21	0.3771	0.0639	5.90	<.0001
panelState	22	0.3939	0.0629	6.26	<.0001
panelState	23	0.4122	0.0621	6.64	<.0001
panelState	24	0.3949	0.0605	6.52	<.0001
panelState	25	0.4386	0.0565	7.77	<.0001
panelState	26	0.4118	0.0627	6.57	<.0001
panelState	27	0.3898	0.0604	6.45	<.0001
panelState	28	0.3818	0.0613	6.23	<.0001
panelState	29	0.4343	0.0632	6.87	<.0001
panelState	30	0.4619	0.0625	7.39	<.0001
panelState	31	0.3730	0.0636	5.86	<.0001
panelState	32	0.3784	0.0589	6.43	<.0001
panelState	33	0.3825	0.0625	6.12	<.0001
panelState	34	0.3784	0.0598	6.32	<.0001
panelState	35	0.4093	0.0628	6.52	<.0001
panelState	36	0.4155	0.0597	6.96	<.0001
panelState	37	0.3960	0.0615	6.44	<.0001
panelState	38	0.4075	0.0602	6.77	<.0001
panelState	39	0.4045	0.0586	6.91	<.0001
panelState	40	0.3918	0.0599	6.55	<.0001
panelState	41	0.4350	0.0608	7.15	<.0001
panelState	42	0.4007	0.0602	6.65	<.0001
panelState	43	0.3196	0.0597	5.36	<.0001
panelState	44	0.4337	0.0609	7.12	<.0001
panelState	45	0.3790	0.0634	5.98	<.0001
panelState	46	0.3767	0.0618	6.10	<.0001
panelState	47	0.4392	0.0597	7.36	<.0001

Output 14.11.1 *continued*

Estimate of the State Equation Regression Vector					
State	Element Index	Estimate	Standard Error	t Value	Pr > t
panelState	48	0.3932	0.0603	6.51	<.0001
panelState	49	0.3938	0.0616	6.40	<.0001

Output 14.11.2 shows the estimates of the autoregression coefficient ρ , the observation error variance σ_0^2 , and the variance of the time effect (variance of ζ) σ_1^2 .

Output 14.11.2 Estimates of ρ , σ_0^2 , and σ_1^2

Estimates of Named Parameters			
Parameter	Estimate	Standard Error	t Value
rho	0.8317	0.0124	66.89
sigma0	0.0012	0.0000	25.08
sigma1	0.0002	0.0001	3.22

Finally, you can compare the fit of the dynamic panel model with the fit of the model that is discussed in the section “Getting Started: CSSM Procedure” on page 722. Output 14.11.3 shows the likelihood-based information criteria for the dynamic panel model, and Output 14.11.4 shows the same information for the other model.

Output 14.11.3 Likelihood-Based Information Criteria: Dynamic Panel Model

Information Criteria		
Statistic	Diffuse Likelihood	Profile Likelihood
AIC (lower is better)	-4733	-4856
BIC (lower is better)	-4717	-4344
AICC (lower is better)	-4733	-4841
HQIC (lower is better)	-4727	-4665
CAIC (lower is better)	-4714	-4246

Output 14.11.4 Likelihood-Based Information Criteria: Getting Started Example

Information Criteria		
Statistic	Diffuse Likelihood	Profile Likelihood
AIC (lower is better)	-4488	-4145
BIC (lower is better)	-4478	-3638
AICC (lower is better)	-4488	-4130
HQIC (lower is better)	-4484	-3955
CAIC (lower is better)	-4476	-3541

Similarly, Output 14.11.5 shows fit criteria based on the delete-one cross validation error for the dynamic panel model, and Output 14.11.6 shows the same information for the other model.

Output 14.11.5 Delete-One Cross Validation Criteria: Dynamic Panel Model

Cross Validation Error Criteria			
		Generalized	
Variable	N	PRESS	Cross-Validation
Isales	1380	1.115	5.628E-07

Output 14.11.6 Delete-One Cross Validation Criteria: Getting Started Example

Cross Validation Error Criteria			
		Generalized	
Variable	N	PRESS	Cross-Validation
Isales	1380	1.29	6.181E-07

On the basis of both these considerations, the dynamic panel model appears to provide a better fit for the Cigar data than the model that is fit in the section “[Getting Started: CSSM Procedure](#)” on page 722.

Example 14.12: Long-Term Temperature Trends Based on a Multivariate Model

In a presentation by Ansley and de Jong (2015), three monthly time series are jointly modeled to obtain long-term—several decades long—temperature predictions for certain regions of the northern hemisphere. This example shows how you can specify and fit the final model that this presentation proposed. The following data set, `Temp`, contains four variables: `date` dates the monthly observations; `UAH` contains monthly satellite global temperature readings, starting in December 1978; `CRU` contains monthly temperature data, starting in January 1850 (from a different source); and `GISS` contains monthly temperature data, starting in January 1880 (from yet another source). All these temperature data are scaled suitably so that the numbers represent temperature readings in centigrade.

```
data Temp;
input UAH CRU GISS @@;
date = intnx('month', '01jan1850'd, _n_-1);
format date date.;
datalines;
.      8.243      .
.      9.733      .

... more lines ...
```

The following statements produce scatter plots of these three series in a single graph:

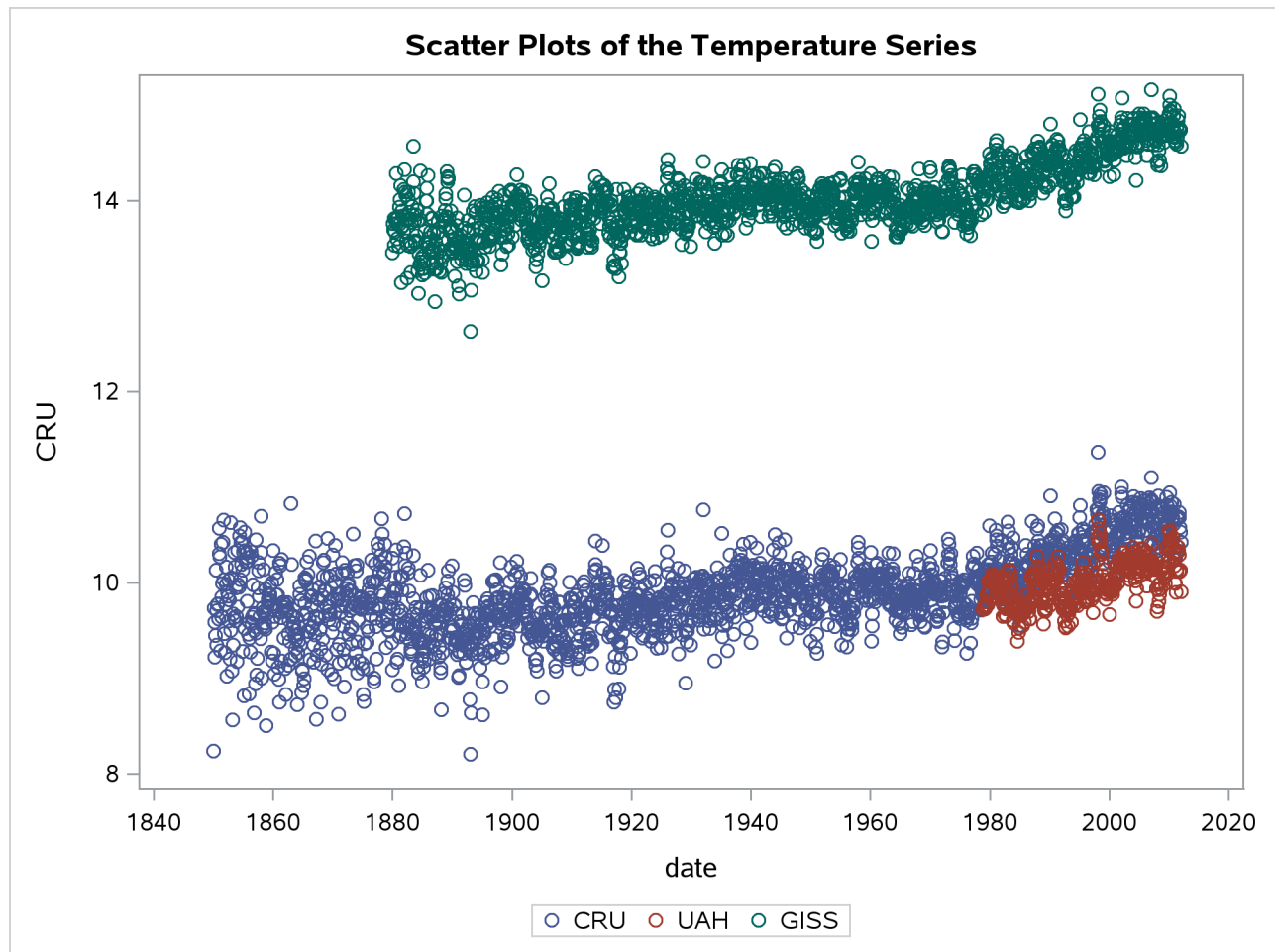
```
proc sgplot data=Temp;
  title "Scatter Plots of the Temperature Series";
  scatter x=date y=cru;
  scatter x=date y=uah ;
  scatter x=date y=giss;
run;
```

Output 14.12.1 shows the resulting graph. As already noted, these three series start at different points in the past. However, they all end at the same time: they all have measurements until January 2012, which is the last month in the data set. The mean levels of these series are different: the GISS measurements are generally larger than CRU and UAH by about 4 degrees. In addition, the variability in the CRU values seems to decrease with time (this is more apparent when the series is plotted by itself). The goal of the analysis is to use these data to make long-term predictions about future temperature levels. The following statements append 1,200 missing measurements to Temp, so that the model fitted by using the CSSM procedure can be extrapolated to obtain temperature forecasts 100 years in the future:

```
data append(keep=date cru giss uah);
  do i=1 to 1200;
    cru = .; giss=.; uah=.;
    date = intnx('month','01jan2012'D, i);
    format date monyy7.;
    output;
  end;
run;

proc append base=Temp data=Append; run;
```

Output 14.12.1 Scatter Plots of CRU, UAH, and GISS



Ansley and de Jong propose a parsimonious model that links these three time series. It can be described as

$$\begin{aligned}\text{GISS}_t &= \mu_t + a \zeta_t + a r_1 \epsilon_{1t} \\ \text{CRU}_t &= \beta_{cru} + \mu_t + a \zeta_t + a \epsilon_{2t} \\ \text{UAH}_t &= \beta_{uah} + \mu_t + a \zeta_t + a r_3 \epsilon_{3t}\end{aligned}$$

where β_{cru} and β_{uah} are intercepts that are associated with CRU and UAH, respectively; μ_t is an integrated random walk trend; ζ_t is a zero-mean, autoregressive noise term (which is scaled by an unknown scaling factor a); and ϵ_{it} ($i = 1, 2, 3$) are independent observation errors with different variances that are also scaled suitably. Note that the trend μ_t and the autoregressive noise term ζ_t are shared by the models of the three series, and, for identification purposes, the intercept for GISS is taken to be zero. In addition, the model parameters are assumed to be interrelated and are parameterized in a particular way (which leads to fewer parameters to estimate, and their relative scaling helps in parameter estimation). This special parameterization can be expressed as a function of seven basic parameters: *loga1*, *logr1*, *logr3*, *logsigma*, *b*, *c*, and *rhoParm* (this naming convention is different from that used by Ansley and de Jong (2015)).

Let $\sigma^2 = \exp(2\text{logsigma})$, and let the scaling factors $a = \exp(\text{loga1})$, $r_1 = \exp(\text{logr1})$, and $r_3 = \exp(\text{logr3})$. Then the model parameters can be described as follows:

- The parameters that are associated with the autoregression ζ_t : the damping factor $\rho = \frac{\exp(\text{rhoParm})-1}{\exp(\text{rhoParm})+1}$, the variance of the disturbance term $= a^2\sigma^2$, and the variance of the initial state $= a^2\sigma^2/(1 - \rho^2)$
- The parameters that are associated with the integrated random walk trend: the variance of the disturbance term in the slope equation $= \sigma^2$
- The variance of the observation errors for GISS $= a^2r_1^2\sigma^2$
- The variance of the observation errors for UAH $= a^2r_3^2\sigma^2$
- The variance of the observation errors for CRU is taken to be time-varying with the following form: for $t \leq \text{nobs}$,

$$\sigma_{2,t}^2 = a^2\sigma^2 \exp(2b + 2c \ t/\text{nobs})$$

where $\text{nobs} = 1,945$ (the number of observations in the unappended data set). For $t > 1,945$, it is fixed at its last value: $\sigma_{2,t}^2 = a^2\sigma^2 \exp(2b + 2c)$.

The following DATA step adds an observation index, *tindex*, to *Temp*, which is used in the CSSM procedure to define the time-varying observation error variance for CRU:

```
data mylib.temp;
  set temp;
  tindex = _n_;
run;
```

The following statements fit the preceding model to the *Temp* data:

```
ods output RegressionEstimates=regEst;
proc cssm data=mylib.Temp;
  id date interval=month;
  parms loga1 logr1 logr3 logsigma / upper=50;
```

```

parms b=0 c=0 / upper=50;
parms rhoParm;
rho = (exp(rhoParm)-1)/(exp(rhoParm)+1);
sigmaSq = exp(2*logsigma);
initSigmaSq = sigmaSq/(1-rho*rho);
a1 = exp(loga1);
a1Sq = a1*a1;
r1sq = exp(2*logr1);
r3sq = exp(2*logr3);
giss_var = a1Sq*r1sq*sigmaSq;
nobs=1945;
if tindex <= nobs then
    cru_var = a1Sq*exp(2*b + 2*c*tindex/nobs)*sigmaSq;
else cru_var = a1Sq*exp(2*b + 2*c)*sigmaSq;
uah_var = a1Sq*r3sq*sigmaSq;
UAH_Intercept=1.0;
CRU_Intercept=1.0;
trend level(11) variance=0 slopevar=sigmaSq;
state auto(1) T(g)=(rho) cov(g)=(sigmaSq) cov1(g)=(initSigmaSq);
comp auto_common = auto*(a1);
state wn(3) type=wn cov(d)=(giss_var cru_var uah_var);
comp wn_giss = wn[1];
comp wn_cru = wn[2];
comp wn_uah = wn[3];
model GISS = level auto_common wn_giss;
model CRU = CRU_Intercept level auto_common wn_cru;
model UAH = UAH_Intercept level auto_common wn_uah;
comp slope = level_state_*(0 1);
output out=mylib.For pdv press;
run;

```

Output 14.12.2 shows the estimated intercepts $\hat{\beta}_{cru}$ and $\hat{\beta}_{uah}$. As expected, they are quite close (see the scatter plots of CRU and UAH in Output 14.12.1).

Output 14.12.2 Estimated Intercepts for CRU and UAH

Regression Parameter Estimates					
Response	Regressor	Estimate	Standard		
			Error	t Value	Pr > t
CRU	CRU_Intercept	-4.0944	0.0036	-1139.0	<.0001
UAH	UAH_Intercept	-4.4756	0.0067	-666.54	<.0001

The estimates of the basic parameters that underlie the model parameters are shown in Output 14.12.3.

Output 14.12.3 Estimates of Basic Model Parameters

Estimates of Named Parameters			
Parameter	Estimate	Standard	
		Error	t Value
loga1	7.7763	0.3906	19.91
logr1	-1.0049	0.0884	-11.36
logr3	-0.2304	0.0453	-5.09
logsigma	-9.6386	0.3893	-24.76
b	0.7373	0.0502	14.69
c	-1.4025	0.0690	-20.34
rhoParm	1.4294	0.0766	18.66

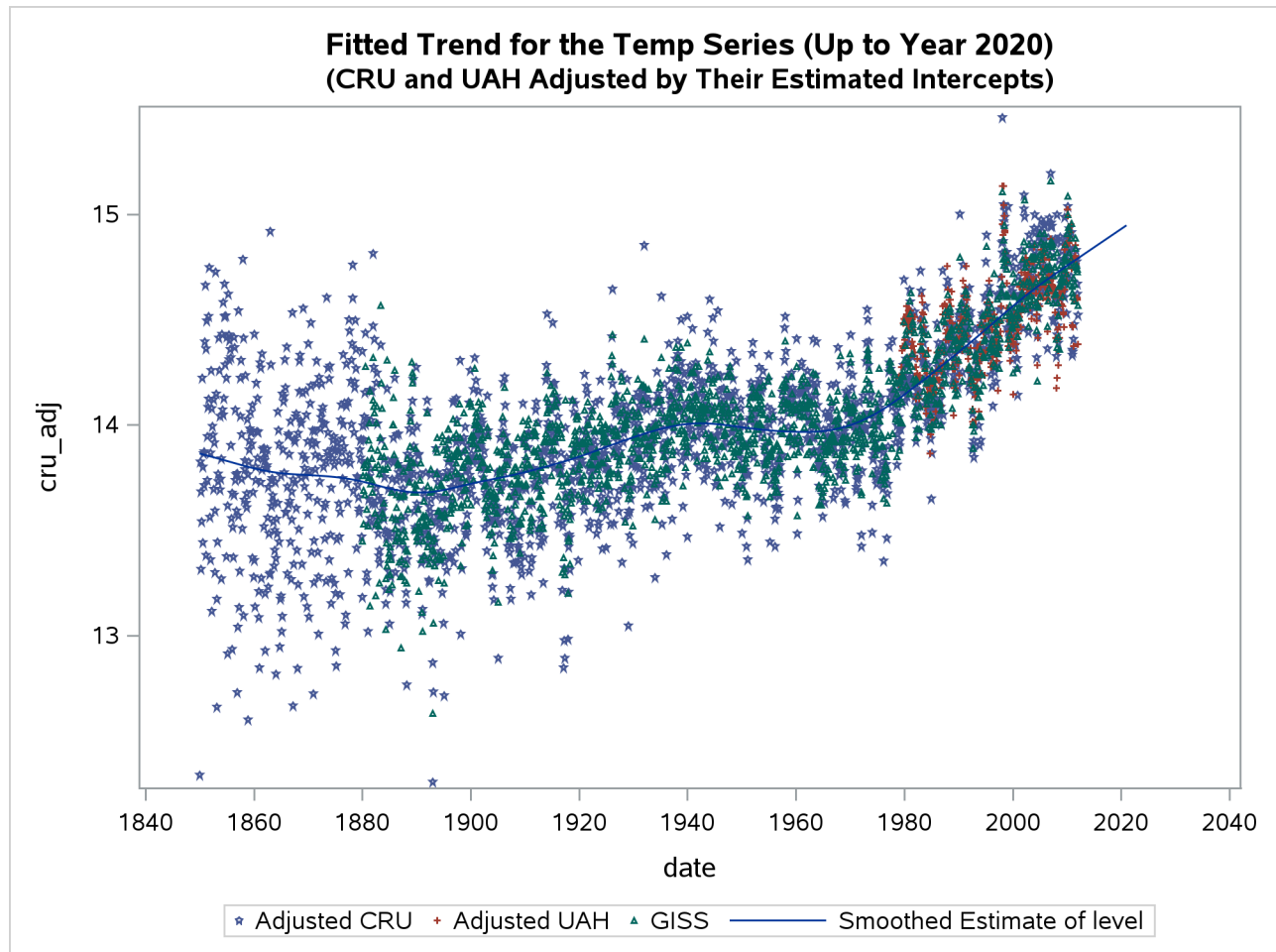
The following DATA steps add two variables ($CRU_ADJ = CRU - \hat{\beta}_{cru}$ and $UAH_ADJ = UAH - \hat{\beta}_{uah}$) to the output data set mylib.For. These adjusted versions of CRU and UAH have the same mean level as GISS—estimated μ_t .

```
data _NULL_;
  set regEst;
  if _n_ = 1 then call symput('intercept1',trim(left(estimate)));
  else call symput('intercept2',trim(left(estimate)));
run;
data for;
  set mylib.For;
  cru_adj = cru - &intercept1;
  uah_adj = uah - &intercept2;
run;
```

The following statements produce a graph that contains four plots: scatter plots of GISS, CRU_ADJ, and UAH_ADJ and a series plot of the estimated μ_t .

```
proc sgplot data=For;
  where date < '01feb2021'd;
  title "Fitted Trend for the Temp Series (Up to Year 2020) ";
  title2 "(CRU and UAH Adjusted by Their Estimated Intercepts) ";
  scatter x=date y=cru_adj / LEGENDLABEL="Adjusted CRU"
    MARKERATTRS=GraphData1(symbol=star size=3);
  scatter x=date y=uah_adj / LEGENDLABEL="Adjusted UAH"
    MARKERATTRS=GraphData2(symbol=plus size=3);
  scatter x=date y=giss /
    MARKERATTRS=GraphData3(symbol=triangle size=3);
  series x=date y=smoothed_level / MARKERATTRS=GraphData4;
run;
```

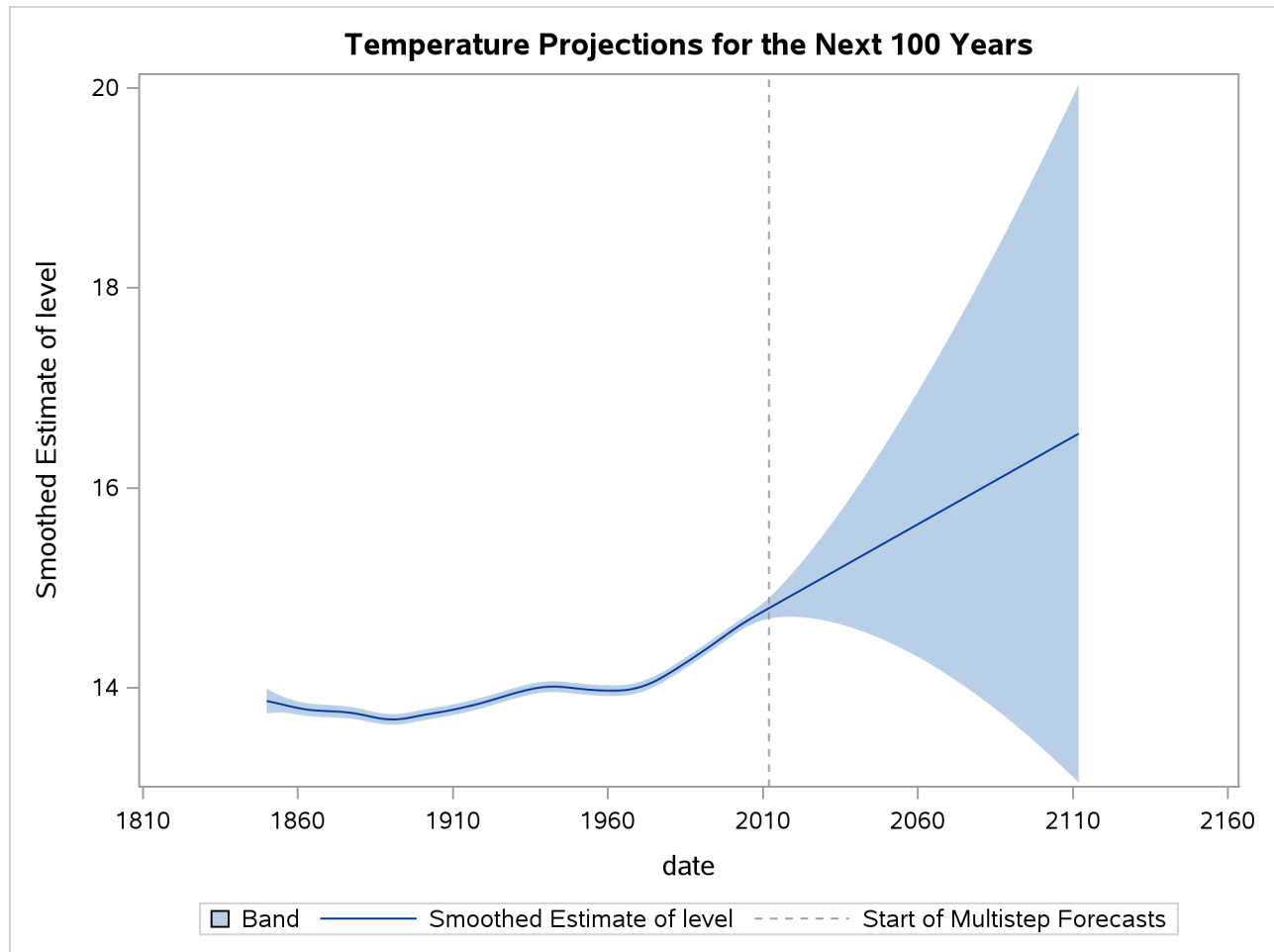
Output 14.12.4 shows the resulting graph. It shows that the estimated mean level μ_t tracks the observed data quite well.

Output 14.12.4 Fitted Trend μ_t 

The following statements produce the time series plot of the estimated μ_t along with the 95% confidence band:

```
proc sgplot data=For;
  title "Temperature Projections for the Next 100 Years";
  band x=date lower=smoothed_lower_level upper=smoothed_upper_level;
  series x=date y=smoothed_level;
  refline '01feb2012'd / axis=x lineattrs=(pattern=shortdash)
    LEGENDLABEL= "Start of Multistep Forecasts"
    name="Forecast Reference Line";
run;
```

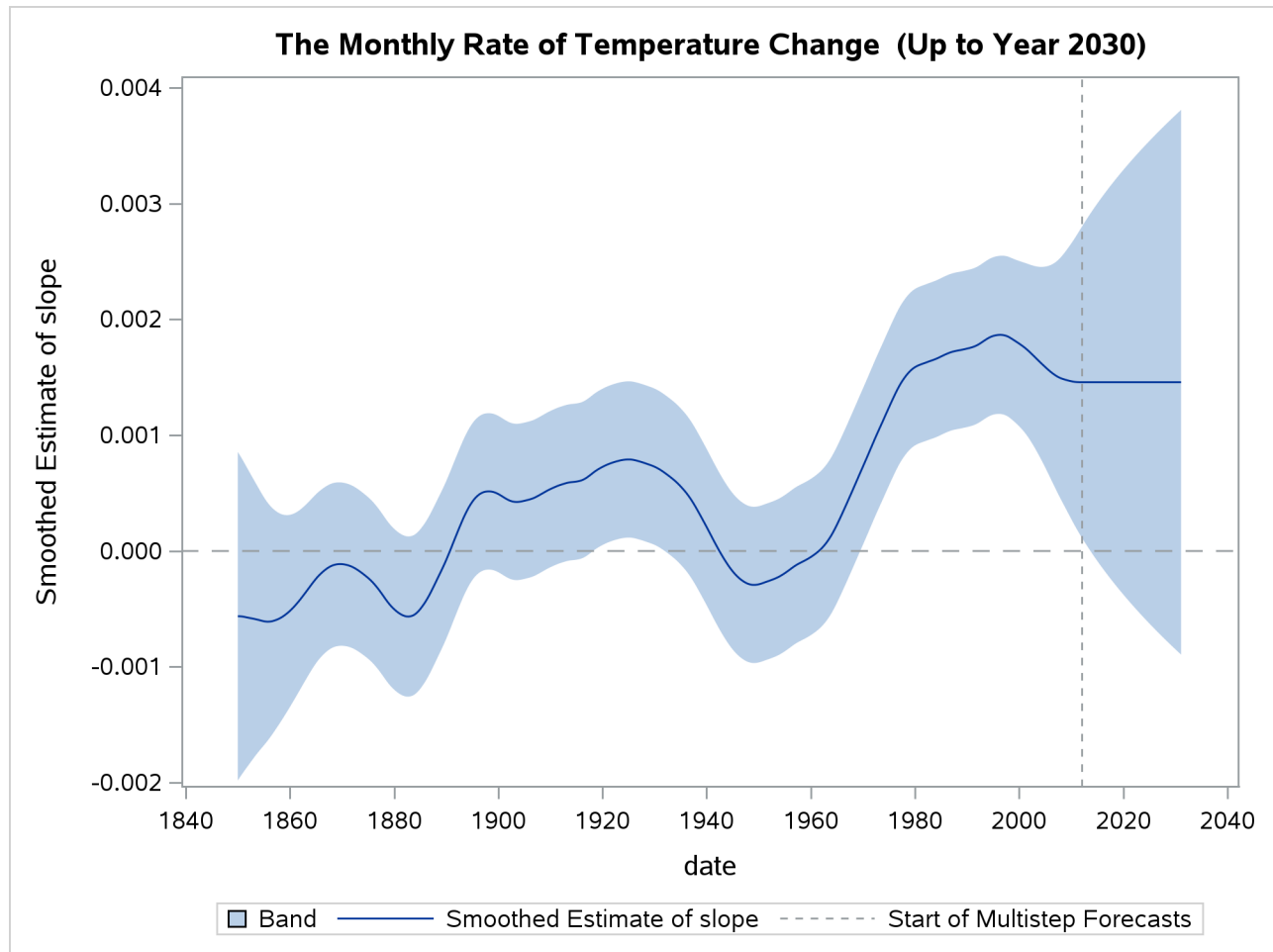
Output 14.12.5 shows the resulting graph.

Output 14.12.5 Long-Term Forecasts of μ_t 

The following statements produce a similar graph for the estimated slope of μ_t :

```
proc sgplot data=For;
  where date <= '01feb2031'd;
  title "The Monthly Rate of Temperature Change (Up to Year 2030)";
  band x=date lower=smoothed_lower_slope upper=smoothed_upper_slope;
  series x=date y=smoothed_slope;
  refline '01feb2012'd / axis=x lineattrs=(pattern=shortdash)
    LEGENDLABEL= "Start of Multistep Forecasts"
    name="Forecast Reference Line";
  refline 0 / axis=y lineattrs=(pattern=dash);
run;
```

Output 14.12.6 shows the resulting plot of the estimated slope of μ_t .

Output 14.12.6 Forecasts of the Slope of μ_t 

Based on the preceding analysis (see the plots of μ_t and its slope in [Output 14.12.5](#) and [Output 14.12.6](#)), it appears that there has been statistically significant warming over the last 10 years, but the warming does not appear to be accelerating.

Example 14.13: Bivariate Model for Sales of Mink and Muskrat Furs

This example considers a bivariate time series of logarithms of the annual sales of mink and muskrat furs by the Hudson's Bay Company for the years 1850–1911. These data have been analyzed previously by many authors, including Chan and Wallis (1978); Harvey (1989); Reinsel (1997). There is known to be a predator-prey relationship between the mink and muskrat species: minks are principal predators of muskrats. Previous analyses for these data generally conclude the following:

- An increase in the muskrat population is followed by an increase in the mink population a year later, and an increase in the mink population is followed by a decrease in the muskrat population a year later.
- Because muskrats are not the only item in the mink diet and because both mink and muskrat populations are affected by many other factors, the model must include additional terms to explain the year-to-year variation.

The analysis in this example, which loosely follows the discussion in Harvey (1989, chap. 8, sec. 8), also leads to similar conclusions. It begins by taking Harvey's model 8.8.8 (a and b), with autoregressive order one, as the starting model—that is, it assumes that the bivariate (mink, muskrat) process $\mathbf{Y}_t$ satisfies the following relationship:

$$\begin{aligned}\boldsymbol{\mu}_t &= \boldsymbol{\mu}_{t-1} + \boldsymbol{\beta} + \mathbf{v}_t \\ \mathbf{Y}_t &= \boldsymbol{\mu}_t + \boldsymbol{\Phi}\mathbf{Y}_{t-1} + \boldsymbol{\xi}_t\end{aligned}$$

This model postulates that $\mathbf{Y}_t$ can be expressed as a sum of three terms: $\boldsymbol{\mu}_t$, a bivariate trend that is modeled as a random walk with drift $\boldsymbol{\beta}$; $\boldsymbol{\Phi}\mathbf{Y}_{t-1}$, an AR(1) correction; and $\boldsymbol{\xi}_t$, a bivariate Gaussian white noise disturbance. It is assumed that the AR coefficient matrix $\boldsymbol{\Phi}$ is stable (that is, its eigenvalues are less than 1 in magnitude) and that the bivariate disturbances $\mathbf{v}_t$ (white noise associated with $\boldsymbol{\mu}_t$) and $\boldsymbol{\xi}_t$ are mutually independent.

The following statements load the furs data into the table mylib.Cigar:

```
/* Load data into mylib */
data mylib.furs;
  set furs;
run;
```

The following statements show how you can specify this model in the CSSM procedure:

```
proc cssm data=mylib.furs plots=residual;

  /* Specify the ID variable */
  id year interval=year;

  /* Define parameters */
  parms rho1 rho2/ lower=-0.9999 upper=0.9999;
  parms msd1 msd2 esd1 esd2 / lower=1.e-6;

  /* Specify the terms with lagged response variables */
  deplag LagsForMink(LogMink) LogMink(lags=1) LogMusk(lags=1);
  deplag LagsForMusk(LogMusk) LogMink(lags=1) LogMusk(lags=1);

  /* Specify the bivariate trend */
  array rwQ{2,2};
  rwQ[1,1] = msd1*msd1; rwQ[1,2] = msd1*msd2*rho1;
  rwQ[2,1] = rwQ[1,2]; rwQ[2,2]=msd2*msd2;
  state alpha(2) type=RW W(I) cov(g)=(rwQ);
  comp minkLevel = alpha[1];
  comp muskLevel = alpha[2];

  /* Specify the bivariate white noise */
  array wnQ{2,2};
  wnQ[1,1] = esd1*esd1; wnQ[1,2] = esd1*esd2*rho2;
  wnQ[2,1] = wnQ[1,2]; wnQ[2,2]=esd2*esd2;
  state error(2) type=WN cov(g)=(wnQ);
  comp minkWn = error[1];
  comp muskWn = error[2];

  /* Specify the observation equation */
  model LogMink = LagsForMink minkLevel minkWn;
```

```

model LogMusk = LagsForMusk muskLevel muskWn;

/* Specify an output data set to store component estimates */
output out=mylib.salesFor press;

run;

```

The different parts of the program are explained as follows:

- The PARMs statements define parameters that are used to form the elements of Σ_1 (the covariance of $\mathbf{v}_t$, the disturbance term in the bivariate level equation) and Σ_2 (the covariance of ξ_t , which is the bivariate white noise). Σ_1 is parameterized as (msd1*msd1 msd1*msd2*rho1; msd1*msd2*rho1 msd2*msd2). Σ_2 is similarly parameterized by using esd1, esd2, and rho2. In addition to ensuring that Σ_1 and Σ_2 are positive semidefinite, it turns out that this parameterization leads to an interpretable model at the end.
- The DEPLAG statements help define the terms that are associated with $\Phi \mathbf{Y}_{t-1}$.
- The remaining statements are self-explanatory.

Output 14.13.1 shows the estimate of the drift vector β in the equation of μ_t ($\mu_t = \mu_{t-1} + \beta + \mathbf{v}_t$).

Output 14.13.1 Estimate of the Drift Vector β

Estimate of the State Equation Regression Vector					
State	Element Index	Estimate	Standard Error	t Value	Pr > t
alpha	1	-0.0008	0.0323	-0.03	0.9798
alpha	2	0.0060	0.0258	0.23	0.8175

Clearly, both elements of β are statistically insignificant, and the μ_t equation can be simplified as $\mu_t = \mu_{t-1} + \mathbf{v}_t$. Next, Output 14.13.2 shows the estimates of the elements of Σ_1 , and Σ_2 , and Output 14.13.3 shows the estimates of the lag coefficients.

Output 14.13.2 Estimates of Σ_1 , and Σ_2

Estimates of Named Parameters			
Parameter	Estimate	Standard Error	t Value
rho1	0.8310	0.1377	6.03
rho2	-0.9999	1.6555	-0.60
msd1	0.2500	0.0354	7.06
msd2	0.1991	0.0592	3.36
esd1	0.0662	0.0597	1.11
esd2	0.1344	0.0527	2.55

Output 14.13.3 Estimates of Lag Coefficients (Elements of Φ)

Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
LagsForMink	Lag Coefficient Of LogMink	Lag[1]	-0.0011	0.1730	-0.01
LagsForMink	Lag Coefficient Of LogMusk	Lag[1]	0.3349	0.1369	2.45
LagsForMusk	Lag Coefficient Of LogMink	Lag[1]	-0.9905	0.1419	-6.98
LagsForMusk	Lag Coefficient Of LogMusk	Lag[1]	0.6570	0.1208	5.44

The main points of the output can be summarized as follows:

- phi11, the first element of Φ , which relates the current value of LogMink with its lagged value, is statistically insignificant. That is, lagged LogMink term could be dropped from the model equation for LogMink.
- rho2, the correlation coefficient between the elements of ξ_t —the bivariate noise vector in the equation $Y_t = \mu_t + \Phi Y_{t-1} + \xi_t$ —is very near its lower boundary of -1 (in such cases the standard error of the parameter estimate is not reliable). This implies that the two elements of ξ_t are perfectly negatively correlated.

Taken together, these observations suggest the reduced model

$$\mu_t = \mu_{t-1} + v_t$$

$$Y_t = \mu_t + \Phi Y_{t-1} + \xi_t$$

where $\Phi = (0 \quad \phi_{12}; \phi_{21} \quad \phi_{22})$ and $\text{Cov}(\xi_t) = \Sigma_2$ is parameterized as (**esd1*esd1 -esd1*esd2; -esd1*esd2 esd2*esd2**). The program that produces the reduced model is a simple modification of the preceding program and is not shown.

Output 14.13.4 Estimates of Σ_1 , and Σ_2 (Reduced Model)

Estimates of Named Parameters			
Parameter	Estimate	Standard Error	t Value
rho1	0.8526	0.0978	8.71
msd1	0.2472	0.0282	8.78
msd2	0.1955	0.0365	5.36
esd1	0.0679	0.0385	1.76
esd2	0.1372	0.0328	4.18

Output 14.13.5 Estimates of Lag Coefficients (elements of Φ) (Reduced Model)

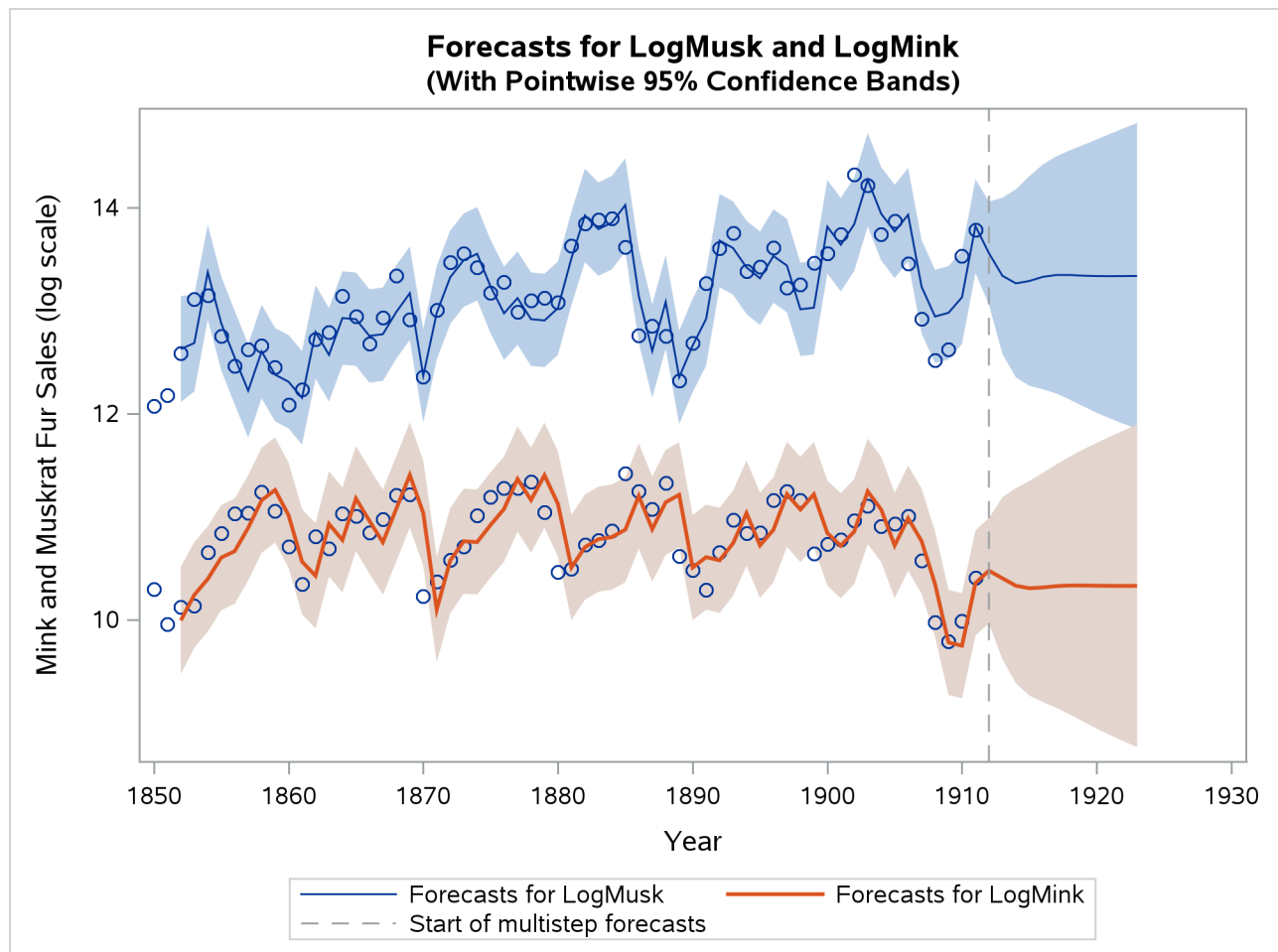
Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
LagsForMink	Lag Coefficient Of LogMusk	Lag[1]	0.3303	0.0986	3.35
LagsForMusk	Lag Coefficient Of LogMink	Lag[1]	-0.9973	0.1168	-8.54
LagsForMusk	Lag Coefficient Of LogMusk	Lag[1]	0.6680	0.1003	6.66

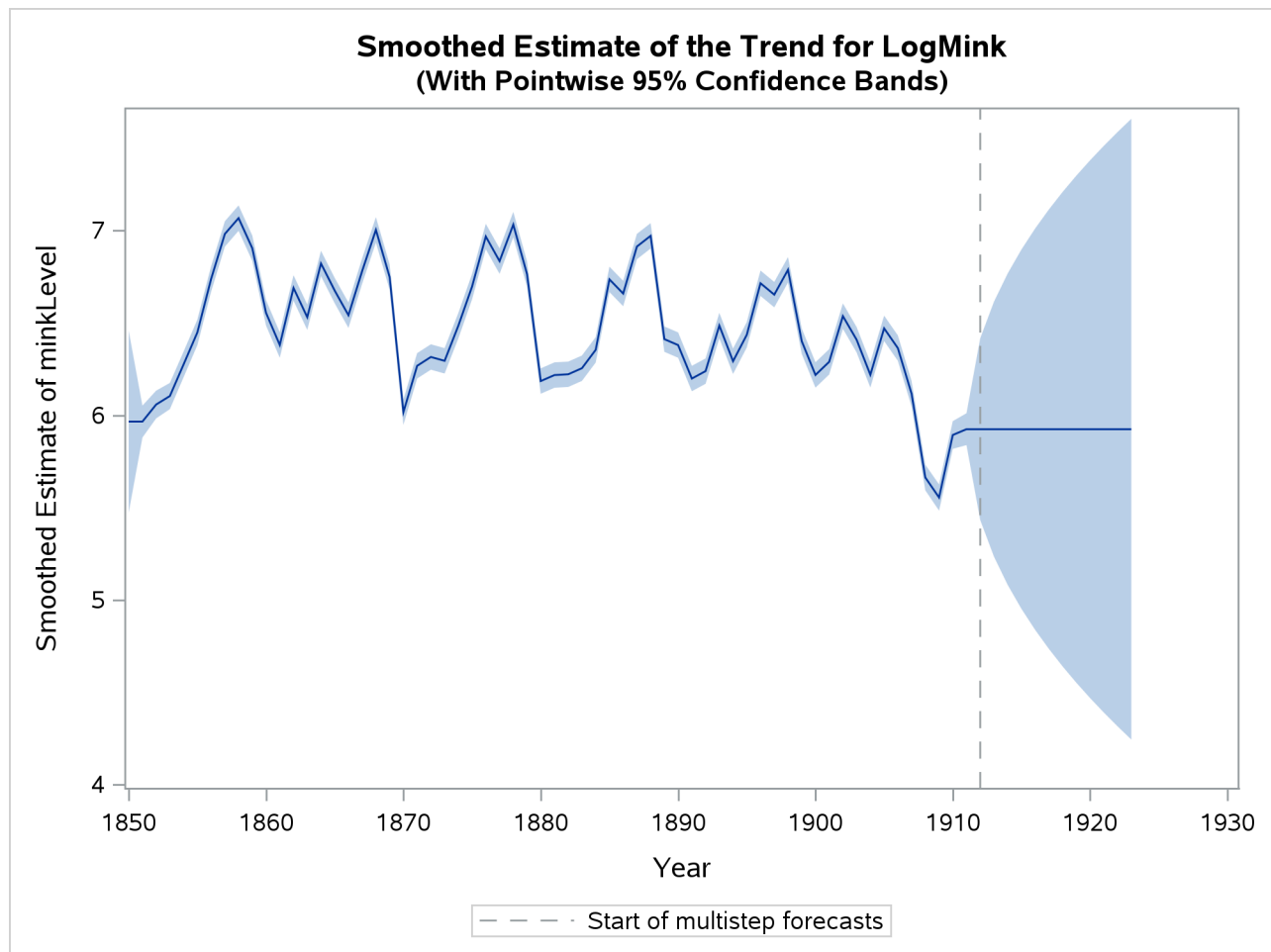
The tables in [Output 14.13.4](#) and [Output 14.13.5](#) show the new parameter estimates. By examining the parameter estimates, you can easily see that this model supports the general conclusions mentioned at the start of this example. In particular, note the following:

- $\hat{\phi}_{12} = 0.33$ implies that this year's mink abundance is positively correlated with last year's muskrat abundance.
- $\hat{\phi}_{21} = -0.99$ and $\hat{\phi}_{22} = 0.66$ imply that this year's muskrat abundance is negatively correlated with last year's mink abundance and positively correlated with last year's muskrat abundance.
- Even though the parameters were not restricted to ensure stability, the estimated Φ turns out to be stable with a pair of complex eigenvalues, $0.317 \pm i 0.473$, and a modulus of 0.570 (these calculations are done separately by using the IML procedure).
- The fact that elements of ξ_t are perfectly negatively correlated further supports the predator-prey relationship.

Finally, [Output 14.13.6](#) shows the plots of one-step-ahead and post-sample forecasts for LogMink and LogMusk, and [Output 14.13.7](#) shows the plot of the smoothed (full-sample) estimate of the first element of μ_t : LogMink Trend.

Output 14.13.6 Forecasts for Mink and Muskrat Fur Sales in Logarithms



Output 14.13.7 Smoothed Estimate of LogMink Trend

Example 14.14: A Factor Model for Now-Casting the US Economy

A well-known business conditions index, the Aruoba-Diebold-Scotti (ADS) business conditions index, is designed to track real business conditions at high frequency (for more information about this index, see <http://www.philadelphiafed.org/research-and-data/real-time-center/business-conditions-index/>). Its underlying (seasonally adjusted) economic indicators (weekly initial jobless claims, monthly payroll employment, industrial production, personal income less transfer payments, manufacturing and trade sales, and quarterly real GDP) blend high- and low-frequency information with stock and flow data. The ADS index is based on a rather elaborate state space model that takes into account the stock and flow nature of the underlying economic indicators. To simplify the illustration, this example uses the same economic indicators to develop a similar index by using a simpler factor model. You can also use PROC CSSM to carry out the more elaborate modeling that underlies the ADS index. All these economic indicators are freely available from the Federal Reserve Economic Data (FRED). You can access these data by using the SASEFRED interface engine; see Chapter 48, “SASEFRED Interface Engine” (*SAS/ETS User’s Guide*). The names of analysis variables and the relevant information that is needed for using the SASEFRED engine to obtain these data are shown in Table 14.11. All variables are transformed versions of the original series: all, except `l_icsa`, are both logged and differenced; `l_icsa` is only logged. The input data set for the

analysis, named *econ*, is formed by merging these six series of different frequencies. This merging is done by treating them as daily series that have missing values on the days when the series values are not available—for example, *ld_pinc* is nonmissing only for the first day of the month. The *econ* data set contains one more variable, *recession*, which is an indicator variable for the recessionary periods. This time series is an interpretation of the US Business Cycle Expansions and Contractions data that are provided by the National Bureau of Economic Research (NBER) at <http://www.nber.org/cycles/cyclesmain.html> (also see <http://research.stlouisfed.org/fred2/series/USRECDM>). The variable *recession* is not used in modeling. It is used later to demonstrate the adequacy of the activity index that is created in this example.

Table 14.11 Analysis Variables and Their FRED IDs

Name	FRED ID	Frequency	Description
<i>ld_payemp</i>	PAYEMS	Monthly	Payroll employment
<i>ld_pinc</i>	W875RX1	Monthly	Real personal income excluding current transfer receipts
<i>ld_mnfctr</i>	CMRMTSPL	Monthly	Real manufacturing and trade industries sales
<i>ld_indpro</i>	INDPRO	Monthly	Industrial production index
<i>ld_gdp</i>	GDPC1	Quarterly	Real GDP
<i>l_icsa</i>	ICSA	Weekly	Initial jobless claims

For easier model description, the variables *ld_payemp* to *ld_gdp* are also denoted as y_{1t} to y_{5t} , and the variable *l_icsa* is denoted as y_{6t} . Using this notation, the following model is postulated for the six daily time series:

$$y_{it} = \text{intercept}_i + \beta_i * \text{irw}_t + \epsilon_{it} \quad 1 \leq i \leq 5$$

$$y_{6t} = \mu_t + \beta_6 * \text{irw}_t + \epsilon_{6t}$$

A justification for this model is based on the following observations:

- The five time series y_{1t} to y_{5t} are logged and differenced versions of the underlying economic variables. Their plots (not shown here) show them to be hovering around a constant level, with some periods of deviation from this level. The plot of the sixth series, y_{6t} , which is logged but not differenced, shows a pronounced nonstationary pattern.
- All these series can be considered as proxies, possibly noisy, for the national economic activity. It is therefore reasonable to assume that a model for each of them will contain a common component, appropriately weighted, that is associated with the economic activity. In the current model this common component, named irw_t , is modeled as an integrated random walk. For y_{1t} to y_{5t} , the only other terms in the model are the respective intercepts, intercept_i , and the random disturbances, ϵ_{it} . Because y_{6t} shows a pronounced nonstationary pattern, its model has a time-varying level, μ_t , which is also modeled as an integrated random walk. For identifiability purposes, the initial condition for irw_t is taken to be 0. For the same reason, β_1 , the coefficient of irw_t in the model for y_{1t} is taken to be 1.
- The underlying economic variables of the five time series y_{1t} to y_{5t} are positively correlated with the economic activity—for example, payroll employment is expected to increase with increased economic activity. On the other hand, y_{6t} , which is associated with the initial jobless claims, is negatively correlated with the economic activity. This means that, with β_1 taken to be 1, the estimates of

$\beta_2, \dots, \beta_5$ are expected to be positive and the estimate of β_6 is expected to be negative. In the factor modeling terminology, irw_t is called a factor and $\beta_1, \dots, \beta_6$ are called the associated factor loadings.

The following statements load the econ data into the table mylib.econ:

```
/* Load data into mylib */
data mylib.econ;
  set econ;
run;
```

The following statements show you how to specify this model in the CSSM procedure:

```
ods output NamedParameterEstimates = named;
proc cssm data=mylib.econ; * opt(tech=activeset);
  id date interval=day;
  parms beta2-beta6;
  parms lv1-lv8 / upper=50;
  avar = exp(lv7);
  wnv1 = exp(lv1);  wnv2 = exp(lv2);
  wnv3 = exp(lv3);  wnv4 = exp(lv4);
  wnv5 = exp(lv5);  wnv6 = exp(lv6);
  tvar = exp(lv8);
  zero = 0;

  /* --- start of model spec ---*/
  state latent(2) t(g)=(1 1 0 1) cov(d)=(zero avar);
  comp c1 = latent[1];
  comp c2 = (beta2)*latent[1];
  comp c3 = (beta3)*latent[1];
  comp c4 = (beta4)*latent[1];
  comp c5 = (beta5)*latent[1];
  comp c6 = (beta6)*latent[1];

  irregular w1 variance=wnv1;
  int1 = 1;
  model ld_payemp = int1 c1 w1;

  irregular w2 variance=wnv2;
  int2 = 1;
  model ld_pinc = int2 c2 w2;

  irregular w3 variance=wnv3;
  int3 = 1;
  model ld_mnfctr = int3 c3 w3;

  irregular w4 variance=wnv4;
  int4 = 1;
  model ld_indpro = int4 c4 w4;

  irregular w5 variance=wnv5;
  int5 = 1;
  model ld_gdp = int5 c5 w5;

  irregular w6 variance=wnv6 ;
```



```

trend t_icsa(11) levelvar=0 slopevar=tvar;
model l_icsa = c6 t_icsa w6;
/* ---model spec done---*/

eval icsaPattern = c6 + t_icsa;
/*--index is a scaled version of the common factor--*/
eval Index = 1000*c1;
comp slope = latent[2];
eval IndexSlope = 1000*slope;
/*--just so recession is output to the output data set--*/
rec = recession;
output out=mylib.forecast1 press pdv;
run;

```

A few comments about the program:

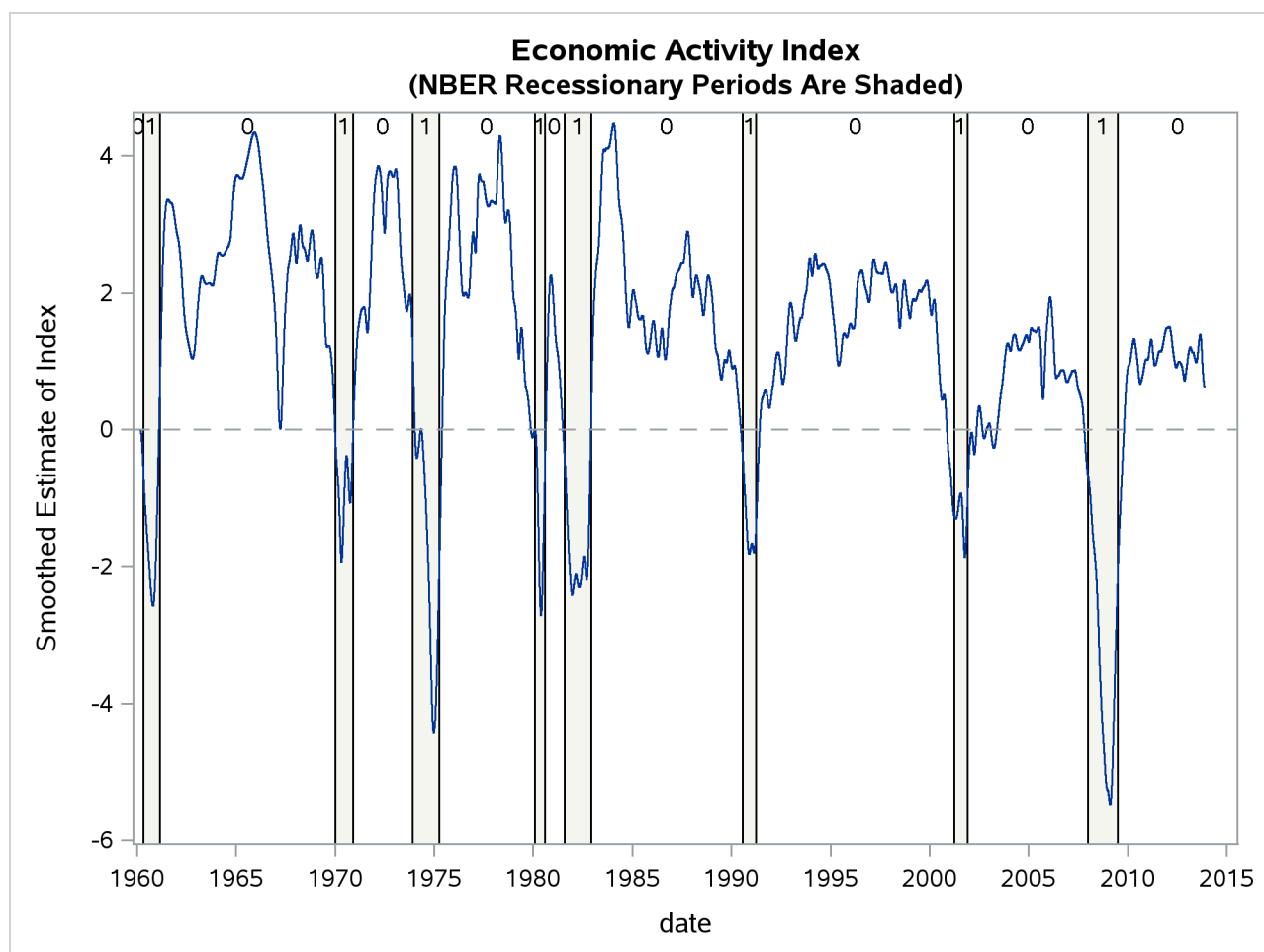
- If the model and data are in reasonable accord, the default likelihood optimization settings work in most situations. However, in some cases the likelihood optimization process needs additional customization. Some experimentation with alternative optimization techniques and different parameterization of the model parameters can help. This example turns out to be one such case. The optimization technique **ACTIVESET** (`opt (tech=activeset)`) works better for WINDOWS and a few other platforms, whereas the default optimization technique works better for the AIX platform. In addition, the variances of all the disturbance terms in the model are parameterized in the exponential scale.
- The two-dimensional state that is associated with irw_t is named `latent`, and irw_t (the first element of `latent`) itself is named `c1`. Note that the second element of `latent` corresponds to the slope of irw_t . The components `c2` to `c6` correspond to $\beta_i * \text{irw}_t$ for $2 \leq i \leq 6$.
- The desired business index, named `index`, is a scaled version of irw_t (`eval Index = 1000*c1;`). This scaling is done purely for ease of display—the scaled values turn out to be in the range of -6.0 to 5.0 . Another component, named `IndexSlope`, contains the slope of `index`, which is also a quantity of interest.

Output 14.14.1 Estimated Factor Loadings

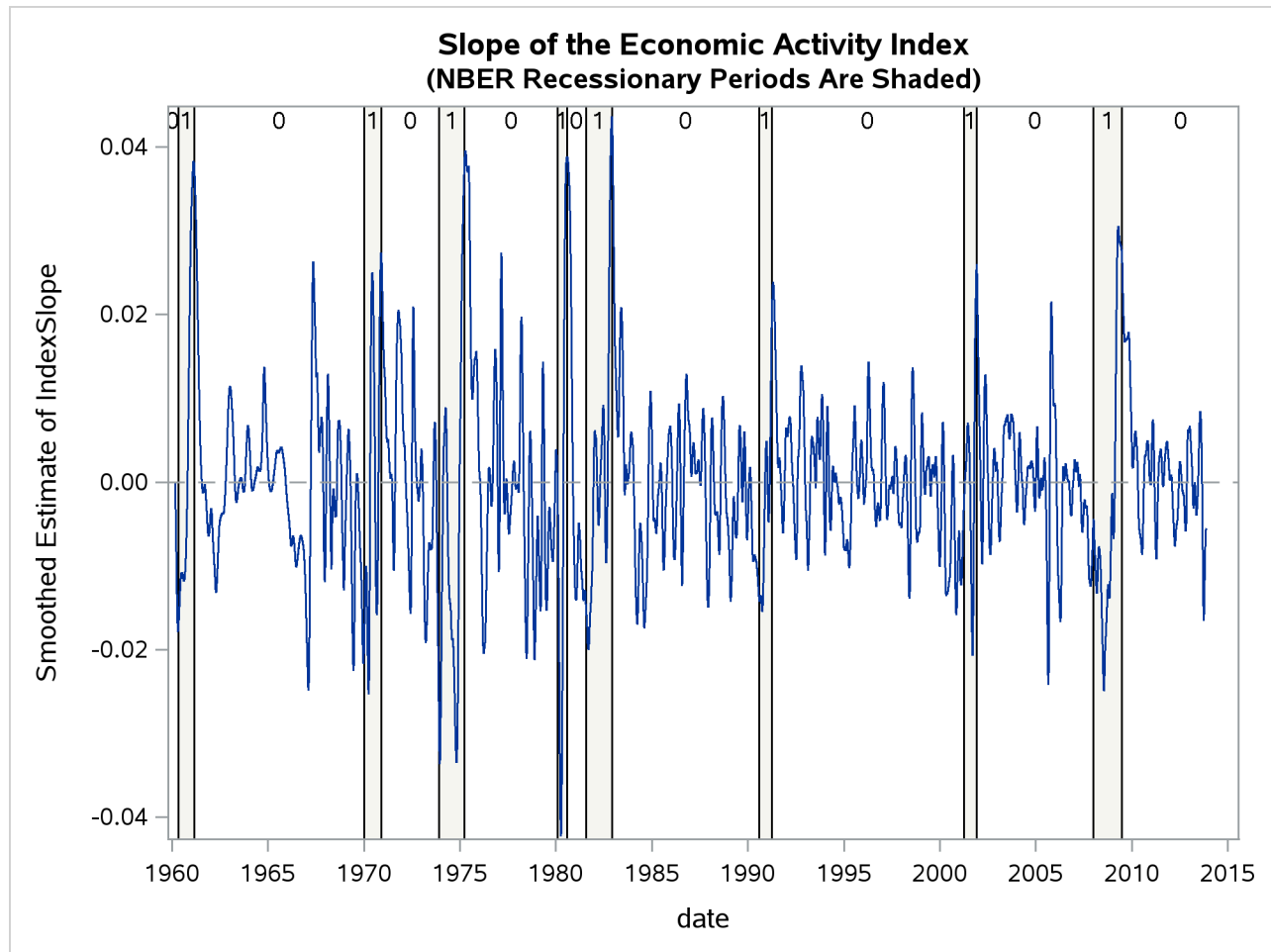
The Estimated Loadings

Parameter	Estimate	StdErr	tValue
beta2	1.1457	0.1276	8.98
beta3	1.9605	0.2390	8.20
beta4	2.4816	0.1646	15.08
beta5	3.2715	0.2653	12.33
beta6	-96.4457	9.6133	-10.03

Output 14.14.1 shows the estimated factor loadings. They are statistically significant and their signs are consistent.

Output 14.14.2 The Estimate of Economic Activity Index

Output 14.14.2 shows the plot of the smoothed index. Note that it coheres quite well with the NBER recessionary periods. In Aruoba, Diebold, and Scotti (2009, sec. 4.4) the features of an earlier version of the ADS index are discussed in detail. Similar comments apply to this indicator also.

Output 14.14.3 Estimated Slope of the Economic Activity Index

Finally, [Output 14.14.3](#) shows the plot of the slope of the index, which gives an idea of the direction of the economic activity.

Example 14.15: Longitudinal Data Analysis of Lung Function Data

The data for this example, which consist of 209 measurements of the lung function of an asthma patient, are analyzed in Wang (2013). The time series is measured mostly at two-hour time intervals but with irregular gaps. Wang (2013) fit a fourth-order continuous-time autoregressive model, CAR(4), to these data. The analysis results in a decomposition of the observed time series in three components:

- a slowly varying trend pattern, which appears to have a slight downward drift
- a diurnal component—a periodic pattern with a period of 24 hours
- a residual component

As shown in Wang (2013), the continuous-time autoregressive models can be formulated as state space models. However, in general, the form of such SSMs is quite complex. Consequently, specifying such a

model by using the current CSSM procedure syntax is impractical. On the other hand, you can analyze these types of longitudinal data by using continuous-time structural models, which are easy to specify in the SSM procedure. In this example, the lung function measurements, y , are modeled as

$$y_t = \text{intercept} + \beta * t + \zeta_t + \epsilon_t$$

where $(\text{intercept} + \beta * t)$ is a simple linear time trend, ζ_t is a continuous-time stochastic cycle, and ϵ_t is a Gaussian white noise sequence. Replacing the linear time trend with a more general time trend, such as a spline smoother, does not seem to change the fit, because the estimated smoothing spline turns out to be almost perfectly linear.

The following statements load the asth data into the table mylib.asth:

```
/* Load data into mylib */
data mylib.asth;
  set asth;
run;
```

The following statements show you how to specify this model in the CSSM procedure:

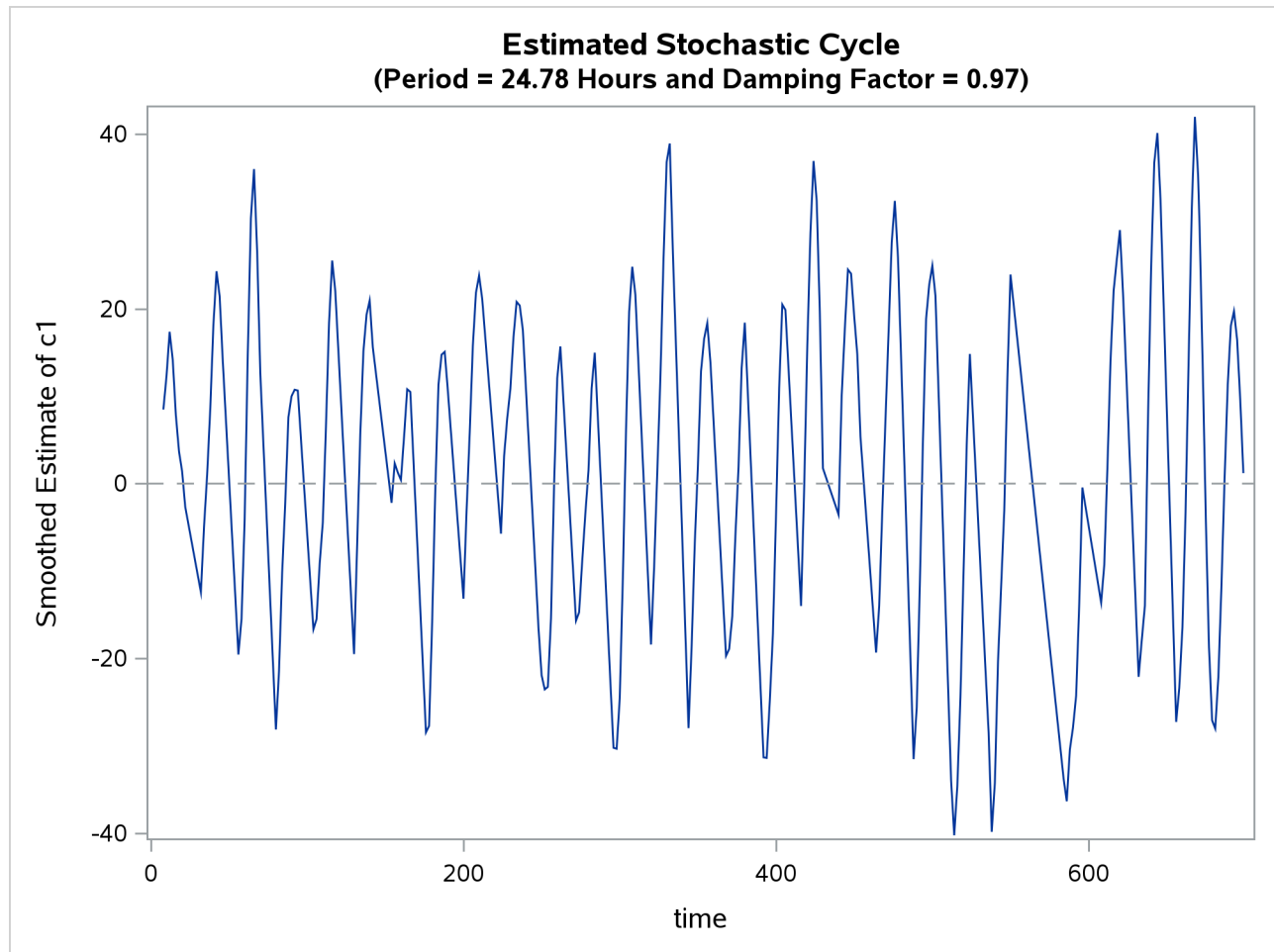
```
proc cssm data=mylib.asth;
  id time;
  state s1(1) type=cycle(CT) cov(g);
  comp c1 = s1[1];
  intercept = 1;
  irregular wn;
  model y= intercept time c1 wn;
  output out=mylib.for1 press;
  eval pattern=intercept+time+c1;
run;
```

The continuous-time stochastic cycle, named $c1$, is defined by a pair of STATE and COMPONENT statements. The STATE statement defines $s1$ as a state subsection that is associated with a univariate, continuous-time cycle (signified by the use of **type=cycle(CT)**). The COMPONENT statement defines $c1$ as its first element.

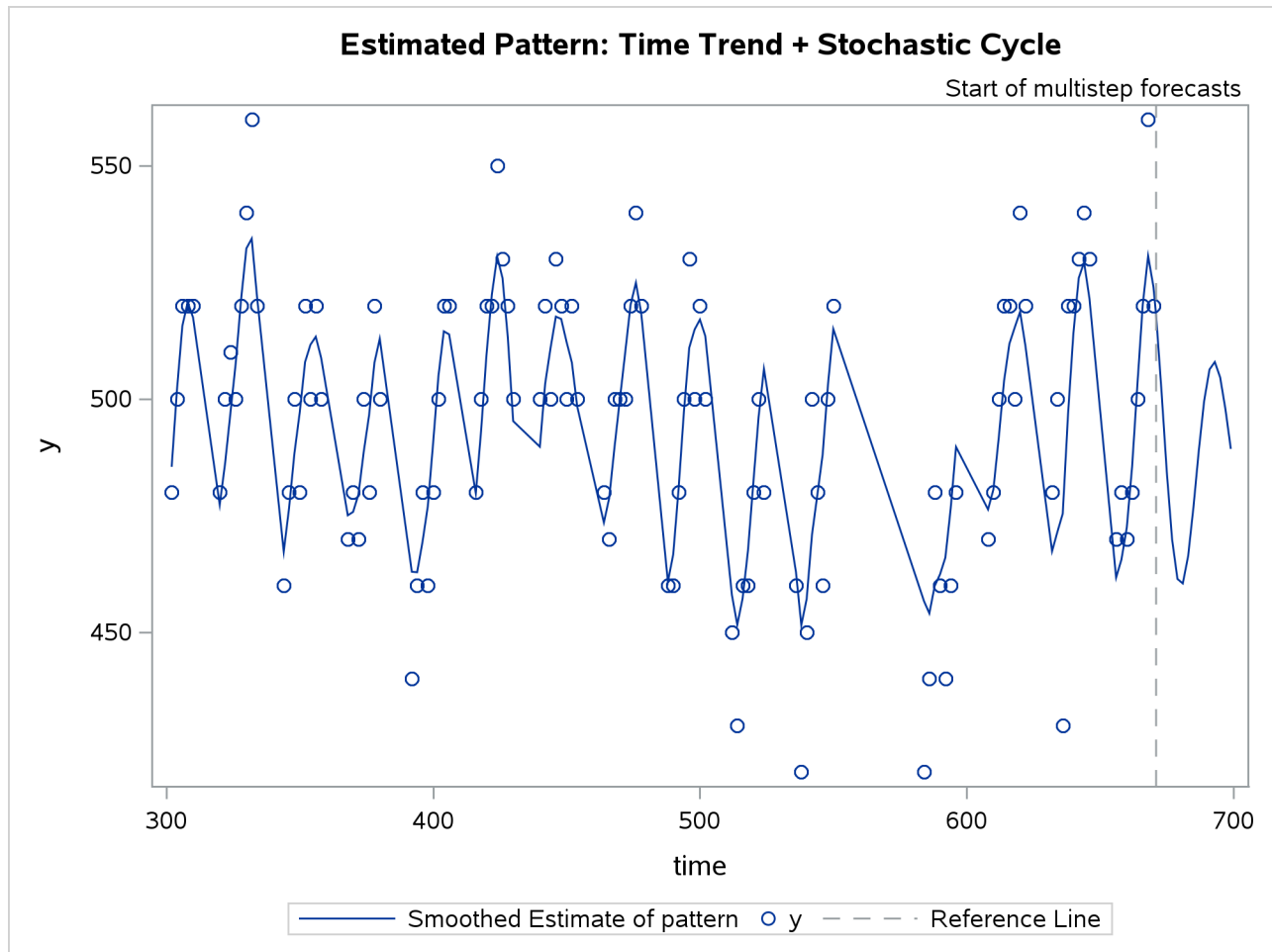
Output 14.15.1 Linear Time Trend: Estimates of Intercept and Slope

Regression Parameter Estimates					
Response	Regressor	Estimate	Standard		
			Error	t Value	Pr > t
y	intercept	502.1637	3.5047	143.28	<.0001
y	time	-0.0201	0.0092	-2.19	0.0286

Output 14.15.1 shows the estimated intercept and slope of the time trend. The estimated slope (only marginally significant) is negative, which is consistent with the overall downward drift.

Output 14.15.2 Estimated Stochastic Cycle: ζ_t 

Output 14.15.2 shows the plot of the estimated cycle component, which has a period of 24.78 hours and a damping factor of 0.97. That is, it is a nearly persistent diurnal cycle.

Output 14.15.3 Estimated Pattern: Intercept + $\beta * t + \zeta_t$ 

Output 14.15.3 shows the fit of the de-noised y values (intercept + $\beta * t + \zeta_t$). To reduce the clutter, only the second half of the data are plotted. The fit appears to be quite reasonable.

Example 14.16: Estimating Monthly GDP by Temporal Distribution

This example is based on a case study described in Pelagatti (2015, chap. 9, Example 9.2). The case study shows how you can estimate the monthly GDP (gross domestic product) for the United States by temporally distributing the quarterly GDP time series, which is readily available. The temporal distribution process is based on a bivariate model that relates two variables, the quarterly GDP and the monthly industrial production index (both for the United States). Assuming that t denotes the monthly time index, $indp_t$ denotes the monthly industrial production index series, and gdp_t denotes the quarterly GDP series that is organized as a monthly series (by setting the GDP numbers to missing for the months when they are not published), the case

study uses the following model,

$$\begin{aligned} \text{indp}_t &= \mu_{1,t} + \psi_{1,t} + \epsilon_{1,t} \\ \text{gdp}_t^\dagger &= \mu_{2,t} + \psi_{2,t} + \epsilon_{2,t} \\ \text{gdp}_t &= \text{gdp}_t^\dagger + \text{gdp}_{t-1}^\dagger + \text{gdp}_{t-2}^\dagger \end{aligned}$$

where

- $\text{gdp}_t^\dagger$ is the unobserved monthly GDP series (which is to be estimated)
- $\boldsymbol{\mu}_t = (\mu_{1,t} \ \mu_{2,t})$ is a bivariate trend component that follows an integrated random walk
- $\boldsymbol{\psi}_t = (\psi_{1,t} \ \psi_{2,t})$ is a bivariate cycle component
- $\boldsymbol{\epsilon}_t = (\epsilon_{1,t} \ \epsilon_{2,t})$ is a bivariate white noise component

As explained in the section “[Temporal Distribution](#)” on page 790, it is easy to fit this model by using the CSSM procedure. As a first step, a data set, `Usec`, is created that organizes the monthly industrial production index and the quarterly GDP as monthly series. Specifically, `Usec` contains four variables: `date` dates the observations, `indpro` contains the monthly industrial production index, `gdp` contains the quarterly GDP, and `startQ` is a dummy variable that indicates the start of the quarter. This data set is essentially the same as the one that is used in the case study, except that, to improve the computational stability, `gdp` is scaled by 100. This data set has one peculiarity that is not mentioned in the case study: the GDP reporting pattern appears to have changed a few times over the years (October 1969, May 1992, and December 2014). The dummy variable, `startQ`, which indicates the start of the aggregation interval, is appropriately modified to take these changes into account. [Output 14.16.1](#) shows the first few rows of `Usec`.

Output 14.16.1 First Few Rows of `Usec`

date	startQ	gdp	indpro
01JAN47	1	.	13.6351
01FEB47	0	.	13.7156
01MAR47	0	19.3447	13.7962
01APR47	1	.	13.6888
01MAY47	0	.	13.7425
01JUN47	0	19.3228	13.7425
01JUL47	1	.	13.6619
01AUG47	0	.	13.7425

The following statements load the `useco` data into the table `mylib.useco`:

```
/* Load data into mylib */
data mylib.useco;
  set useco;
run;
```

The following statements show you how to specify the bivariate model for `indpro` and `gdp`:

```

proc cssm data=mylib.useco opt(maxiter=100);
  id date interval=month;
  /* Bivariate integrated random walk */
  state irwState(2) type=ll(slopecov(g));
  comp irwInd = irwState[1];
  comp irwGdp = irwState[2];
  /* Bivariate cycle */
  state cycle(2) type=cycle cov(g);
  comp cycInd = cycle[1];
  comp cycGdp = cycle[2];
  /* Bivariate white noise */
  state noise(2) type=wn cov(g);
  comp noiseInd = noise[1];
  comp noiseGdp = noise[2];
  /* Observation equations */
  model indpro = irwInd cycInd noiseInd;
  model gdp = irwGdp cycGdp noiseGdp / distribute(start=startQ);
  /* Components for output */
  eval trendCycGdp = irwGdp + cycGdp;
  eval trendCycInd = irwInd + cycInd;
  eval monthlyGdp = irwGdp + cycGdp + noiseGdp;
  /* Output data set */
  output out=mylib.forGdp pdv press;
run;

```

Here are a few comments about this program:

- The first STATE statement specifies `irwState` as a bivariate trend that follows an integrated random walk (which is a local linear trend without the disturbance term in the level equation); `irwState` corresponds to μ_t . The trend components in the models for `indpro` and `gdp` are specified in the two COMP statements that follow: `irwInd` and `irwGdp` correspond to $\mu_{1,t}$ and $\mu_{2,t}$, respectively.
- Similarly, the second STATE statement and the two COMP statements that follow it define `cycInd` and `cycGdp` as the two cycle components ($\psi_{1,t}$ and $\psi_{2,t}$) in the model.
- The noise components, `noiseInd` and `noiseGdp`, which correspond to $\epsilon_{1,t}$ and $\epsilon_{2,t}$, respectively, are also defined in the same way.
- The MODEL statement for `indpro` corresponds to the equation $indp_t = \mu_{1,t} + \psi_{1,t} + \epsilon_{1,t}$. On the other hand, the DISTRIBUTE(START=*startQ*) option in the MODEL statement of `gdp` causes `gdp` to be modeled as an aggregated version of the unobserved monthly GDP series ($gdp_t^\dagger$):

$$\begin{aligned}
 gdp_t^\dagger &= \mu_{2,t} + \psi_{2,t} + \epsilon_{2,t} \\
 gdp_t &= gdp_t^\dagger + gdp_{t-1}^\dagger + gdp_{t-2}^\dagger
 \end{aligned}$$

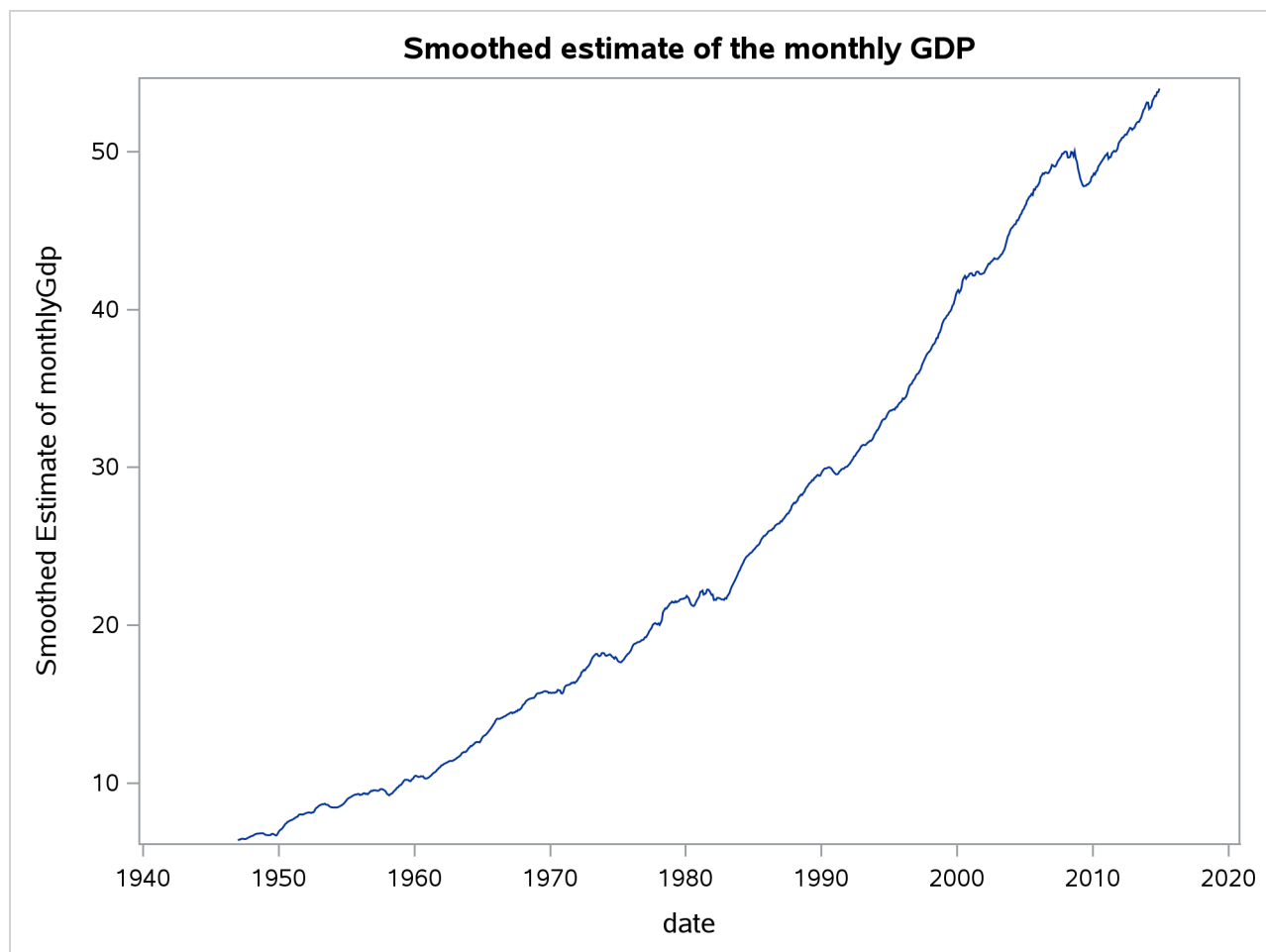
- After the model specification is complete, EVAL statements are used to define some useful linear combinations of the components that are part of the model specification—for example, `monthlyGdp` (which is defined as a sum of `irwGdp`, `cycGdp`, and `noiseGdp`) corresponds to the unobserved monthly GDP ($gdp_t^\dagger$). The CSSM procedure outputs the estimates of these components to the data set that is specified in the OUT= option in the OUTPUT statement.

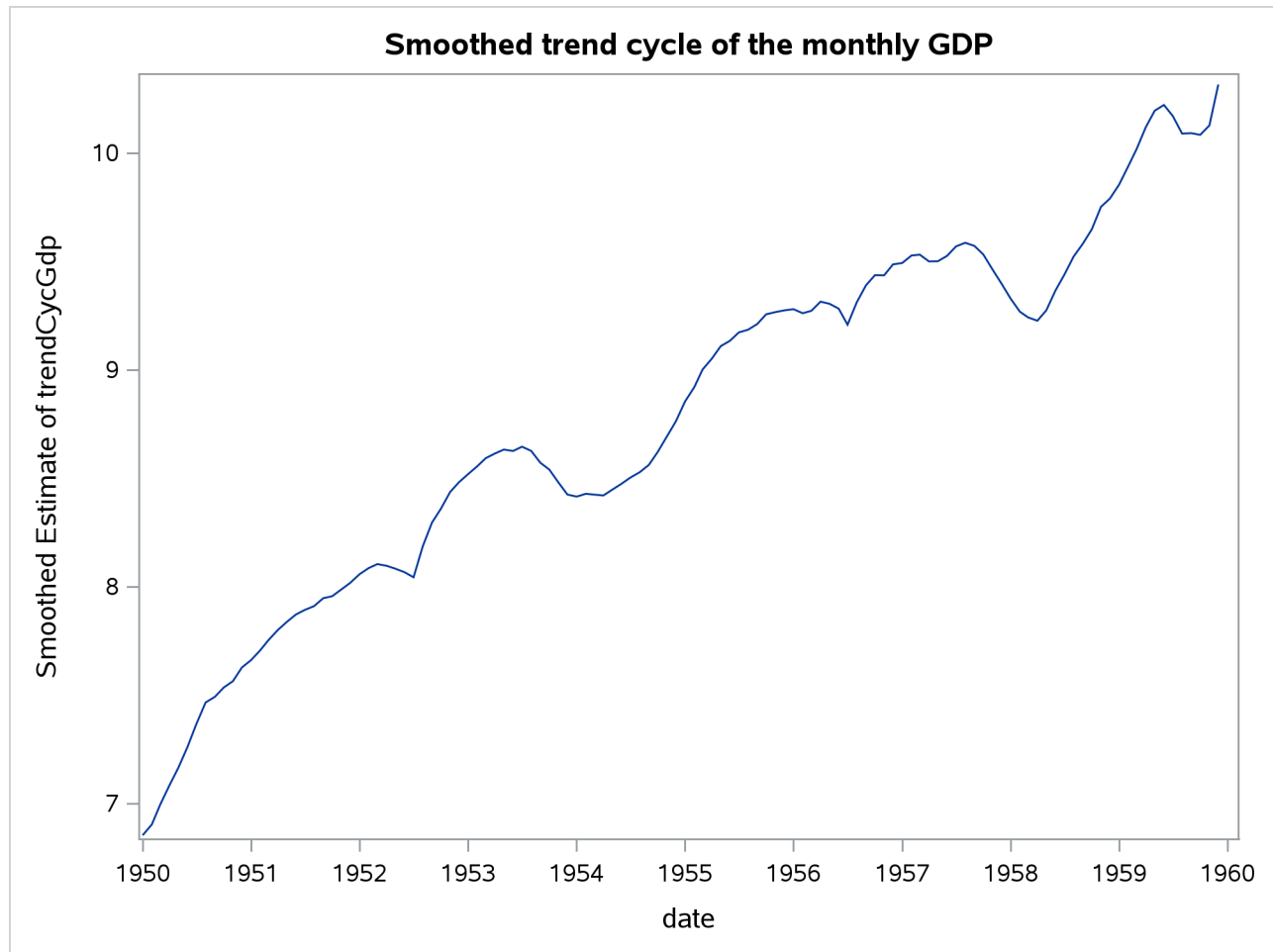
The parameter estimates in [Output 14.16.2](#) are similar to (but not the same as) the parameter estimates reported in the case study. In particular, the estimates of the parameters of the cycle component—for example, the damping factor ($\text{Rho} = 0.99228$) and the period (293.32178)—are reasonably close.

Output 14.16.2 Estimated Model Parameters

Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
irwState	Slope Disturbance Covariance	RootCov[1, 1]	0.1065	0.0136	7.85
irwState	Slope Disturbance Covariance	RootCov[2, 1]	0.0206	0.0029	7.05
irwState	Slope Disturbance Covariance	RootCov[2, 2]	0.0013	0.0005	2.52
cycle	Damping Factor	Rho	0.9923	0.0046	215.80
cycle	Cycle Period	Period	293.3218	94.2697	3.11
cycle	Disturbance Covariance	RootCov[1, 1]	0.3278	0.0288	11.39
cycle	Disturbance Covariance	RootCov[2, 1]	0.0420	0.0136	3.09
cycle	Disturbance Covariance	RootCov[2, 2]	0.0607	0.0077	7.92
noise	Disturbance Covariance	RootCov[1, 1]	0.0862	0.0389	2.22
noise	Disturbance Covariance	RootCov[2, 1]	-0.1212	0.0942	-1.29
noise	Disturbance Covariance	RootCov[2, 2]	0.0371	0.3225	0.11

[Output 14.16.3](#) shows the plot of the estimated monthly GDP, and [Output 14.16.4](#) shows the plot of the estimate of monthly trend-cycle estimate ($\mu_{2,t} + \psi_{2,t}$) for GDP.

Output 14.16.3 Estimate of Monthly GDP

Output 14.16.4 Smoothed Trend-Cycle Component of Monthly GDP (1950 to 1960)

Example 14.17: Temporal Aggregation-Based Triannual Nile River Level

This example illustrates how you can do model-based temporal aggregation of a response variable. The following DATA step creates a data set, Nile, by using a well-known data set that contains annual recordings of the Nile water level measured between the years 1871 and 1970. The Nile water level is clearly a stock variable, and temporal aggregation of such variables is usually meaningless. However, for illustration purposes, assume that you are interested in forecasting triannual totals of the water level.

```
data Nile;
  input level @@;
  year = intnx( 'year', '1jan1871'd, _n_-1 );
  format year year4.;
  startAggr = (mod(_n_, 3) = 1);
datalines;
1120 1160 963 1210 1160 1160 813 1230 1370 1140
995 935 1110 994 1020 960 1180 799 958 1140
1100 1210 1150 1250 1260 1220 1030 1100 774 840
874 694 940 833 701 916 692 1020 1050 969
```

```

831    726    456    824    702    1120    1100    832    764    821
768    845    864    862    698    845    744    796    1040    759
781    865    845    944    984    897    822    1010    771    676
649    846    812    742    801    1040    860    874    848    890
744    749    838    1050    918    986    797    923    975    815
1020    906    901    1170    912    746    919    718    714    740
. . . . .
;

```

The following statements load the Nile data into the table mylib.Nile:

```

/* Load data into mylib */
data mylib.Nile;
    set Nile;
run;

```

The Nile data set contains three variables: year indicates the observation year, level contains the yearly water level, and startAggr is a dummy variable that indicates the start of the triannual aggregation intervals. It is known that for the time span of the observations, the yearly water levels can be reasonably modeled as a sum of a random walk trend, a level shift in the year 1899, and the observation error. The following statements show you how to obtain forecasts of the triannual water level that are consistent with the model postulated for the yearly water levels:

```

proc cssm data=mylib.Nile;
    id year interval=year;
    shift1899 = ( year >= '1jan1899'd );
    trend rw(rw);
    irregular wn;
    model level = shift1899 RW wn / aggregate(start=startAggr);
    output out=mylib.nileOut;
quit;

```

As a result of running this program, you get the usual output that is associated with fitting the specified model to the yearly water level. In addition (as explained in the section “[Temporal Aggregation](#)” on page 792), the AGGREGATE option in the MODEL statement causes the estimation and printing of triannual aggregates of the water level. [Output 14.17.1](#) shows the last few rows of this output. When the summands—the response values—in the aggregation are known, the aggregation can be done without error; that is, the standard error of the estimation is zero. However, when at least one summand in the aggregate is missing, the standard error of estimation is nonzero.

Output 14.17.1 Triannual Aggregate Values of the Nile Water Levels (Partial Output)

Time	Response	Start_Flag	Aggregate	StdErr	Lower	Upper
1967	919	1	919	0.0000	919.0	919
1968	718	0	1637	0.0000	1637.0	1637
1969	714	0	2351	0.0000	2351.0	2351
1970	740	1	740	0.0000	740.0	740
1971	.	0	1590	128.4764	1338.2	1842
1972	.	0	2440	182.9334	2081.4	2798
1973	.	1	850	128.4764	598.2	1102
1974	.	0	1700	182.9334	1341.4	2058
1975	.	0	2550	225.5554	2107.8	2992
1976	.	1	850	128.4764	598.2	1102
1977	.	0	1700	182.9334	1341.4	2058

Example 14.18: Invariance of the Marginal Likelihood under Linear Rescaling of the Diffuse Effects

Consider the following alternate but equivalent specifications of a trend-plus-seasonal model (monthly seasonality):

$$y_t = \mu_t + \psi_t + \epsilon_t \quad \text{Spec1}$$

$$y_t = \mu_t + m_{1,t} + \cdots + m_{11,t} + \epsilon_t \quad \text{Spec2}$$

Here the trend (μ_t , a random walk with drift) and the irregular component (ϵ_t , white noise) are the same in both the specifications. However, the seasonal component is specified differently: in Spec1 the seasonality is modeled as a deterministic trigonometric seasonal component (ψ_t) whereas in Spec2 it is modeled using the seasonal dummies ($m_1 \cdots m_{11}$). Spec1 and Spec2 are statistically equivalent models from the perspective of the data generation process. This example uses these two specifications to demonstrate a useful invariance property of the marginal and profile likelihoods, which is described in the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769. The airline passenger series, given as Series G in Box and Jenkins (1976), is used to illustrate the computations. The following DATA step prepares the log-transformed passenger series and the seasonal dummies that are needed for this example:

```
data mylib.seriesG;
  set sashelp.air;
  logair = log(air);
  array m{11} m1-m11;
  do i=1 to 11;
    m[i] = (month(date)=i);
  end;
run;
```

The following statements fit the two models to the log-transformed passenger series. The first PROC CSSM call fits Spec1, and the second call fits Spec2.

```

proc cssm data=mylib.seriesG plots=none like=marginal;
  id date interval=month;
  trend rwDrift(11) slopevar=0;
  irregular wn;
  state trigState(1) type=season(length=12);
  comp season = trigState[1];
  model logair = rwDrift season wn;
run;

proc cssm data=mylib.seriesG plots=none like=marginal;
  id date interval=month;
  trend rwDrift(11) slopevar=0;
  irregular wn;
  model logair = rwDrift m1-m11 wn;
run;

```

For these two models, the parameter estimates that are based on the diffuse likelihood (REML_D) and the marginal likelihood ((REML_M) coincide because the extra term in the marginal likelihood ($-\log(|S_{n,p_n}^*|)$) turns out to be independent of these parameters. Nevertheless, it is useful to use the LIKE=MARGINAL option in the PROC CSSM statement so that both the likelihood computation summary and the information criteria tables display the likelihood values and the information criteria for all three likelihoods—diffuse, marginal, and profile—at the estimated parameters. The parameter estimates for Spec1 and Spec2 are displayed in [Output 14.18.1](#) and [Output 14.18.2](#), respectively. As expected, the parameter estimates for the two specifications are the same because they are statistically equivalent models. The other aspects of the fit (such as model-based forecasts), which are not shown, also agree.

Output 14.18.1 Parameter Estimates For Spec1

Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
rwDrift	LL Trend	Level Variance	0.0008	0.0002	3.49
wn	Irregular	Variance	0.0004	0.0001	2.60

Output 14.18.2 Parameter Estimates For Spec2

Model Parameter Estimates					
Component	Type	Parameter	Estimate	Standard Error	t Value
rwDrift	LL Trend	Level Variance	0.0008	0.0002	3.49
wn	Irregular	Variance	0.0004	0.0001	2.60

The fit summary tables shown in [Output 14.18.3](#) (for Spec1) and [Output 14.18.4](#) (for Spec2) show that the marginal and profile likelihoods (the last two lines in each table) for the two specifications also agree. However, you can see that the diffuse likelihood value for the two specifications differ (diffuse likelihood = 215.45 for Spec1 and diffuse likelihood = 226.89 for Spec2). This difference occurs because the diffuse likelihood is not invariant to the different (but equivalent) formulations of the seasonal effects. This also means that the information criteria that are based on the marginal and profile likelihoods, which are shown in [Output 14.18.5](#) (for Spec1) and [Output 14.18.6](#) (for Spec2), correctly conclude that the two specifications cannot be distinguished on the basis of these criteria, whereas the information criteria that are based on the

diffuse likelihood erroneously suggest that Spec1 is inferior to Spec2.

Output 14.18.3 Likelihood Computation Summary For Spec1

Likelihood Computation Summary	
Nonmissing Response Values Used	144
Estimated Parameters	2
Initialized Diffuse State Elements	13
Normalized Residual Sum of Squares	131.0000
Diffuse Log Likelihood	215.4522
Profile Log Likelihood	265.6388
Marginal Log Likelihood	248.0141

Output 14.18.4 Likelihood Computation Summary For Spec2

Likelihood Computation Summary	
Nonmissing Response Values Used	144
Estimated Parameters	2
Initialized Diffuse State Elements	13
Normalized Residual Sum of Squares	131.0000
Diffuse Log Likelihood	226.8959
Profile Log Likelihood	265.6388
Marginal Log Likelihood	248.0141

Output 14.18.5 Information Criteria For Spec1

Information Criteria			
Statistic	Diffuse Likelihood	Profile Likelihood	Marginal Likelihood
AIC (lower is better)	-426.9	-501.3	-492.0
BIC (lower is better)	-421.2	-456.7	-486.3
AICC (lower is better)	-426.8	-497.5	-491.9
HQIC (lower is better)	-424.6	-483.2	-489.7
CAIC (lower is better)	-419.2	-441.7	-484.3

Output 14.18.6 Information Criteria For Spec2

Information Criteria			
Statistic	Diffuse Likelihood	Profile Likelihood	Marginal Likelihood
AIC (lower is better)	-449.8	-501.3	-492.0
BIC (lower is better)	-444.0	-456.7	-486.3
AICC (lower is better)	-449.7	-497.5	-491.9
HQIC (lower is better)	-447.5	-483.2	-489.7
CAIC (lower is better)	-442.0	-441.7	-484.3

This example highlights the care that must be taken while doing model selection based on information criteria. It suggests that information criteria that are based on the marginal and profile likelihoods are preferred over

the information criteria that are based on diffuse likelihood.

Example 14.19: Scenario Analysis Using Scoring

In many data analyses, after you identify and fit a suitable state space model, you need to use the fitted model to predict the response series under different predictor scenarios in the future. You can easily accomplish this by using the scoring functionality that the **SCORE** statement provides. This example uses the data and the model discussed in the section “[Getting Started: CSSM Procedure](#)” on page 722 to demonstrate this type of scenario analysis. Recall that the data set, *Cigar*, contains information about yearly per capita cigarette sales for 46 geographic regions in the United States over the period 1963–1992. The variables *lsales*, *lprice*, *lndi*, and *lpimin* denote the per capita cigarette sales, price per pack of cigarettes, per capita disposable income, and minimum price in adjoining regions per pack of cigarettes, respectively (all in the natural log scale). The variable *year* contains the observation year, and the variable *region* contains an integer from 1 to 46 that serves as the unique identifier of the region. The goal of the analysis is to study the impact of the regressors on smoking behavior and to understand the changes in smoking patterns in different regions over those years. The following model for *lsales* is fitted:

$$\text{lsales}_{i,t} = \mu_{i,t} + \text{lprice} \beta_1 + \text{lndi} \beta_2 + \text{lpimin} \beta_3 + \epsilon_{i,t}$$

This example uses this model to predict *lsales* in the years 1993 and 1994 for three different settings of the predictors *lprice*, *lpimin*, and *lndi*. The first step in this type of scenario analysis is to create a score store in which the necessary context information is saved. The following statements specify and fit this model and also create the required score store:

```
proc cssm data=mylib.Cigar;
  id year interval=year;
  obsindex region;
  array RegionArray{46} region1-region46;
  do i=1 to 46;
    RegionArray[i] = (region=i);
  end;
  trend IrwTrend(11) cross(matchparm)=(RegionArray) levelvar=0;
  irregular wn;
  model lsales = lprice lndi lpimin IrwTrend wn;
  eval TrendPlusReg = IrwTrend + lprice + lndi + lpimin;
  score out=mylib.store;
run;
```

The score store is specified in the **OUT=** option of the **SCORE** statement (**score out=mylib.store;**). This score store, *mylib.store*, saves all the context information that is needed for continuing the forecasting process for the years after 1992, based on the fitted model and the history up to 1992.

The score store *mylib.store* is used to predict *lsales* in 1993 and 1994 for the following three scenarios:

Same price: In this scenario, the predictors retain their values from years 1991 and 1992; that is, the predictor values for 1991 are copied for 1993, and the predictor values for 1992 are copied for 1994. The data table to be scored is named *mylib.same*.

Low price: In this scenario, the predictor settings are as follows:

- For the year 1993, the values of disposable income (Indi) are the same as for 1991, and the values of lprice and lpimin are set to 3.15 for all regions.
- For the year 1994, the values of disposable income (Indi) are the same as for 1992, and the values of lprice and lpimin are set to 3.0 for all regions.

The data table to be scored is named mylib.low.

High price: In this scenario, the predictor settings are as follows:

- For the year 1993, the values of disposable income (Indi) are the same as for 1991, and the values of lprice and lpimin are set to 5.5 for all regions.
- For the year 1994, the values of disposable income (Indi) are the same as for 1992, and the values of lprice and lpimin are set to 6.0 for all regions.

The data table to be scored is named mylib.high.

The following statements create the data tables mylib.same, mylib.low, and mylib.high:

```
data mylib.same;
  set Cigar(where=(year >= '1jan1991'd));
  lsales = .;
  if year = '1jan1991'd then year = '1jan1993'd;
  else year = '1jan1994'd;
run;
data mylib.low;
  set Cigar(where=(year >= '1jan1991'd));
  lsales = .;
  if year = '1jan1991'd then do;
    year = '1jan1993'd;
    lprice = 3.15;
    lpimin = 3.15;
  end;
  else do;
    year = '1jan1994'd;
    lprice = 3.0;
    lpimin = 3.0;
  end;
run;
data mylib.high;
  set Cigar(where=(year >= '1jan1991'd));
  lsales = .;
  if year = '1jan1991'd then do;
    year = '1jan1993'd;
    lprice = 5.5;
    lpimin = 5.5;
  end;
  else do;
    year = '1jan1994'd;
    lprice = 6.0;
    lpimin = 6.0;
  end;
run;
```

The following statements show how to score these data tables by using the score store mylib.store:

```
proc cssm data=mylib.same;
  score in=mylib.store;
  output out=mylib.scoreSame;
run;

proc cssm data=mylib.low;
  score in=mylib.store;
  output out=mylib.scoreLow;
run;

proc cssm data=mylib.high;
  score in=mylib.store;
  output out=mylib.scoreHigh;
run;
```

In each case, the following are true:

- The data table to be scored is specified in the DATA= option in the PROC statement.
- The score store is specified by using the IN= option in the SCORE statement.
- An output table to save the scored rows is specified in the OUT= option in the OUTPUT statement.

Output 14.19.1 shows the forecasts for `lsales` for the first three regions (Alabama, Arkansas, and Arizona). This table is produced by merging the three output tables (`mylib.scoreSame`, `mylib.scoreLow`, and `mylib.scoreHigh`), by renaming the forecast variables, and by subsetting the rows suitably. As expected, the sales forecasts increase as the prices decrease.

Output 14.19.1 Forecasts under Different Price Settings

Cigarette Sales Forecasts under Different Price Settings

region	year	sales_highPrice	sales_samePrice	sales_lowPrice
Alabama	1993	4.59	4.73	5.26
Arizona	1993	4.19	4.33	4.87
Arkansas	1993	4.74	4.88	5.41
Alabama	1994	4.47	4.71	5.33
Arizona	1994	4.02	4.27	4.87
Arkansas	1994	4.65	4.87	5.50

Example 14.20: Monitoring Streaming Data Using Scoring

This example shows how you can use the scoring functionality to monitor a continuous stream of observations. The goal is to detect structural changes in the observation process as early as possible. For illustration purposes, the example uses the quarterly bivariate series (f_KSI, r_KSI), which is discussed in “[Example 14.1: Bivariate Basic Structural Model](#)” on page 803. The variable f_KSI represents the quarterly average of the log of the monthly totals of front-seat passengers who are killed or seriously injured in car accidents, and r_KSI represents a similar number for rear-seat passengers. Recall that the `seatbelt` data set contains 16 years of series values, starting from the first quarter of 1969. This example uses the first 13 years—52 quarters—of data (through the last quarter of 1981) to fit a suitable model for the series and to create a score store that saves the context for later use. It is assumed that the fitted model continues to hold in the subsequent years. Monitoring of the process starts with the first quarter of 1982. The remaining 12 rows of the `seatbelt` data set are processed one by one and mimic a quarterly streaming data sequence. In real-world applications, the data source can be more general, and the observations could arrive at a much higher frequency, such as every minute.

Assume that the model for $y = (f_KSI, r_KSI)$ is

$$y_t = \mu_t + \zeta_t + \xi_t$$

where

- μ_t denotes a bivariate random walk. To capture the fact that f_KSI and r_KSI move together (that is, they are co-integrated), the covariance of the disturbance term of this random walk is assumed to be of less than full rank.
- ζ_t denotes a bivariate, time-invariant, trigonometric seasonal term.
- ξ_t denotes a bivariate white noise term.

This model is the same as the one used in “[Example 14.1: Bivariate Basic Structural Model](#)” on page 803, except that in this case no correction is made for the seat-belt law of 1983. In fact, the goal of this exercise is to determine, without the knowledge of this law, whether the level shift in the front-seat passenger series that is caused by this law can be detected during this monitoring.

The following DATA step loads the first 13 years of data into the table `mylib.train`:

```
data mylib.train;
    set seatBelt(obs=52);
run;
```

The following statements specify and fit the model, and save the scoring context in the store `mylib.inscore`:

```
proc cssm data=mylib.train opt(tech=activeset);
    id date interval=quarter;
    state error(2) type=WN cov(g);
    component wn1 = error[1];
    component wn2 = error[2];
    state level(2) type=RW cov(rank=1) checkbreak;
    component rw1 = level[1];
```

```

component rw2 = level[2];
state season(2) type=season(length=4);
component s1 = season[1];
component s2 = season[2];
model f_KSI = rw1 s1 wn1;
model r_KSI = rw2 s2 wn2;
output break(alpha=0.01) ao(alpha=0.01);
score out(10)=mylib.inscore;
run;

```

Note that because the `OUT(k)=` form of the `OUT` option is used in the `SCORE` statement (`score out(10)=mylib.inscore;`), the `mylib.inscore` store retains the context information from the last 10 quarters (from 1979/3 to 1981/4). The score store also retains the structural break detection settings that are specified in the `STATE` statement for `level`, and it retains the significance levels for the additive outlier detection and structural break detection that are specified in the `OUTPUT` statement.

After the model fitting and the initial score-store creation, the monitoring of streaming data is done using the following steps:

- Starting with 1982/1 (which happens to be the 53rd row of `seatbelt`), successive quarters are loaded into the `mylib.nextObs` table one at a time; that is, `mylib.nextObs` always has only one row.
- Each quarter is scored by using the `mylib.inscore` store and, additionally, an output score store (`mylib.outscore`) is created that incorporates the information in the current observation (`score in=mylib.inscore out=mylib.outscore;`).
- The two score stores, `mylib.inscore` and `mylib.outscore`, are swapped so that `mylib.outscore` becomes the input score store for the next observation.

The following macro, `%STREAM`, performs these three steps:

```

%macro STREAM;
  %do i=53 %to 64;
    /* read the next row */
    data mylib.nextObs;
      set seatBelt(firstobs=&i obs=&i);
    run;
    /* score the row that is read */
    proc cssm data=mylib.nextObs;
      ods output AOSummary=ao&i;
      ods output StateElementBreakSummary=ls&i;
      %if &i = 53 %then %do;
        ods output IdInformation=idInfo;
        ods output ResponseInfo=respInfo;
      %end;
      score in=mylib.inscore out=mylib.outscore;
    run;
    /* swap inscore and outscore */
    proc cas;
      table.altertable/name="outscore", rename="tmp";
      table.altertable/name="inscore", rename="outscore";
      table.altertable/name="tmp", rename="inscore";
    run;
  %end;
%macroend;

```

```

quit;
%end;
%mend;

%STREAM;

```

For each new quarter, the %STREAM macro saves the results, if any, of the additive outlier detection and the structural break detection in SAS data sets (`ods output AOSummary=ao&i;` and `ods output StateElementBreakSummary=ls&i;`). For the first quarter, the ID information table and the response information table are also saved. The ID information table for the first quarter (1982/1) is shown in [Output 14.20.1](#). Note that even though the input table (`mylib.nextObs`) has just one observation (which corresponds to 1982/1), PROC CSSM processed 11 observations (from 1979/3 to 1982/1). This is because, internally, PROC CSSM works with a data set that is created by combining the information from the previous 10 quarters (1979/3 to 1981/4) that is present in the input score store and the one quarter (1982/1) that is present in the current input data table (`mylib.nextObs`).

Output 14.20.1 ID Variable Information at Start of Scoring

ID Variable Information

Name	Start	End	NumberDistinct	MaximumDelta	Type
date	1979/3	1982/1	11	1.0000	Regular

For the same reason, [Output 14.20.2](#) also shows the response variable information for 11 quarters.

Output 14.20.2 Response Variable Information at Start of Scoring

Response Variable Information

Name	NObs	NMiss	InducedNMiss	Minimum	Maximum	Mean	StdDev
f_KSI	11	0	0	6.464	6.799	6.663	0.1147
r_KSI	11	0	0	5.635	6.119	5.914	0.1610

The outlier detection and the structural break detection, which are performed after the arrival of each new quarterly observation, are also performed on the set of 11 observations that is formed by the current observation and the observations in the 10 previous quarters. It happens that no additive outliers or structural breaks are detected in the four quarters of 1982. However, starting with the first quarter of 1983 (when the seat-belt law is introduced), additive outliers and structural breaks are detected in each quarter. The aggregate summary of this detection activity (obtained by combining the following pairs of data sets: `ao57` with `ao64` and `ls57` with `ls64`) is shown in two tables: [Output 14.20.3](#) shows the results of break detection for the bivariate state, level, and [Output 14.20.4](#) shows the results of the additive outlier detection. In both tables, an extra column, `obs_date`, is added to indicate the quarter that is associated with the observation that was scored. In addition, two more columns, `zValue` ($= \text{Estimate}/\text{StdErr}$) and `ProbZ` (the corresponding p -value), are added to the additive outlier table for ease of comparison with the break detection table. An examination of the two tables shows the following:

- When the observation that is associated with the first quarter of 1983 is processed, an additive outlier is flagged for `f_ksi` and a structural break is flagged at the first element of the bivariate state, level, both at time 1983/1. Recall that the first element of level corresponds to `f_ksi`. The associated Z-score, -7.66 ,

is the same in both cases. This makes sense, because at this point it is difficult to determine whether the unusually low `f_ksi` value is attributable to a shift in the level or is simply a one-time change.

- When the observation that is associated with the second quarter of 1983 is processed, 1983/1 is again flagged as an additive outlier for `f_ksi` and as a structural break for the first element of `level`. The only difference is that the evidence of a structural break has become stronger than before (more extreme Z-score: -9.23), whereas the evidence for an additive outlier has weakened (less extreme Z-score: -5.67).
- When the observation that is associated with the third quarter of 1983 is processed, 1983/1 continues to be flagged as a structural break for the first element of `level`, but it is no longer the most extreme additive outlier for `f_ksi`. Instead, 1983/1 is flagged as an additive outlier for `r_ksi`, because it has a larger value than expected! This is understandable because, as the model shows, `f_KSI` and `r_KSI` move together, so the 1983/1 value of `r_ksi` should have been lower than the observed value.
- As more observations are processed, the evidence that 1983/1 is the location of the level shift in `f_KSI` keeps getting stronger.

In real-world applications, when sufficient evidence for a break is gathered, the monitoring is paused and some appropriate action is taken.

Output 14.20.3 Level Shift Detection during Monitoring of 1982/1 to 1984/4

Detection of Break in the Bivariate State: Level

obs_date	Time	StateBreak	ElementIndex	zValue	ProbZ
1983/1	1983/1	level	1	-7.66	<.0001
1983/2	1983/1	level	1	-9.23	<.0001
1983/3	1983/1	level	1	-10.50	<.0001
1983/4	1983/1	level	1	-11.60	<.0001
1984/1	1983/1	level	1	-13.17	<.0001
1984/2	1983/1	level	1	-14.43	<.0001
1984/3	1983/1	level	1	-15.10	<.0001
1984/4	1983/1	level	1	-16.12	<.0001

Output 14.20.4 Additive Outlier Detection during Monitoring of 1982/1 to 1984/4

Detection of Additive Outliers

obs_date	Time	Name	Estimate	StdErr	zValue	ProbZ
1983/1	1983/1	f_KSI	-0.3505	0.0458	-7.66	<.0001
1983/2	1983/1	f_KSI	-0.2345	0.0413	-5.67	<.0001
1983/3	1983/1	r_KSI	0.2669	0.0488	5.465	<.0001
1983/4	1983/1	r_KSI	0.2658	0.0488	5.442	<.0001
1984/1	1984/1	r_KSI	0.3216	0.0490	6.566	<.0001
1984/2	1984/1	r_KSI	0.3226	0.0488	6.615	<.0001
1984/3	1984/1	r_KSI	0.3232	0.0487	6.632	<.0001
1984/4	1984/1	r_KSI	0.3239	0.0487	6.648	<.0001

Example 14.21: Simulation-Smoothing-Based Sampling Distribution of the Sum of Forecasts

This example shows how you can use the CSSM procedure to derive the sampling distribution of a complicated function of the forecasts of a response variable by using simulation smoothing. (For more information, see the section “[Simulation Smoothing](#)” on page 787.) The example uses the well-known airline passenger series, given as Series G in (Box and Jenkins 1976), and the equally well-known model of it, the Airline model. This series is a monthly series that consists of the number of airline passengers who traveled during the years 1949 to 1960. The data set `sashelp.air` contains this series (`air`) along with the date variable (`date`). The Airline model postulates an ARIMA (autoregressive, integrated, moving average) model for the logarithm of the air series (`logAir`). That is, the modeling is done in the log scale. However, often the results of the modeling are needed in the original scale. For example, you might want to obtain a model-based confidence interval for the total number (sum of 12 months) of airline passengers in some future year. The following steps summarize how to do this by using simulation smoothing:

1. Use PROC CSSM to fit the Airline model to the `logAir` series by using the first 11-year span of the data (1949 through 1959), and obtain the model-based forecasts of `logAir`, denoted by `forecast_logAir`, for the 12 months of 1960. The actual data for the months in 1960 are withheld from the fitting and forecasting phases and reserved for the “sanity check” at the end of the analysis.
2. Save the fitted model for its subsequent use in simulation smoothing by using the `OUTMODEL=` option in the `SIMSMOOTH` statement.
3. Let $\text{TotalAir1960} = \sum_{\text{Jan1960}}^{\text{Dec1960}} \exp(\text{forecast_logAir})$. `TotalAir1960` is a point forecast of the total number of airline passengers in 1960, based on the data from 1949 to 1959. Because it is a sum of the exponential of `forecast_logAir` for different months of 1960 that are also correlated, its sampling distribution is analytically intractable. Simulation smoothing is applicable in precisely such situations.
4. Using PROC CSSM again with the `SIMSMOOTH` statement, generate 1,000 random draws of the forecasts of `logAir` (`forecast_logAir`) for the 12 months of 1960. Using the `OUTPRED=` option, save these draws to an output table for postprocessing.
5. Using the generated 1,000 draws of `forecast_logAir`, obtain 1,000 draws of `TotalAir1960`. These are used to obtain the simulation-based sampling distribution of `TotalAir1960`.

The remainder of this example shows the SAS code for carrying out these steps and the results that they produce. The following DATA step creates a data set, `mylib.air`, from `sashelp.air` such that it has the log-transformed passenger series (`logAir`); `logAir` is set to missing during 1960 so that PROC CSSM produces forecasts and later the simulations for that period:

```
data mylib.air;
    set sashelp.air;
    if year(date) < 1960 then logAir = log(air);
    else logAir = .;
run;
```

The following statements fit the Airline model (ARIMA(0,1,1)(0,1,1)12) to the `logAir` series and save the model in the BLOB table `mylib.airModel`:

```
proc cssm data=mylib.air;
  id date interval=month;
  deplag airDiffs(logAir) logAir(lags=(1 12 13) coeff=(1 1 -1));
  trend ma(arma(q=1 sq=1 s=12));
  model logAir = airDiffs ma;
  simsmooth outmodel=mylib.airModel;
run;
```

The following statements use the fitted model that is saved in the mylib.airModel table and the input data table mylib.air to generate the desired simulations:

```
proc cssm data=mylib.air;
  simsmooth inmodel=mylib.airModel seed=1 nsim=1000
    outpred=mylib.airPred;
run;
```

Output 14.21.1 shows the summary of the simulation smoothing process.

Output 14.21.1 Simulation Smoothing of logAir Predictions

Simulation Smoothing Summary	
Input Data Table	AIR
Model Store	AIRMODEL
Number of Simulations	1000
Random Number Seed Sequence	1 : 1000
Simulated Response Predictions Table	AIRPRED

Because of the option NSIM=1000, PROC CSSM produces 1,000 draws of predictions of logAir for the 12 months of 1960. The simulated predictions are output to a table, mylib.airPred, for postprocessing. In total, mylib.airPred contains 1,001 sets of values (12 rows for each set that correspond to the months of 1960):

- One set of 12 rows, which is assigned a seed value of -1, contains the usual model-based predictions for logAir.
- The other 1,000 sets of 12 rows, with a seed value of 1 through 1,000 (because of the options SEED=1 and NSIM=1000), contain the simulations of predictions of logAir.

The table mylib.airPred has three columns: SEED contains the seed values, Date contains the observation dates, and SimSmooth_logAir contains the actual predictions or their simulations. The following statements create a data set, yearly, that contains the 1,000 draws of $\text{TotalAir}_{1960} = \sum_{\text{Jan1960}}^{\text{Dec1960}} \exp(\text{forecast_logAir})$.

```
data airPred;
  set mylib.airPred;
  where seed > -1;
  simAir = exp(simsmooth_logAir);
run;
proc sort data=airPred;
  by seed date;
run;
proc means data=airPred noprint;
  by seed;
```



```
var simAir;
output out=yearly sum=TotalAir1960;
run;
```

The following statements compute the actual total number of passengers in 1960, which turns out to be 5,714:

```
proc means data=sashelp.air sum;
  where year(date)=1960;
  var air;
run;
```

By using the 1,000 draws of TotalAir1960 in the yearly table, you can easily obtain a variety of simulation-based measures of sampling distribution of TotalAir1960. For example, [Output 14.21.2](#) shows the mean, median, and a few percentiles of the sampling distribution of TotalAir1960, which is generated by the following statements:

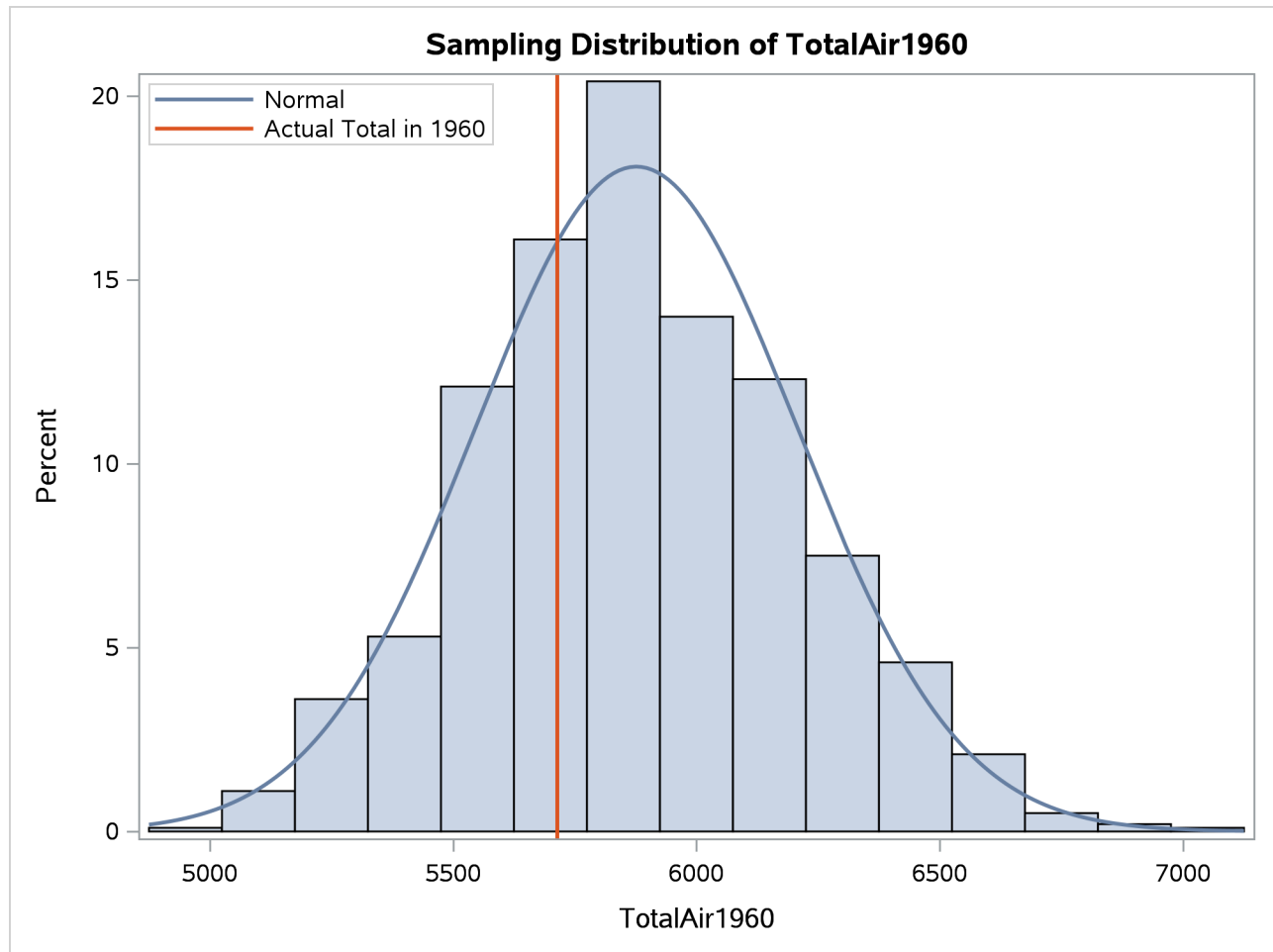
```
proc summary data=yearly print p1 q1 mean median q3 p99;
  var TotalAir1960;
run;
```

Output 14.21.2 A Few Percentiles of the Sampling Distribution of TotalAir1960

Analysis Variable : TotalAir1960					
	Lower			Upper	
1st Pctl	Quartile	Mean	Median	Quartile	99th Pctl
5132.71	5653.00	5876.41	5862.91	6102.39	6657.69

The table shows that the mean (5876.4) and the median (5862.9) of the sampling distribution are quite close to the actual total (5714) observed in 1960. Moreover, the interval (5133, 6658) can serve as a 98% confidence band for the total number of passengers in 1960. Finally, the following statements generate the graph shown in [Output 14.21.3](#), which displays the sampling distribution of TotalAir1960:

```
proc sgplot data=yearly;
  title Sampling Distribution of TotalAir1960;
  histogram TotalAir1960;
  density TotalAir1960 / legendlabel="Normal" name="Normal";
  refline 5714.00 / axis=x lineattrs=GraphFit2
    legendlabel="Actual Total in 1960" name="Coeff";
  discreteLegend "Normal" "Coeff" / across=1 location=inside;
run;
```

Output 14.21.3 Histogram of TotalAir1960

References

- Akaike, H. (1974). "A New Look at the Statistical Model Identification." *IEEE Transactions on Automatic Control* AC-19:716–723.
- Anderson, B. D., and Moore, J. B. (1979). *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall.
- Ansley, C., and de Jong, P. (2015). "Inferring and Predicting Global Temperature Trends." In *Unobserved Components and Time Series Econometrics*, edited by S. J. Koopman, and N. Shephard, 71–89. Oxford: Oxford University Press.
- Aruoba, B. S., Diebold, F. X., and Scotti, C. (2009). "Real-Time Measurement of Business Conditions." *Journal of Business and Economic Statistics* 27:417–427.
- Baltagi, B. H. (1995). *Econometric Analysis of Panel Data*. New York: John Wiley & Sons.
- Baltagi, B. H., and Levin, D. (1992). "Cigarette Taxation: Raising Revenues and Reducing Consumption." *Structural Change and Economic Dynamics* 3:321–335.

- Bell, W. R. (2011). "REGCMPNT—a Fortran Program for Regression Models with ARIMA Component Errors." *Journal of Statistical Software* 41:1–23. <http://www.jstatsoft.org/v41/i07/>.
- Box, G. E. P., and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*. Rev. ed. San Francisco: Holden-Day.
- Bozdogan, H. (1987). "Model Selection and Akaike's Information Criterion (AIC): The General Theory and Its Analytical Extensions." *Psychometrika* 52:345–370.
- Burnham, K. P., and Anderson, D. R. (1998). *Model Selection and Inference: A Practical Information-Theoretic Approach*. New York: Springer-Verlag.
- Chan, W.-Y. T., and Wallis, K. F. (1978). "Multiple Time Series Modelling: Another Look at the Mink-Muskrat Interaction." *Journal of the Royal Statistical Society, Series C* 27:168–175.
- De Jong, P. (1989). "Smoothing and Interpolation with the State-Space Model." *Journal of the American Statistical Association* 84:1085–1088.
- De Jong, P. (1991). "The Diffuse Kalman Filter." *Annals of Statistics* 19:1073–1083.
- De Jong, P., and Chu-Chun-Lin, S. (2003). "Smoothing with an Unknown Initial Condition." *Journal of Time Series Analysis* 24:141–148.
- De Jong, P., and Mazzi, S. (2001). "Modeling and Smoothing Unequally Spaced Sequence Data." *Statistical Inference for Stochastic Processes* 4:53–71.
- De Jong, P., and Penzer, J. (1998). "Diagnosing Shocks in Time Series." *Journal of the American Statistical Association* 93:796–806.
- Diggle, P. J., Liang, K.-Y., and Zeger, S. L. (1994). *Analysis of Longitudinal Data*. Oxford: Clarendon Press.
- Durbin, J., and Koopman, S. J. (2012). *Time Series Analysis by State Space Methods*. 2nd ed. Oxford: Oxford University Press.
- Eubank, R. L., Huang, C., and Wang, S. (2003). "Adaptive Order Selection for Spline Smoothing." *Journal of Computational and Graphical Statistics* 12:382–397.
- Francke, M. K., Koopman, S. J., and de Vos, A. F. (2010). "Likelihood Functions for State Space Models with Diffuse Initial Conditions." *Journal of Time Series Analysis* 31:407–414.
- Givens, G. H., and Hoeting, J. A. (2005). *Computational Statistics*. Hoboken, NJ: John Wiley & Sons.
- Hannan, E. J., and Quinn, B. G. (1979). "The Determination of the Order of an Autoregression." *Journal of the Royal Statistical Society, Series B* 41:190–195.
- Harvey, A. C. (1989). *Forecasting, Structural Time Series Models, and the Kalman Filter*. Cambridge: Cambridge University Press.
- Hurvich, C. M., and Tsai, C.-L. (1989). "Regression and Time Series Model Selection in Small Samples." *Biometrika* 76:297–307.
- Jones, R. H. (1980). "Maximum Likelihood Fitting of ARMA Models to Time Series with Missing Observations." *Technometrics* 22:389–396.

- Jones, R. H. (1993). *Longitudinal Data with Serial Correlation: A State Space Approach*. London: Chapman & Hall.
- Kohn, R., Ansley, C., and Tharm, D. (1991). “The Performance of Cross-Validation and Maximum Likelihood Estimators of Spline Smoothing Parameters.” *Journal of the American Statistical Association* 86:1042–1050.
- Kohn, R., and Ansley, C. F. (1991). “A Signal Extraction Approach to the Estimation of Treatment and Control Curves.” *Journal of the American Statistical Association* 86:1034–1041.
- Koopman, S. J., Mallee, M. I. P., and van der Wel, M. (2010). “Analyzing the Term Structure of Interest Rates Using the Dynamic Nelson-Siegel Model with Time-Varying Parameters.” *Journal of Business and Economic Statistics* 28:329–343.
- Laird, N. (2004). *Analysis of Longitudinal and Cluster-Correlated Data*. Vol. 8 of NSF-CBMS Regional Conference Series in Probability and Statistics. Beachwood, OH: Institute of Mathematical Statistics. <https://www.jstor.org/stable/4153193>.
- Nelson, C. R., and Siegel, A. F. (1987). “Parsimonious Modeling of Yield Curves.” *Journal of Business* 60:473–489.
- Pelagatti, M. M. (2015). *Time Series Modelling with Unobserved Components*. Boca Raton, FL: CRC Press.
- Reinsel, G. C. (1997). *Elements of Multivariate Time Series Analysis*. 2nd ed. New York: Springer-Verlag.
- Schwarz, G. (1978). “Estimating the Dimension of a Model.” *Annals of Statistics* 6:461–464.
- Searle, S. R., Casella, G., and McCulloch, C. E. (1992). *Variance Components*. New York: John Wiley & Sons.
- Selukar, R. S. (2010). “Estimability of the Linear Effects in State Space Models with an Unknown Initial Condition.” *Journal of Time Series Analysis* 31:167–168.
- Selukar, R. S. (2015). “Functional Modeling of Longitudinal Data with the SSM Procedure.” In *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. <http://support.sas.com/resources/papers/proceedings15/SAS1580-2015.pdf>.
- Selukar, R. S. (2017). “Detecting and Adjusting Structural Breaks in Time Series and Panel Data Using the SSM Procedure.” In *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. <http://support.sas.com/resources/papers/proceedings17/SAS0456-2017.pdf>.
- Wang, Z. (2013). “cts: An R Package for Continuous Time Autoregressive Models via Kalman Filter.” *Journal of Statistical Software* 53:1–19.
- Wecker, W. E., and Ansley, C. F. (1983). “The Signal Extraction Approach to Nonlinear Regression and Spline Smoothing.” *Journal of the American Statistical Association* 78:81–89.

Chapter 15

DEEPCAUSAL Procedure

Contents

Overview: DEEPCAUSAL Procedure	908
Using CAS Sessions and CAS Engine Librefs	909
Getting Started: DEEPCAUSAL Procedure	910
Simulation Study	910
Syntax: DEEPCAUSAL Procedure	915
Functional Summary	915
PROC DEEPCAUSAL Statement	917
DNN Statement	917
ID Statement	927
INFER Statement	927
MODEL Statement	928
PSMODEL Statement	928
SCORE Statement	929
Details: The DEEPCAUSAL Procedure	931
Causal Inference Overview	931
Estimating Unknown Functions in Propensity Score and Outcome Models	932
Estimating Parameters of Interest	933
Missing Values	938
Data Table Output	939
ODS Table Names	942
Examples: DEEPCAUSAL Procedure	942
Example 15.1: Treatment Effect of Unemployment Insurance Bonus on Unemployment Duration	942
Example 15.2: Personalized Discount Policy for an Online Media Company	948
References	952

Overview: DEEPCAUSAL Procedure

The DEEPCAUSAL procedure estimates the average causal effect and performs policy evaluation and policy comparison by using deep neural networks (DNNs). DNNs overcome several technical difficulties in the big data era, including the following:

- the huge amount of discrete or continuous potential covariates
- the unknown nonlinear relationships among the covariates, the outcome, and the treatment assignment

A problem in applying DNN to causal inference is interpretability. To solve this problem, the DEEPCAUSAL procedure applies the DNNs via a two-step semiparametric framework (see the section “[Details: The DEEPCAUSAL Procedure](#)” on page 931) and gives inferential results for the parameters of interest through the corresponding influence functions.

The DEEPCAUSAL procedure can estimate 11 types of causal effects (or parameters of interest). These include full-population causal effects, such as the average treatment effect (ATE), and subpopulation causal effects, such as the average treatment effect for the treated (ATT) and the average treatment effect for the untreated (ATU). For more information, see the sections “[Full-Population Average Effect Parameters](#)” on page 934 and “[Subpopulation Average Effect Parameters](#)” on page 936. The DEEPCAUSAL procedure can also perform policy evaluation and policy comparison. For more information, see the section “[Policy Evaluation and Comparison](#)” on page 937. PROC DEEPCAUSAL supports the SCORE statement, which enables you to save the causal model specifications and estimation results for scoring and policy evaluation and comparison without needing to reestimate the DNNs for these models.

Tools for causal analysis of nonrandomized data are available in the following procedures:

- The CAUSALGRAPH procedure enables you to analyze graphical causal models and to construct sound statistical strategies for causal effect estimation. For more information, see Chapter 37, “The CAUSALGRAPH Procedure” (*SAS/STAT User’s Guide*).
- The PSMATCH procedure enables you to perform propensity score analyses and to assess covariate balance. For more information, see Chapter 101, “The PSMATCH Procedure” (*SAS/STAT User’s Guide*).
- The CAUSALTRT procedure enables you to perform estimation of a causal effect. For more information, see Chapter 39, “The CAUSALTRT Procedure” (*SAS/STAT User’s Guide*).
- The CAUSALMED procedure enables you to decompose a (total) causal effect into direct and indirect effects. For more information, see Chapter 38, “The CAUSALMED Procedure” (*SAS/STAT User’s Guide*).
- The MODEL, PANEL, QLIM, and TMODEL procedures in SAS/ETS and the CPANEL and CQLIM procedures in SAS Econometrics support endogenous variables and instrumental variables. For more information, see Chapter 24, “MODEL Procedure” (*SAS/ETS User’s Guide*), Chapter 25, “PANEL Procedure” (*SAS/ETS User’s Guide*), Chapter 27, “QLIM Procedure” (*SAS/ETS User’s Guide*), Chapter 39, “TMODEL Procedure” (*SAS/ETS User’s Guide*), Chapter 11, “CPANEL Procedure,” and Chapter 12, “CQLIM Procedure.”

- The VARMAX procedure supports Granger causality tests. For more information, see Chapter 42, “VARMAX Procedure” (*SAS/ETS User’s Guide*).

PROC DEEPCAUSAL requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: DEEPCAUSAL Procedure

This section provides an example that the DEEPCAUSAL procedure supports.

Simulation Study

This example illustrates how the DEEPCAUSAL procedure estimates the parameters of interest and performs policy evaluation and policy comparison. It also demonstrates how you can save the model information and use it later to score other new data. A simulated data set is generated, based on the following assumptions:

- The treatment variable, denoted by T , is binary (0 or 1).
- The outcome variable, denoted by Y , is continuous.
- There are 20 covariates: $x_j \sim \text{Uniform}(0, 1)$ for $j = 1, \dots, 20$.
- Four of the covariates are randomly selected and used in the outcome model. The first selected variable is used in the propensity score model.

The outcome model is given as

$$Y = \alpha(\mathbf{X}) + \beta(\mathbf{X})t + \epsilon, \quad \epsilon \stackrel{\text{iid}}{\sim} N(0, 1)$$

$$\alpha(X) = 1 + \cos(2\pi \sum_{i=1}^4 w_i x_{s_i})$$

$$\beta(X) = \sin(2\pi \sum_{i=1}^4 w_i x_{s_i})$$

where $\{s_i\}_{i=1}^4$ denotes the indices of the four covariates that are randomly selected from the 20 covariates and $\mathbf{X}$ denotes the covariates vector $\{x_j\}_{j=1}^{20}$.

The propensity score model is given as

$$\Pr(T = 1 | \mathbf{X} = \mathbf{x}) = p(\mathbf{x}) = 0.1 + 0.8 \sin(2\pi |x_{s_1} - 0.5|)$$

where T denotes the treatment variable.

The following statements simulate 20,000 samples that are based on the propensity score and outcome models that were just presented:

```
%macro deepEconGenerateData(n,d,eff,seed);
  data deepEconDGP;
    array x[&d.] x1-x&d.;
    array w[&d.] w1-w&d.;
    array selectedX[&eff.] selectedX1-selectedX&eff.;
    retain w;
    call streaminit('pcg', &seed.);
    * weights;
```



```

    sumw = 0;
    do i = 1 to &eff.;
        w[i] = rand('Uniform');
        sumw = sumw + w[i];
    end;
    do i = 1 to &eff.;
        w[i] = w[i] / sumw;
    end;
    * select x;
    do i = 1 to &eff.;
        selectedX[i] = ceil(&d.*rand('Uniform'));
    end;
    do i = 1 to &n.;
        * covariates;
        do j = 1 to &d.;
            x[j] = rand('Uniform');
        end;
        * treatment assignment;
        u = rand('Uniform');
        sumx = 0;
        do j = 1 to &eff.;
            sumx = sumx + w[j]*x[selectedX[j]];
        end;
        px = (0.1 + 0.8*sin(2*abs(x[selectedX[1]]-0.5)*3.1415926));
        if(u<=px) then t=1;
        else t=0;
        * outcome;
        eps = rand('Normal');
        mu = 1 + cos(sumx*2*3.1415926);
        tau = sin(sumx*2*3.1415926);
        y = mu + tau*t + eps;
        s1=(tau>0);
        s0=1-s1;
        output;
    end;
run;
%mend;

%let n=20000;
%let d=20;
%let eff=4;
%let seed=12346;
%deepEconGenerateData(&n., &d., &eff., &seed.);

```

The following DATA steps load the first 10,000 observations in the data set `deepEconDGP` into the data table `simCausal_train` and the second 10,000 observations into the data table `simCausal_test` in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```

data mylib.simCausal_train;
    set deepEconDGP(obs=10000 keep=i x1-x20 t y px mu tau s0 s1
                    selectedX1-selectedX4);
run;

```

```
data mylib.simCausal_test;
  set deepEconDGP(firstobs=10001 keep=i x1-x20 t y px mu tau s0 s1
    selectedX1-selectedX4);
run;
```

The unknown functions $p(\cdot)$, $\alpha(\cdot)$, and $\beta(\cdot)$ are estimated through two DNNs by using the data set `simCausal_train`. The parameters of interest, policy evaluation, and policy comparison are then estimated. The following statements use PROC DEEPCAUSAL to get these estimates and store the model information and parameter learning results:

```
proc deepcausal data=mylib.simCausal_train;
  id i;
  psmodel t=x1-x20 / dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=2000 regL1=0.0001 maxEpochs=1000
      algorithm=adam(learningRate=0.0001)) nthreads=5 seed=12345));
  model y=x1-x20 / dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=2000 regL1=0.001 maxEpochs=1000
      algorithm=adam) nthreads=5 seed=12345));
  infer policy=(s1 s0) policyComparison=(base=(t) compare=(s0 s1))
    out=mylib.oest outdetails=mylib.odetails;
  score outps=mylib.outpsdata outa=mylib.outadata outb=mylib.outbdata;
run;
```

The OUTPS= option writes the propensity score model information to the data table `outpsdata`, the OUTA= option writes the $\alpha(\cdot)$ estimation information to the data table `outadata`, and the OUTB= option writes the $\beta(\cdot)$ estimation information to the data table `outbdata`. The number of observations is shown in Figure 15.1.

Figure 15.1 Number of Observations

The DEEPCAUSAL Procedure

Number of Observations	10000
Observations with Missing Covariates	0
Observations with Missing Covariates for PS Model	0
Observations with Missing Covariates for Outcome Model	0
Observations with Missing Outcome	0

The estimates of parameters of interest are shown in Figure 15.2.

Figure 15.2 Parameters of Interest

Population Parameter Estimates								
Parameter	Potential Outcome	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z	
Average	0	0.351933	0.016247	0.320089	0.383777	21.66	<.0001	
Average	1	0.361219	0.015255	0.331319	0.391119	23.68	<.0001	
ATE		0.009286	0.021574	-0.032998	0.051570	0.43	0.6669	

Subpopulation Parameter Estimates								
Parameter	Potential Outcome	Treatment Level	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
Conditional Average	0	0	0.363550	0.017686	0.328885	0.398214	20.56	<.0001
Conditional Average	0	1	0.344208	0.016541	0.311788	0.376628	20.81	<.0001
Conditional Average	1	0	0.368315	0.018645	0.331772	0.404859	19.75	<.0001
Conditional Average	1	1	0.356500	0.016645	0.323877	0.389123	21.42	<.0001
ATT		1	0.012292	0.022012	-0.030850	0.055435	0.56	0.5765
ATU		0	0.004766	0.023280	-0.040863	0.050394	0.20	0.8378
Composition Effect	1		-0.011815	0.017386	-0.045891	0.022261	-0.68	0.4968
Observed Difference			-0.007049	0.024815	-0.055686	0.041587	-0.28	0.7764

The policy evaluation estimates the expected utility of a policy. The policy evaluation results are shown in Figure 15.3.

Figure 15.3 Policy Evaluation

Policy Evaluation						
Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
s1	0.653567	0.015892	0.622419	0.684715	41.13	<.0001
s0	0.059585	0.015050	0.030087	0.089083	3.96	<.0001

The policy comparison estimates the difference in expected utility of two policies. The policy comparison results are shown in Figure 15.4.

Figure 15.4 Policy Comparison

Policy Comparison						
Comparison Policy	Base Policy	Estimate	Standard Error	95% Confidence Limits		Z Pr > Z
s0	t	-0.306508	0.015133	-0.336168	-0.276848	-20.25 <.0001
s1	t	0.287473	0.014791	0.258483	0.316464	19.44 <.0001

You can apply the previously saved propensity score model, the $\alpha(\cdot)$ estimation information, and the $\beta(\cdot)$ estimation information to a new data set by specifying the INPS=, INA=, and INB= options in the SCORE statement. The following statements use the outpsdata, outadata, and outbdata data tables as input data tables and estimate the parameters of interest, policy evaluation, and policy comparison by using the simCausal_test data set:

```

proc deepcausal data=mylib.simCausal_test;
  id i;
  infer policy=(s1 s0) policyComparison=(base=(t) compare=(s0 s1))
    out=mylib.oest2 outdetails=mylib.odetails2;
  score inps=mylib.outpsdata ina=mylib.outadata inb=mylib.outbdata;
run;

```

When you specify the INPS=, INA=, and INB= options in the SCORE statement, you do not need to specify the PSMODEL or MODEL statement. The estimates of parameters of interest, policy evaluation, and policy comparison are shown in [Figure 15.5](#).

Figure 15.5 Parameters of Interest, Policy Evaluation, and Policy Comparison

The DEEPCASUAL Procedure

Population Parameter Estimates								
Parameter	Potential Outcome	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z	
Average	0	0.370338	0.016174	0.338637	0.402038	22.90	<.0001	
Average	1	0.332223	0.015308	0.302220	0.362227	21.70	<.0001	
ATE		-0.038115	0.021562	-0.080375	0.004146	-1.77	0.0771	

Subpopulation Parameter Estimates								
Parameter	Potential Outcome	Treatment Level	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
Conditional Average	0	0	0.384077	0.017848	0.349096	0.419058	21.52	<.0001
Conditional Average	0	1	0.361401	0.016303	0.329448	0.393355	22.17	<.0001
Conditional Average	1	0	0.351818	0.018954	0.314669	0.388966	18.56	<.0001
Conditional Average	1	1	0.319478	0.016443	0.287251	0.351705	19.43	<.0001
ATT		1	-0.041923	0.021779	-0.084609	0.000763	-1.92	0.0542
ATU		0	-0.032260	0.023603	-0.078521	0.014002	-1.37	0.1717
Composition Effect	1		-0.032339	0.017249	-0.066147	0.001468	-1.87	0.0608
Observed Difference			-0.064599	0.024768	-0.113143	-0.016055	-2.61	0.0091

Policy Evaluation						
Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
s1	0.646127	0.015751	0.615255	0.676998	41.02	<.0001
s0	0.056434	0.015183	0.026675	0.086193	3.72	0.0002

Policy Comparison							
Comparison Policy	Base Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
s0	t	-0.302291	0.015035	-0.331758	-0.272824	-20.11	<.0001
s1	t	0.287402	0.014887	0.258224	0.316580	19.31	<.0001

For more information about the estimates of parameters of interest, policy evaluation, and policy comparison, see the sections “[Full-Population Average Effect Parameters](#)” on page 934, “[Subpopulation Average Effect Parameters](#)” on page 936, and “[Policy Evaluation and Comparison](#)” on page 937.

Syntax: DEEPCAUSAL Procedure

The following statements are available in the DEEPCAUSAL procedure. Items within angle brackets (<>) are optional.

```
PROC DEEPCAUSAL options ;
  DNN options ;
  ID variable ;
  INFER <options> ;
  MODEL outcome-variable <=covariates> </options> ;
  PSMODEL treatment-variable <=covariates> </options> ;
  SCORE options ;
```

Functional Summary

The statements and options available in the DEEPCAUSAL procedure are summarized in [Table 15.1](#). The sections after the table provide a description of each statement in alphabetical order, following the PROC DEEPCAUSAL statement.

Table 15.1 Functional Summary

Description	Statement	Option
Input Data Table Option		
Specifies the input data table	PROC DEEPCAUSAL	DATA=
Specifies the input data table for $\alpha(\cdot)$ estimation information	SCORE	INA=
Specifies the input data table for $\beta(\cdot)$ estimation information	SCORE	INB=
Specifies the input data table for outcome model information	SCORE	INO=
Specifies the input data table for propensity score model information	SCORE	INPS=
Output Data Table Options		
Writes the estimation of the parameters of interest, policy evaluation, and policy comparison to an output data set	INFER	OUTEST=
Writes the details of the estimation, including $\alpha(\cdot)$, $\beta(\cdot)$, $p(\cdot)$, the residual, and influence functions for each unit, to an output data set	INFER	OUTDETAILS=
Writes the $\alpha(\cdot)$ estimation information to the specified output data set	SCORE	OUTA=
Writes the $\beta(\cdot)$ estimation information to the specified output data set	SCORE	OUTB=

Table 15.1 *continued*

Description	Statement	Option
Writes the outcome model information to the specified output data set	SCORE	OUTO=
Writes the propensity score model information to the specified output data set	SCORE	OUTPS=
ID Variable		
Specifies the identifying variable	ID	
Deep Neural Network (DNN) Options		
Specifies the parameters of the DNN for the propensity score model	PSMODEL	DNN=
Specifies the parameters of the DNN for the outcome model	MODEL	DNN=
Specifies the number of nodes for all hidden layers of the DNN for the propensity score model and outcome model	DNN	NODES=
Specifies the training options of the DNN for the propensity score model and outcome model	DNN	TRAIN=
Outcome Model Variables		
Specifies the outcome variable	MODEL	<i>outcome-variable</i>
Specifies the covariates for the outcome model	MODEL	<i>covariates</i>
Propensity Score Model Variables		
Specifies the treatment variable	PSMODEL	<i>treatment-variable</i>
Specifies the covariates for the propensity score model	PSMODEL	<i>covariates</i>
Inference Options		
Specifies the list of policy variables to be evaluated	INFER	POLICY=
Specifies the parameters for policy comparison	INFER	POLICYCOMPARISON=

PROC DEEPCAUSAL Statement

PROC DEEPCAUSAL *options* ;

The PROC DEEPCAUSAL statement invokes the DEEPCAUSAL procedure. You must specify the following *option*:

DATA=*libref.data-table*

names the input data table for PROC DEEPCAUSAL to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

data-table specifies the name of the input data table.

NOTE: The input data table must not contain any of the reserved variables (case-insensitive): *_PSHAT_*, *_YHAT_*, *_Y0_*, *_Y1_*, *_AVGPX_*, *_px_*, *_alpha_*, *_beta_*, *_rho00_*, *_rho01_*, *_rho10_*, *_rho11_*, *_att_*, *_deltaX_*, *_atu_*, or *_delta_*.

DNN Statement

DNN *< options >* ;

The DNN statement specifies the parameters for the deep neural network (DNN). It includes options that are related to the DNNs for the propensity score model and outcome models. You can specify the following *options*:

NODES=*(numeric-list)*

specifies the number of nodes for each hidden layer in the DNN, where each value in the *numeric-list* must be a positive number and values are separated by commas or spaces. The length of the list specifies the number of layers, and each value specifies the number of nodes for the corresponding layer. If the length of the list is denoted by n , the i th ($1 \leq i \leq n$) value denotes the number of nodes for the i th layer. By default, NODES=(32 32 32 32), which yields four hidden layers, with 32 nodes in each layer.

TRAIN=*(train-options)*

specifies the train options for the DNN. You can specify one or more of the following *train-options*:

GPU=*(< gpu-options >)*

specifies that the DNN perform calculations by using graphics processing unit (GPU) hardware. You can specify the following *gpu-options*:

DEVICE=(*numeric-list*)

DEVICES=(*numeric-list*)

specifies a list of GPU devices to use.

PRECISION=FP16 | FP32

specifies the computational precision in forward-backward computations. You can specify the following values:

FP16 uses half-precision (16-bit floating-point) computations to train and score the DNN. The lower computational precision in forward-backward computations can encourage tensor core engagements.

FP32 uses single-precision (32-bit floating-point) computations to train and score the DNN.

By default, PRECISION=FP32.

USEEXCLUSIVE

specifies that only GPU devices be used.

USETensorRT

enables the use of the TensorRT software development kit for fast inference.

MISSING=MAX | MEAN | MIN | NONE

specifies the policy to use for replacing missing input covariate values with imputed values. You can specify one of the following values:

MAX replaces missing values with the maximum of the corresponding covariate.

MEAN replaces missing values with the mean of the corresponding covariate.

MIN replaces missing values with the minimum of the corresponding covariate.

NONE ignores observations that have missing values.

By default, MISSING=MEAN. The MISSING= option is for DNN training only. In the later inference, observations that have missing covariates are removed.

NTHREADS=*number*

specifies the number of threads to use for the DNN.

OPTIMIZATION=(*optimizer-options*)

OPTIMIZE=(*optimizer-options*)

OPTIMIZER=(*optimizer-options*)

specifies the settings for the optimization algorithm, optimization mode, and other settings, such as a seed, the maximum number of epochs, and so on. You can specify the following *optimizer-options*:

ALGORITHM=*algorithm-keyword*<(common-algorithm-options)>

specifies the optimization algorithm and the corresponding options. First, you can specify some common options of all the optimization algorithms, *common-algorithm-options*, in the parentheses that follow the *algorithm-keyword*. They are as follows:

CLIPGRADMAX=*number*

specifies the maximum gradient value. All gradient values that exceed the specified maximum value are set to the specified maximum value. The *number* should be greater than or equal to 0.

CLIPGRADMIN=*number*

specifies the minimum gradient value. All gradient values that are less than the specified minimum value are set to the specified minimum value. The *number* should be less than or equal to 0.

FCMPEARLYSTOPPING=*symbol-name***FCMPES=***symbol-name*

specifies the FCMP early stopping function.

FCMPLEARNINGRATE=*symbol-name***FCMPLR=***symbol-name*

specifies the FCMP learning rate function.

GAMMA=*number*

specifies the gamma value for the learning rate policy. The *number* must be between 0 and 100. By default, GAMMA=0.1.

LEARNINGRATE=*number*

specifies the learning rate (a positive value) for stochastic gradient descent (SGD). By default, LEARNINGRATE=0.001.

LEARNINGRATEPOLICY=FIXED | INV | MULTISTEP | POLY | STEP**LRPOLICY=**FIXED | INV | MULTISTEP | POLY | STEP

specifies the learning rate policy. You can specify one of the following values:

FIXED

specifies a fixed learning rate.

INV

sets the learning rate option value after each epoch, according to the initial learning rate, the value of the GAMMA= option, and the value of the POWER= option. The rate is calculated as $\text{learningRate} * (1 + \text{gamma} * \text{currentEpoch})^{-\text{power}}$.

MULTISTEP

sets the learning rate after each of the epochs that you specify in the STEPS= option. The updated learning rate is the product of the previous learning rate value and the GAMMA= option value.

POLY

sets the learning rate option value after each epoch, according to the initial learning rate, the maximum number of epochs, and the value of the POWER= option. The rate is calculated as $\text{learningRate} * (1 - \text{currentEpoch}/\text{maxEpochs})^{\text{power}}$.

STEP sets the learning rate option by multiplying the current learning rate option value by the GAMMA= option value. The number of steps is specified in the STEPSIZE= option. The configured learning rate is recalculated for each group of epochs, according to the step size.

By default, LEARNINGRATEPOLICY=FIXED.

NOUSELOCKING

specifies that gradients be computed asynchronously with multiple threads.

POWER=number

specifies the power for the learning rate policy. The *number* must be nonnegative. By default, POWER=0.75.

STEPS=(numeric-list)

specifies a list of epoch counts. When the current epoch matches one of the specified steps, the learning rate is multiplied by the value of the GAMMA= option. For example, if you specify STEPS=(5 9 13), then the learning rate is multiplied by gamma after the 5th, 9th, and 13th epochs.

STEPSIZE=number

specifies the step size (a value greater than or equal to 1) when the learning rate policy is set to STEP. By default, STEPSIZE=10.

For the ALGORITHM= option, you can specify one of the following *algorithm-keywords* and its associated options:

ADAM<(adam-options)>

ADAMSGD<(adam-options)>

specifies the ADAM optimization algorithm. For the *adam-options*, you can specify one or more of the *common-algorithm-options*. You can also specify the following *adam-options*:

BETA1=number

specifies the exponential decay rate for the first moment in an ADAM learning algorithm. The *number* must be in the range [0, 1). By default, BETA1=0.9.

BETA2=number

specifies the exponential decay rate for the second moment in an ADAM learning algorithm. The *number* must be in the range [0, 1). By default, BETA2=0.999.

LARS<(lars-options)>

specifies the LARS optimization algorithm. For the *lars-options*, you can specify one or more of the *common-algorithm-options*. You can also specify one or more of the following *lars-options*:

LAYERRATES=(*numeric-list*)

LOCALRATES=(*numeric-list*)

produces an array of values that specifies the learning rate for each model layer. You cannot specify this option in the DNN statement; you can specify it only in the PSMODEL or MODEL statement. In the PSMODEL statement, the number of local learning rates that you specify in the *numeric-list* must equal $m + 2$, where m is the number of hidden layers of the DNN for the propensity score model. In the MODEL statement, the number of local learning rates that you specify in the *numeric-list* must equal $n + 9$, where n is the number of hidden layers of the DNN for the outcome model.

MOMENTUM=*number*

specifies the momentum parameter value for stochastic gradient descent (SGD). The *number* must be in the range $[0, 1]$. By default, MOMENTUM=0.9.

SCALEFACTOR=*number*

specifies the scaling coefficient that is used to adjust the base learning rate. The *number* must be in the range $[0, 1]$. By default, SCALEFACTOR=0.001.

WARMUP=*number*

specifies the number of warmup epochs to complete before applying layerwise learning rates in the LARS optimization algorithm. The *number* must be greater than or equal to -1 . By default, WARMUP= -1 , which means that there are no warmup epochs.

LBFGS<(*lbfgs-options*)>

specifies the L-BFGS optimization algorithm. In the *lbfgs-options*, you can specify one or more of the *common-algorithm-options*. The GAMMA=, LEARNINGRATE=, LEARNINGRATEPOLICY=, POWER=, and STEPSIZE= options do not support the L-BFGS optimization algorithm. You can also specify one or more of the following *lbfgs-options*:

BACKTRACKRATIO=*number*

specifies the backtrack ratio of line search iterations for the L-BFGS optimization algorithm. The *number* must be in the range $[0, 1)$. By default, BACKTRACKRATIO=0.5.

M=*number*

NUMCORRECTIONS=*number*

specifies the number of corrections to use in the L-BFGS update. The *number* must be greater than or equal to 0. By default, M=6.

MAXLINESEARCHITERS=*number*

specifies the maximum number of line search iterations for L-BFGS optimization algorithm. The *number* must be an integer greater than or equal to 1. By default, MAXLINESEARCHITERS=60.

MOMENTUMALG<(momentumalg-options)>

MOMENTUMSGD<(momentumalg-options)>

specifies the MOMENTUM optimization algorithm. In the *momentumalg-options*, you can specify one or more of the *common-algorithm-options*. You can also specify the following *momentumalg-option*:

MOMENTUM=*number*

specifies the momentum parameter value for stochastic gradient descent (SGD). The *number* must be between 0 and 1. By default, MOMENTUM=0.9.

TRISH<(trish-options)>

TRISHSGD<(trish-options)>

specifies the trust-region-ish (TRish) optimization algorithm. In the *trish-options*, you can specify one or more of the *common-algorithm-options*. You can also specify one or more of the following *trish-options*:

GAMMA1=*number*

specifies the Gamma1 scaling for the gradient. The *number* must be in the range [0, 10,000). By default, GAMMA1=2.

GAMMA2=*number*

specifies the Gamma2 scaling for the gradient. The *number* must be in the range [0, 10,000). By default, GAMMA2=2.

MOMENTUM=*number*

specifies the momentum parameter value for stochastic gradient descent (SGD). The *number* must be in the range [0, 1]. By default, MOMENTUM=0.9.

VANILLA<(common-algorithm-options)>

VANILLASGD<(common-algorithm-options)>

specifies the vanilla optimization algorithm.

By default, ALGORITHM=VANILLA.

COMPRESSION=(compression-options)

sparsifies the communication between workers for faster training. You can specify one or more of the following *compression-options*:

LOCALMOMENTUM=*number*

determines the strength of aggregation on the masked gradient elements when compression is used. The *number* must be in the range [0, 1]. By default, LOCALMOMENTUM=0.95.

SAMPLERATIO=*number*

determines the number of samples for gradients in each layer. The *number* must be in the range $[10^{-5}, 0.2]$. By default, SAMPLERATIO=0.05.

SPARSITYRATIO=number

determines the number of samples for gradients in each layer. The *number* must be in the range [0, 1]. By default, SPARSITYRATIO=0.75.

DROPOUT=number

specifies the probability that the output of a neuron in a fully connected layer will be set to 0 during training. The specified probability is recalculated each time an observation is processed. The *number* must be in the range [0, 1). By default, DROPOUT=0.

DROPOUTINPUT=number

specifies the probability that an input variable will be set to 0 during training. The specified probability is recalculated each time an observation is processed. The *number* must be in the range [0, 1). By default, DROPOUTINPUT=0.

DROPOUTTYPE=INVERTED | STANDARD

specifies what type of dropout to use. You can specify one of the following types:

INVERTED	specifies the inverted dropout, in which activations of some neurons are set to 0 and the remaining ones are scaled.
STANDARD	specifies the standard dropout, in which activations of some neurons are set to 0 and the remaining ones are not scaled.

By default, DROPOUTTYPE=STANDARD.

FCONV=number

specifies the relative function convergence criterion. If the relative loss error, $\text{abs}[(\text{previous_loss} - \text{current_loss})/\text{previous_loss}]$, does not result in a change in the objective function, the optimization stops. The *number* must be greater than or equal to 0. By default, FCONV=0, and the relative function convergence is not checked.

FREEZEBATCHNORMSTATS**FREEZENSTATS**

freezes statistics of all batch normalization layers.

IGNORETRAININGERROR

specifies that the model training continue without interruption by ignoring the following issue: there are training data observations that contain invalid or missing variable data. If you omit this option, training stops, and PROC DEEPCAUSAL terminates if bad input data are found.

MAXEPOCHS=number

specifies the maximum number of epochs. The *number* must be greater than or equal to 1. By default, MAXEPOCHS=1. For SGD with a single-machine server, or for a session that uses one worker on a distributed server, one epoch is reached when the optimizer passes through the data one time. For a session that uses more than one worker, one epoch is reached when all the workers exchange the weights with the controller one time. The SYNCFREQ= option specifies the number of times that each worker passes through the data before exchanging weights with the controller. For the L-BFGS optimization algorithm with a full batch, each iteration might process more than one epoch, and the final number of epochs might exceed the parameter value for the maximum number of epochs.

MINIBATCHBUFSIZE=*number*

specifies the number of observations to buffer in memory (including input data and intermediate calculations) before processing the remaining records in the minibatch. The *number* must be greater than or equal to 1 and must be smaller than the *number* that you specify for the MINIBATCHSIZE= option. The MINIBATCHBUFSIZE= option setting is ignored on GPUs.

MINIBATCHSIZE=*number*

specifies the number of observations per thread in a minibatch. The parameter controls the number of observations used

per worker in each thread to compute the gradient, prior to updating the weights. Larger values use more memory. The *number* is greater than or equal to 1. By default, MINIBATCHSIZE=1. When you use synchronous SGD (the default), the total minibatch size is calculated as $\text{miniBatchSize} * \text{number of threads} * \text{number of workers}$. When you use asynchronous SGD by specifying the ELASTICSYNCFREQ= option, each worker trains its own local model. In this case, the total minibatch size for each worker is calculated as $\text{miniBatchSize} * \text{number of threads}$.

MODE=DOWNPOUR | ELASTIC | SYNCHRONOUS

specifies the optimization mode and the corresponding options. You can specify one of the following values:

DOWNPOUR

specifies the downpour optimization mode.

ELASTIC<(elastic-options)>

specifies the elastic optimization mode. You can specify the following *elastic-options*:

ALPHA=*number***ELASTICALPHA=*number***

specifies the significance level (a value between 0 and 1) that is used for elastic SGD. By default, ALPHA=0.

ELASTICSYNCFREQ=*number***SYNCFREQ=*number***

specifies the frequency (a nonnegative value) for communication between the workers and controller for exchanging weights. You can exchange the weights more often than once each epoch by specifying a *number* that is less than the number of batches in an epoch. If the *number* is greater than the number of batches in an epoch, then the weights are exchanged once for each epoch. By default, ELASTICSYNCFREQ=0.

SYNCHRONOUS<(SYNCFREQ=*number*)>

specifies the synchronous optimization mode. The SYNCFREQ= option, which is not required, specifies the synchronization frequency for SGD in terms of epochs. By default, SYNCFREQ=1.

By default, MODE=SYNCHRONOUS.

NOBNSRCLAYERWARNINGS

suppresses the warning, if the batch normalization source layer has an atypical type, activation, or includeBias setting.

REGL1=number

specifies the weight for the L1 regularization term. The *number* must be greater than or equal to 0. By default, REGL1=0. At the default setting, PROC DEEPCAUSAL does not perform L1 regularization. Setting the L1 regularization weight to 0 disables the regularization. Initial L1 weight values should be small (such as $1E-3$). You can combine L1 regularization with L2 regularization.

REGL2=number

specifies the weight for the L2 regularization term. The *number* must be greater than or equal to 0. By default, REGL2=0. At the default setting, PROC DEEPCAUSAL does not perform L2 regularization. Setting the L2 regularization weight to 0 disables the regularization. Initial L2 weight values should be small (such as $1E-3$). You can combine L2 regularization with L1 regularization.

SEED=number

specifies the random number seed value for the random number generator in the L-BFGS optimization algorithm. Seed values less than or equal to 0 generate random number streams by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences. By default, SEED=0.

SNAPSHOTFREQ=number

specifies the frequency for generating snapshots of the neural weights and storing the weights in a weight table during the training process. When asynchronous SGD is specified, PROC DEEPCAUSAL synchronizes all the weights before writing out the weights. The *number* must be greater than or equal to 0. By default, SNAPSHOTFREQ=0.

STAGNATION=number

specifies the number of iterations to complete without improvement before stopping the optimization early. The *number* must be greater than or equal to 0. By default, STAGNATION=0; this turns off the stagnation monitoring.

THRESHOLD=number

specifies the threshold that is used to determine whether the loss error or validation scores are improving or stagnating across iterations. The iteration does not improve when the magnitude of the score change between successive iterations is less than or equal to the absolute value of `current_score * threshold value`. When an iteration does not improve, the stagnation counter increments. Otherwise, the stagnation counter is set to 0. The *number* must be greater than or equal to 0. By default, THRESHOLD= 10^{-8} .

TOTALMINIBATCHSIZE=(totalminibatchsize-options)

specifies the number of observations in a minibatch. You can specify one or more of the following *totalminibatchsize-options*:

NONOTIFY

suppresses the message that is printed if the total minibatch size is rounded up.

ROUND

rounds up the total minibatch size.

SIZE=number**N=number**

specifies the number of observations in a minibatch. The computation for the total minibatch size is $nWorkers * miniBatchSize * nThreads$. For example, a two-worker scenario with 32 threads per worker and a minibatch size of 10 has a total minibatch size of $2 * 32 * 10 = 640$ records.

You can use these options to control the number of observations that the DNN uses to compute the gradient prior to updating the weights. Larger values use more memory. If the specified size cannot be evenly divided by the number of threads (if you are using asynchronous SGD) or by the number of threads * number of workers (if you are using synchronous SGD), then PROC DEEPCAUUSAL terminates with an error unless you specify the ROUND option. In that case, the total minibatch size is rounded up so that it is evenly divided.

RECORDSEED=number

specifies the random number seed for random record selection within a worker. The default (RECORDSEED=0) disables random record selection. Records are read in the order in which they are written in memory. Seed values that are less than or equal to 0 generate random number streams by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences.

SEED=number

specifies the random number seed value for the random number generator in stochastic gradient descent (SGD). Seed values that are less than or equal to 0 generate random number streams by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences. By default, SEED=0.

TARGETMISSING=MAX | MEAN | MIN | NONE

specifies the policy to use for replacing missing outcome variables with imputed values. You can specify one of the following values:

MAX	replaces missing values with the maximum.
MEAN	replaces missing values with the mean value.
MIN	replaces missing values with the minimum.
NONE	ignores observations that have missing target variables.

By default, TARGETMISSING=MEAN. The TARGETMISSING= option is for DNN training only. In the later inference, observations that have missing outcome variables have missing values for residuals and for any estimates that need residuals.

ID Statement

ID *variable* ;

The ID statement specifies the ID variable in the data set. The *variable* must have a unique value for each row in the input data table, or PROC DEEPCAUSAL issues an error.

INFER Statement

INFER *options* ;

The INFER statement specifies the parameters for inference. You can specify one or more of the following *options*:

OUT=*libref.data-table*

OUTEST=*libref.data-table*

writes the estimation of the parameters of interest, policy evaluation, and policy comparison to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

OUTDETAILS=*libref.data-table*

writes the details of the estimation, including $\alpha(\cdot)$, $\beta(\cdot)$, $p(\cdot)$, the residual, and influence functions for each unit, to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

POLICY=(*policy-variable-list*)

specifies the list of policy variables to be evaluated. All symbol names in the *policy-variable-list* must correspond to the policy variables; the symbol names are separated by spaces.

POLICYCOMPARISON=(*comparison-options*)

specifies the options for policy comparison. You must specify the following *comparison-options*:

BASE=(*base-list*)

specifies a list of policy variables, each of which is used as a base policy in the policy comparison. All symbol names in the *base-list* must correspond to the policy variables; the symbol names are separated by spaces.

COMPARE=(*compare-list*)

specifies a list of policy variables, each of which is used to compare with each base policy that you specify in the BASE= option. All symbol names in the *compare-list* must correspond to the policy variables; the symbol names are separated by spaces.

MODEL Statement

MODEL *outcome-variable* <=*covariates*> </options> ;

The MODEL statement specifies options that are related to the outcome model and the DNN to fit the outcome model. The *outcome-variable* specifies the outcome variable of the outcome model. The *covariates* specify the covariates of the outcome model. If you omit the *covariates*, no DNN is trained for the outcome model. For more information about the outcome model, see the section “[Outcome Model](#)” on page 932. Covariate names must be separated by spaces. The treatment variable is automatically included in the outcome model as an independent variable.

You can specify the following *option* after a forward slash (/):

DNN=(*dnn-options*)

specifies the options of the DNN for the outcome model. If you omit this option and specify the DNN statement, the DNN for the outcome model has the options that you specify in the DNN statement. If you omit both this option and the DNN statement, the DNN for the outcome model uses the default option values. If you specify both this option and the DNN statement, the DNN statement is ignored for the outcome model. For the *dnn-options*, you can specify one or more options that are listed in the section “[DNN Statement](#)” on page 917. If no covariates are specified in the MODEL statement, the DNN= option is ignored.

PSMODEL Statement

PSMODEL *treatment-variable* <=*covariates*> </options> ;

The PSMODEL statement specifies options that are related to the propensity score model and the DNN to fit the propensity score model. The *treatment-variable* specifies the treatment variable of the propensity score model. It corresponds to one variable in the input data table; that variable must satisfy the following requirements:

- The treatment variable in the input data table must be binary; that is, its value must be either 0 or 1.
- The values of the treatment variable across all observations in the input data table must not be identical.

The *covariates* specify the covariates of the propensity score model. All covariate names are separated by spaces. If you omit the *covariates*, no DNN is trained for the propensity score model. For more information about the propensity score model, see the section “[Propensity Score Model](#)” on page 932.

You can specify the following *option* after a forward slash (/):

DNN=(*dnn-options*)

specifies the options of the DNN for the propensity score model. If you omit this option and specify the DNN statement, the DNN for the propensity score model has the options that you specify in the DNN statement. If you omit both this option and the DNN statement, the DNN for the propensity score model uses the default option values. If you specify both this option and the DNN statement, the DNN statement is ignored for the propensity score model. For the *dnn-options*, you can specify one or more options that are listed in the section “[DNN Statement](#)” on page 917. If no *covariates* are specified in the PSMODEL statement, the DNN= option is ignored.

SCORE Statement

SCORE options ;

The SCORE statement specifies options that are related to scoring data. You can specify the following options:

INA=*libref.data-table*

INALPHA=*libref.data-table*

specifies the input data table that contains information about $\alpha(\cdot)$ estimation. The data table must be an output data table that the OUTA= option produces in a previous call of the DEEPCAUSAL procedure. When you specify this option, the MODEL statement and the OUTA=, OUTB=, and OUTO= options in the SCORE statement are ignored. If this option is specified, you must also specify the INB= option. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

INB=*libref.data-table*

INBETA=*libref.data-table*

specifies the input data table that contains information about $\beta(\cdot)$ estimation. The data table must be an output data table that the OUTB= option produces in a previous call of the DEEPCAUSAL procedure. When you specify this option, the MODEL statement and the OUTA=, OUTB=, and OUTO= options in the SCORE statement are ignored. If this option is specified, you must also specify the INA= option. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

INO=*libref.data-table*

INOUTCOME=*libref.data-table*

specifies the input data table that contains information about the outcome model. The data table must be an output data table that the OUTO= option produces in a previous call of the DEEPCAUSAL procedure. When you specify this option, the MODEL statement and the INA=, INB=, OUTA=, OUTB=, and OUTO= options in the SCORE statement are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

INPS=*libref.data-table*

INPROPSITYSCORE=*libref.data-table*

specifies the input data table that contains information about the propensity score model. The data table must be an output data table that the OUTPS= option produces in a previous call of the DEEPCAUSAL procedure. When you specify this option, the PSMODEL statement and the OUTPS= option in the SCORE statement are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

OUTA=*libref.data-table*

OUTALPHA=*libref.data-table*

writes the $\alpha(\cdot)$ estimation information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

OUTB=*libref.data-table*

OUTBETA=*libref.data-table*

writes the $\beta(\cdot)$ estimation information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

OUTO=*libref.data-table*

OUTOUTCOME=*libref.data-table*

writes the outcome model information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

OUTPS=*libref.data-table*

OUTPROPENSITYSCORE=*libref.data-table*

writes the propensity score model information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 909.

Details: The DEEPCAUSAL Procedure

Causal Inference Overview

Causal inference is the science of why. Many classical statistics models, such as linear regression, show how the outcome variable is associated with the covariates. It is known that correlation does not equal causation. However, it is often not known whether a change in the outcome is caused by the change in a treatment. Randomized experiments provide a solution, but there are many constraints that often make the experiment impossible. For example, designing and then conducting the experiment is time-consuming and expensive. Moreover, an experiment might encounter some ethical issues that prevent the experiment from being conducted. Therefore, causal inference based on observational data is indispensable to answering the causal questions.

The DEEPCAUSAL procedure is designed primarily for observational data when the subjects are not randomly assigned to the treatment conditions. However, in some situations the randomized trials need to be treated as observational studies. For example, subjects can be randomly assigned to the treatment group, or block randomization can be used when there are several confounding variables. However, after the experiment, maybe a few more confounding variables are discovered. Or worse yet, maybe the treatment and control groups do not have similar distributions across the segments of one or more new confounding variables. In these cases, instead of doing the experiment again, you need to estimate the causal effect by treating the experiment as an observational study. Moreover, selection bias can affect the treatment effect estimation. For example, suppose that there is a new version of a smartphone app, and the relevant notifications are sent to randomly selected individuals, who can be seen as the treatment group. Then the effect of the new app version on a certain outcome, such as the average time spent on the app every day, is worth investigating. The issue is that not everyone in the treatment group will download the new version. In the treatment group, those who download the new version are different from those who do not, and then selection bias arises. To avoid that bias, the control group can be compared to all subjects in the treatment group no matter whether a particular individual downloads the app. The estimated treatment effect is no doubt diluted because some individuals in the treatment group are not actually treated. Again, one solution is to treat the experiment as an observational study.

PROC DEEPCAUSAL exploits deep neural networks (DNNs) for causal inference. It applies deep learning to causal inference in a two-step semiparametric framework: the nonparametric step and the parametric step. In the nonparametric step, two DNNs are used to estimate the unknown functions in the propensity score model and outcome model. In the parametric step, parameters of interest are estimated through the corresponding influence functions.

Estimating Unknown Functions in Propensity Score and Outcome Models

There are two main models in causal inference: the propensity score model and the outcome model. Two approximate DNNs are constructed to estimate these two models. The DEEPCAUSAL procedure supports one binary treatment and one continuous outcome. The unconfoundedness is assumed; it is also called ignorability, selection on the observables, or no omitted variable bias.

Propensity Score Model

The propensity score model estimates the treatment assigning probability conditional on some covariates:

$$p(\mathbf{x}^{(1)}) = \Pr(T = 1 | \mathbf{X}^{(1)} = \mathbf{x}^{(1)})$$

where $p(\mathbf{x}^{(1)})$ denotes the propensity score, T denotes the binary treatment, and $\mathbf{x}^{(1)}$ is a vector of the observed covariates. The superscript (1) in $\mathbf{X}^{(1)}$ denotes the covariates in Set 1, which are the covariates used in the propensity score model. The $p(\cdot)$ is one unknown function to be estimated by a DNN.

The DNN for the propensity score model, a binary classification predictor, predicts the binary treatment by using the covariates in Set 1. It trains the observations in the input data set. The DNN is a fully connected multilayer perceptron (MLP). The input layer of that DNN corresponds to $\mathbf{X}^{(1)}$, and the output layer corresponds to the observed treatment variable. The activation functions for the hidden layers are all rectified linear unit (ReLU) functions, and the activation function for the output layer is a sigmoid function. The loss function is the negative log-likelihood function.

If no covariates are used to predict the binary treatment ($\mathbf{X}^{(1)} = \emptyset$), then no DNN needs to be trained for the propensity score model. A random assignment of the treatment and many other reasons might result in the empty $\mathbf{X}^{(1)}$; that is, all covariates are independent of the treatment. The propensity score is estimated by

$$p(\emptyset) = \Pr(T = 1 | \mathbf{X}^{(1)} = \emptyset) = \Pr(T = 1) = \frac{\sum_{i=1}^n \mathbb{1}\{t_i = 1\}}{n}$$

where t_i denotes the treatment value of the i th observation and n denotes the number of observations. The indicator function $\mathbb{1}\{A\} = 1$ if A is true; otherwise, it is 0.

Outcome Model

The outcome model describes how the outcome depends on the treatment and some covariates in Set 2, which are the covariates used in the propensity score model. The covariates in Set 1 and Set 2 might be different. The DEEPCAUSAL procedure supports only one continuous outcome variable, and the outcome model is as follows:

$$\mathbb{E}[Y | \mathbf{X}^{(2)} = \mathbf{x}^{(2)}, T = t] = G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t)$$

where Y is the outcome, t is the treatment, and $G(\cdot)$ is a known function. For example, $G(u)$ might be the identity function; that is, $G(u) = u$. The superscript (2) in $\mathbf{X}^{(2)}$ denotes the covariates in Set 2. In PROC DEEPCAUSAL, $G(\cdot)$ is the identity function. The $\alpha(\cdot)$ and $\beta(\cdot)$ are unknown functions to be estimated by a DNN.

The DNN for the outcome model, a regression predictor, predicts the outcome by using the covariates in Set 2 and the treatment variable. It trains the observations in the input data set. The input layer of that DNN corresponds to $\mathbf{X}^{(2)}$. After the hidden layers, there is a parameter layer with two neurons that correspond to

$\alpha(\mathbf{x}^{(2)})$ and $\beta(\mathbf{x}^{(2)})$. The activation functions for this parameter layer are linear functions. Subsequently, t is included, and $G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t)$ is calculated as the result of the final output layer. $G(\cdot)$ is set to the identity function in PROC DEEPCAUSAL, so $\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t$ is output as the prediction.

If no covariates are used to predict the outcome ($\mathbf{X}^{(2)} = \emptyset$), then no DNN needs to be trained for the outcome model. The outcome model is predicted by

$$\mathbb{E}[Y|\mathbf{X}^{(2)} = \emptyset, T = t] = \mathbb{E}[Y|T = t]$$

Moreover, when $\mathbf{X}^{(2)} = \emptyset$, $\alpha(\mathbf{x}^{(2)})$ and $\beta(\mathbf{x}^{(2)})$ are constants and are computed by

$$\begin{aligned}\alpha(\emptyset) &= \mathbb{E}[Y|T = 0] = \frac{\sum_{i=1}^n y_i \mathbb{1}\{t_i = 0\}}{\sum_{i=1}^n \mathbb{1}\{t_i = 0\}} \\ \beta(\emptyset) &= \mathbb{E}[Y|T = 1] - \mathbb{E}[Y|T = 0] = \frac{\sum_{i=1}^n y_i \mathbb{1}\{t_i = 1\}}{\sum_{i=1}^n \mathbb{1}\{t_i = 1\}} - \frac{\sum_{i=1}^n y_i \mathbb{1}\{t_i = 0\}}{\sum_{i=1}^n \mathbb{1}\{t_i = 0\}}\end{aligned}$$

where y_i denotes the outcome value of the i th observation, t_i denotes the treatment value of the i th observation, and n denotes the number of observations. The indicator function $\mathbb{1}\{A\} = 1$ if A is true; otherwise, it is 0.

Estimating Parameters of Interest

A parameter of interest can be defined through the following structural model:

$$\theta_0 = \mathbb{E}[H(y, \mathbf{X}, T, \alpha(\mathbf{X}^{(2)}), \beta(\mathbf{X}^{(2)}); t^*)]$$

where θ_0 is a certain parameter of interest, $H(\cdot)$ is a known function that corresponds to θ_0 , $\mathbf{X}$ is the union of $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$, and t^* is a fixed treatment value of interest. To have the expected value, a naive approach is to plug the DNN-based estimates $\hat{\alpha}(\mathbf{x}^{(2)})$ and $\hat{\beta}(\mathbf{x}^{(2)})$ into the structural model of θ_0 . This estimator, which is known as the plug-in estimator, is given by

$$\hat{\theta}_0 = \frac{1}{n} \sum_{i=1}^n H(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}); t^*)$$

where n is the sample size and the subscript i stands for the i th observation. For example, if T is binary ($T \in \{0, 1\}$), then θ_0 is the estimator of the average treatment effect (ATE) when $H(y, \mathbf{X}, T, \alpha(\mathbf{X}^{(2)}), \beta(\mathbf{X}^{(2)}); t^*) = \beta(\mathbf{X}^{(2)})$. Then the plug-in estimator of ATE is given by

$$\hat{\theta}_0 = \frac{1}{n} \sum_{i=1}^n \hat{\beta}(\mathbf{x}_i^{(2)})$$

This estimator might be highly biased, because the covariates might have both effects on the outcome and treatment assignment. One way to solve this problem is to use an influence function–based estimator. With one treatment variable, Farrell, Liang, and Misra (2020) proposed the following influence function:

$$\begin{aligned}\psi(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) + \nabla \mathbf{H}(\mathbf{x}) \Lambda(\mathbf{x})^{-1} (1, t)' (y - G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t))\end{aligned}$$

where $\nabla \mathbf{H}(\mathbf{x}) = (\frac{\partial H}{\partial \alpha}, \frac{\partial H}{\partial \beta})$, $\Lambda(\mathbf{x}) = \mathbb{E}[\dot{G}(1, T)'(1, T)|\mathbf{X} = \mathbf{x}]$, and $\dot{G} = [dG(u)/du]_{u=\alpha(\mathbf{x}^{(2)})+\beta(\mathbf{x}^{(2)})t}$.

Because PROC DEEPCAUSAL sets $G(\cdot)$ to the identity function and supports one binary treatment, $\dot{G} = 1$ and the inverse of $\Lambda(\mathbf{x})$ can be computed directly. The preceding influence function can be simplified as follows:

$$\begin{aligned} \psi(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) \\ + \frac{\dot{H}_\alpha(\mathbf{x})(\lambda_2(\mathbf{x}) - \lambda_1(\mathbf{x})t) + \dot{H}_\beta(\mathbf{x})(\lambda_0(\mathbf{x})t - \lambda_1(\mathbf{x}))}{\lambda_2(\mathbf{x})\lambda_0(\mathbf{x}) - \lambda_1(\mathbf{x})^2} (Y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t) \end{aligned}$$

where $\lambda_j(\mathbf{x}) = \mathbb{E}[T^j | \mathbf{X} = \mathbf{x}]$, $j = 0, 1, 2$; $\dot{H}_\alpha(\mathbf{x}) = \frac{\partial H}{\partial \alpha}$; and $\dot{H}_\beta(\mathbf{x}) = \frac{\partial H}{\partial \beta}$. The binary treatment ($T \in \{0, 1\}$) is supported in PROC DEEPCAUSAL so that $\lambda_0(\mathbf{x}) = 1$ and $\lambda_1(\mathbf{x}) = \lambda_2(\mathbf{x}) = p(\mathbf{x})$.

The estimator for the parameter of interest is given by

$$\hat{\theta}_0 = \frac{\sum_{i=1}^n \psi(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

This estimator is doubly robust, which means that the estimator is unbiased when either the propensity score model or the outcome model is correct. The standard error of $\hat{\theta}_0$ is

$$\sigma(\hat{\theta}_0) = \frac{\sqrt{\sum_{i=1}^n (\psi(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*) - \hat{\theta}_0)^2}}{n}$$

For ease of presentation, the missing value is not indicated in the preceding equations. In practice, if $\psi(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$ is a missing value for $i = 1, \dots, n$, then it is ignored in both the denominator and numerator of the equation of $\hat{\theta}_0$ or $\sigma(\hat{\theta}_0)$. This policy is applied to all parameters of interest described in the sections “[Full-Population Average Effect Parameters](#)” on page 934, “[Subpopulation Average Effect Parameters](#)” on page 936, and “[Policy Evaluation and Comparison](#)” on page 937.

Next, the calculation of some common parameters of interest is introduced.

Full-Population Average Effect Parameters

There are three types of full-population average effect parameters:

- $\mathbb{E}[Y(0)] = \frac{1}{n} \sum_{i=1}^n Y_i(0)$, where Y_0 denotes the population-level potential outcome without the treatment, $Y_i(0)$ denotes the potential outcome for the i th observation without the treatment, and n denotes the number of observations
- $\mathbb{E}[Y(1)] = \frac{1}{n} \sum_{i=1}^n Y_i(1)$, where Y_0 denotes the population-level potential outcome with the treatment, $Y_i(1)$ denotes the potential outcome for the i th observation with the treatment, and n denotes the number of observations
- average treatment effect (ATE); $\text{ATE} = \mathbb{E}[Y(1) - Y(0)]$

The word “potential” is used in the preceding definitions because it is often impossible for an individual who is treated to return to the original status and be untreated, and vice versa.

The three full-population average effect parameters are estimated as follows:

- average with potential outcome 0, $\mu_0 = \mathbb{E}[Y(0)]$:

The influence function, denoted by $\psi_{\mu_0}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*)$, is defined by setting $H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = H_{\mu_0}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = \alpha(\mathbf{x}^{(2)})$ in the influence function. When $G(\cdot)$ is set to the identity function and the treatment is binary, the influence function is simplified as

$$\psi_{\mu_0}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) = \alpha(\mathbf{x}^{(2)}) + \frac{1-t}{1-p(\mathbf{x}^{(1)})}(y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t)$$

where $p(\mathbf{x}^{(1)}) = \Pr(T = 1 | \mathbf{X}^{(1)} = \mathbf{x}^{(1)})$. Then μ_0 is estimated by

$$\mu_0 = \frac{\sum_{i=1}^n \psi_{\mu_0}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

- average with potential outcome 1, $\mu_1 = \mathbb{E}[Y(1)]$:

The influence function, denoted by $\psi_{\mu_1}(Y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*)$, is defined by setting $H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = H_{\mu_1}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = \alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})$ in the influence function. When $G(\cdot)$ is set to the identity function and the treatment is binary, the influence function is simplified as

$$\psi_{\mu_1}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) = \alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)}) + \frac{t}{p(\mathbf{x}^{(1)})}(y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t)$$

where $p(\mathbf{x}^{(1)}) = \Pr(T = 1 | \mathbf{X}^{(1)} = \mathbf{x}^{(1)})$. Then μ_1 is estimated by

$$\mu_1 = \frac{\sum_{i=1}^n \psi_{\mu_1}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

- average treatment effect (ATE):

The influence function, denoted by $\psi_{\text{ATE}}(Y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*)$, is defined by setting $H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = H_{\text{ATE}}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = \beta(\mathbf{x}^{(2)})$ in the influence function. When $G(\cdot)$ is set to the identity function and the treatment is binary, the influence function is simplified as

$$\psi_{\text{ATE}}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) = \beta(\mathbf{x}^{(2)}) + \frac{t - p(\mathbf{x}^{(1)})}{p(\mathbf{x}^{(1)})(1 - p(\mathbf{x}^{(1)}))}(y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t)$$

where $p(\mathbf{x}^{(1)}) = \Pr(T = 1 | \mathbf{X}^{(1)} = \mathbf{x}^{(1)})$. Then ATE is estimated by

$$\text{ATE} = \frac{\sum_{i=1}^n \psi_{\text{ATE}}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

Subpopulation Average Effect Parameters

The subpopulation average effect parameters are based on the conditional average with potential outcome t_1 and treatment level t_2 , ρ_{t_1, t_2} , which is defined by

$$\rho_{t_1, t_2} = \mathbb{E}[Y(t_1)|T = t_2], \quad t_1, t_2 \in \{0, 1\}$$

The subpopulation average effect parameters are estimated as follows:

- ρ_{t_1, t_2} :

The influence function is calculated by

$$\begin{aligned} & \psi_{t_1, t_2}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ &= \frac{\Pr(t = t_2 | \mathbf{X}^{(1)} = \mathbf{x}^{(1)})}{\Pr(t = t_2)} \frac{\mathbb{1}\{t = t_1\}(Y - \mathbb{E}[Y | \mathbf{X}^{(2)} = \mathbf{x}^{(2)}, t = t_1])}{\Pr(t = t_1 | \mathbf{X}^{(1)} = \mathbf{x}^{(1)})} \\ &+ \frac{\mathbb{1}\{t = t_2\}\mathbb{E}[Y | \mathbf{X}^{(2)} = \mathbf{x}^{(2)}, t = t_1]}{\Pr(t = t_2)} \end{aligned}$$

where $\Pr(t = t_i | \mathbf{X}^{(1)} = \mathbf{x}^{(1)})$ is acquired from the DNN for the propensity model and $\mathbb{E}[Y | \mathbf{X}^{(2)} = \mathbf{x}^{(2)}, t = t_1]$ is acquired from the DNN for the outcome model. For more information, see Farrell, Liang, and Misra (2021). Then ρ_{t_1, t_2} is estimated by

$$\rho_{t_1, t_2} = \frac{\sum_{i=1}^n \psi_{t_1, t_2}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

- average treatment effect for the treated (ATT), $\text{ATT} = \rho_{1,1} - \rho_{0,1}$:

For ATT, the influence function is replaced by

$$\begin{aligned} & \psi_{\text{ATT}}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ &= \psi_{1,1}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) - \psi_{0,1}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \end{aligned}$$

Then ATT is estimated by

$$\text{ATT} = \frac{\sum_{i=1}^n \psi_{\text{ATT}}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

- average treatment effect for the untreated (ATU), $\text{ATU} = \rho_{1,0} - \rho_{0,0}$:

For ATU, the influence function is replaced by

$$\begin{aligned} & \psi_{\text{ATU}}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ &= \psi_{1,0}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) - \psi_{0,0}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \end{aligned}$$

Then ATU is estimated by

$$\text{ATU} = \frac{\sum_{i=1}^n \psi_{\text{ATU}}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

- composition effect, $\Delta_X = \rho_{1,1} - \rho_{1,0}$:

For Δ_X , the influence function is replaced by

$$\begin{aligned} \psi_{\Delta_X}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = \psi_{1,1}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) - \psi_{1,0}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \end{aligned}$$

Then Δ_X is estimated by

$$\Delta_X = \frac{\sum_{i=1}^n \psi_{\Delta_X}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

- observed difference, $\Delta = \rho_{1,1} - \rho_{0,0}$:

For Δ , the influence function is replaced by

$$\begin{aligned} \psi_{\Delta}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = \psi_{1,1}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) - \psi_{0,0}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \end{aligned}$$

Then Δ is estimated by

$$\Delta = \frac{\sum_{i=1}^n \psi_{\Delta}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

Policy Evaluation and Comparison

First, a policy is defined to be a rule that assigns a given set of information, represented by covariates $\mathbf{X}$, to treatment status; that is, a policy is a known map function that is defined by

$$s(\mathbf{X}) : \mathbf{X} \rightarrow \{0, 1\}$$

The expected utility of a policy, denoted by $\pi(s)$, is given by

$$\pi(s) = \mathbb{E}[s(\mathbf{X})Y(1) + (1 - s(\mathbf{X}))Y(0)]$$

To estimate $\pi(s)$, the influence function is replaced by

$$\begin{aligned} \psi_{\pi(s)}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = s(\mathbf{x})\psi_{\mu_1}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) + (1 - s(\mathbf{x}))\psi_{\mu_0}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \end{aligned}$$

where $\psi_{\mu_j}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*)$ denotes the influence function for μ_j ($j = 0$ or 1). Then $\pi(s)$ is estimated by

$$\pi(s) = \frac{\sum_{i=1}^n \psi_{\pi(s)}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

The difference in expected utility of two policies is defined by

$$\pi(s_1, s_0) = \mathbb{E}[(s_1(\mathbf{X}) - s_0(\mathbf{X}))(Y(1) - Y(0))]$$

To estimate $\pi(s_1, s_0)$, the influence function is replaced by

$$\begin{aligned} \psi_{\pi(s_1, s_0)}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = (s_1(\mathbf{x}) - s_0(\mathbf{x}))\psi_{\text{ATE}}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \end{aligned}$$

where $\psi_{\text{ATE}}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*)$ denotes the influence function for estimating ATE. Then $\pi(s_1, s_0)$ is estimated by

$$\pi(s_1, s_0) = \frac{\sum_{i=1}^n \psi_{\pi(s_1, s_0)}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

Missing Values

Treatment Variable

The input treatment variable cannot have missing values. If the input treatment variable has missing values, PROC DEEPCAUSAL terminates with an error.

Outcome Variable

In the DNN training stage of the outcome model, the missing outcome values are replaced by imputed values. For more information, see the **TARGETMISSING=** option. In the inference stage, the observations that have missing outcome values have missing values for residuals and any estimates that need residuals.

Covariate

In the DNN training stage of the propensity score model or the outcome model, the missing covariate values are replaced by imputed values. For more information, see the **MISSING=** option. In the inference stage, the observations that have missing covariates are ignored. However, the probability mass function (PMF) of the treatment variable ($\Pr(t = 0)$ or $\Pr(t = 1)$) considers all observations. As mentioned in the section “Treatment Variable” on page 938, each observation has a nonmissing treatment variable, and that value is used for calculating the PMF of the treatment variable.

Parameter of Interest

For any parameter of interest, $\psi(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$ is assigned a missing value if any component of it is missing, except for the following two subpopulation parameters of interest:

- $\rho_{0,0}$:
If $t_i = 1$, then $\psi_{0,0}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*) = 0$ always holds no matter whether any component of it is missing for $i = 1, \dots, n$, where n is the number of observations in the input data table.
- $\rho_{1,1}$:
If $t_i = 0$, then $\psi_{1,1}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*) = 0$ always holds no matter whether any component of it is missing for $i = 1, \dots, n$, where n is the number of observations in the input data table.

Data Table Output

The DEEPCASUAL procedure can create the data tables that are specified in the OUT= and OUTDETAILS= options in the INFER statement. The information in each table is described in the following sections.

OUT= Data Table Generated from the INFER Statement

The estimate and the corresponding standard error of each parameter of interest are displayed in this table. For more information about the parameters of interest, see the sections “[Full-Population Average Effect Parameters](#)” on page 934 and “[Subpopulation Average Effect Parameters](#)” on page 936. The output data table contains the following variables:

NOBS	number of observations in the input data table
_N_MU0_	number of nonmissing μ_0
_EST_MU0_	estimate of μ_0
_STD_MU0_	standard error of μ_0
_N_MU1_	number of nonmissing μ_1
_EST_MU1_	estimate of μ_1
_STD_MU1_	standard error of μ_1
_N_ATE_	number of nonmissing ATEs
_EST_ATE_	estimate of ATE
_STD_ATE_	standard error of ATE
_N_RHO00_	number of nonmissing $\rho_{0,0}$
_EST_RHO00_	estimate of $\rho_{0,0}$
_STD_RHO00_	standard error of $\rho_{0,0}$
_N_RHO01_	number of nonmissing $\rho_{0,1}$
_EST_RHO01_	estimate of $\rho_{0,1}$
_STD_RHO01_	standard error of $\rho_{0,1}$
_N_RHO10_	number of nonmissing $\rho_{1,0}$
_EST_RHO10_	estimate of $\rho_{1,0}$
_STD_RHO10_	standard error of $\rho_{1,0}$
_N_RHO11_	number of nonmissing $\rho_{1,1}$
_EST_RHO11_	estimate of $\rho_{1,1}$
_STD_RHO11_	standard error of $\rho_{1,1}$
_N_ATT_	number of nonmissing ATTs
_EST_ATT_	estimate of ATT
_STD_ATT_	standard error of ATT
_N_DELTAX_	number of nonmissing Δ_X

<code>_EST_DELTAX_</code>	estimate of Δ_X
<code>_STD_DELTAX_</code>	standard error of Δ_X
<code>_N_ATU_</code>	number of nonmissing ATUs
<code>_EST_ATU_</code>	estimate of ATU
<code>_STD_ATU_</code>	standard error of ATU
<code>_N_DELTA_</code>	number of nonmissing Δ . NOTE: In releases through 2023.03, this variable is named <code>_N_DELTA_</code> . Because of a conflict with some reserved SAS system variables, it has been renamed <code>_N_DELTA__</code> (with two final underscores) as of release 2023.04.
<code>_EST_DELTA_</code>	estimate of Δ . NOTE: In releases through 2023.03, this variable is named <code>_EST_DELTA_</code> . Because of a conflict with some reserved SAS system variables, it has been renamed <code>_EST_DELTA__</code> (with two final underscores) as of release 2023.04.
<code>_STD_DELTA_</code>	standard error of Δ . NOTE: In releases through 2023.03, this variable is named <code>_STD_DELTA_</code> . Because of a conflict with some reserved SAS system variables, it has been renamed <code>_STD_DELTA__</code> (with two final underscores) as of release 2023.04.
<code>_N_POLICY_POLICY-VAR_</code>	number of nonmissing $\pi(s)$, where s is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>POLICY=(policy-variable-list)</code> option in the <code>INFER</code> statement. The estimate is displayed for each policy variable.
<code>_EST_POLICY_POLICY-VAR_</code>	estimate of $\pi(s)$, where s is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>POLICY=(policy-variable-list)</code> option in the <code>INFER</code> statement. The estimate is displayed for each policy variable.
<code>_STD_POLICY_POLICY-VAR_</code>	standard error of $\pi(s)$, where s is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>POLICY=(policy-variable-list)</code> option in the <code>INFER</code> statement. The standard error is displayed for each policy variable.
<code>_N_POLICY_COMPARISON_POLICY-VAR_BASE-VAR_</code>	number of nonmissing $\pi(s_c, s_b)$, where s_c is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>POLICYCOMPARISON(COMPARE=(compare-list))</code> option in the <code>INFER</code> statement and s_b is the value of a policy variable, <code>BASE-VAR</code> , that you specify in the <code>POLICYCOMPARISON(BASE=(base-list))</code> option in the <code>INFER</code> statement. The estimate is displayed for each combination of s_c and s_b .
<code>_EST_POLICY_COMPARISON_POLICY-VAR_BASE-VAR_</code>	estimate of $\pi(s_c, s_b)$, where s_c is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>POLICYCOMPARISON(COMPARE=(compare-list))</code> option in the <code>INFER</code> statement and s_b is the value of a policy variable, <code>BASE-VAR</code> , that you specify in the <code>POLICYCOMPARISON(BASE=(base-list))</code> option in the <code>INFER</code> statement. The estimate is displayed for each combination of s_c and s_b .
<code>_STD_POLICY_COMPARISON_POLICY-VAR_BASE-VAR_</code>	standard error of $\pi(s_c, s_b)$, where s_c is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>POLICYCOMPARISON(COMPARE=(compare-list))</code> option in the <code>INFER</code> statement and s_b is the value of a policy variable, <code>BASE-VAR</code> , that you specify in the <code>POLICYCOMPARISON(BASE=(base-list))</code> option in the <code>INFER</code> statement. The standard error is displayed for each combination of s_c and s_b .

OUTDETAILS= Data Table Generated from the INFER Statement

The output data first consists of the variables in the input data table and the details in the estimation, including $\alpha(\cdot)$, $\beta(\cdot)$, $p(\cdot)$, the residual, and the influence functions for each observation in the input data table. For more information about the influence functions for parameters of interest, the propensity score prediction model, and the outcome prediction model, see the sections “[Estimating Unknown Functions in Propensity Score and Outcome Models](#)” on page 932 and “[Estimating Parameters of Interest](#)” on page 933. In what follows, i denotes the observation index. Moreover, the output data table contains the following variables in addition to the input variables:

<code>_PSHAT_</code>	logarithm of the odds $p(\mathbf{x}_i^{(1)})/(1 - p(\mathbf{x}_i^{(1)}))$ for each observation, where $p(\mathbf{x}_i^{(1)}) = \Pr(T = 1 \mathbf{X}^{(1)} = \mathbf{x}_i^{(1)})$
<code>_YHAT_</code>	value of $\mathbb{E}[Y \mathbf{X}^{(2)} = \mathbf{x}_i^{(2)}, T = t_i] = \hat{\alpha}(\mathbf{x}_i^{(2)}) + \hat{\beta}(\mathbf{x}_i^{(2)})t_i$ for each observation
<code>_Y0_</code>	value of $\mathbb{E}[Y \mathbf{X}^{(2)} = \mathbf{x}_i^{(2)}, T = 0] = \hat{\alpha}(\mathbf{x}_i^{(2)})$ for each observation
<code>_Y1_</code>	value of $\mathbb{E}[Y \mathbf{X}^{(2)} = \mathbf{x}_i^{(2)}, T = 1] = \hat{\alpha}(\mathbf{x}_i^{(2)}) + \hat{\beta}(\mathbf{x}_i^{(2)})$ for each observation
<code>_AVGPX_</code>	value of $\Pr(T = 1) = \frac{\sum_{i=1}^N \mathbb{1}\{t_i=1\}}{n}$, where n the total number of observations. This is the unconditional probability that a treatment equals 1. The value is identical for each observation.
<code>_px_</code>	value of $p(\mathbf{x}_i^{(1)}) = \Pr(T = 1 \mathbf{X}^{(1)} = \mathbf{x}_i^{(1)})$ for each observation
<code>_alpha_</code>	value of $\hat{\alpha}(\mathbf{x}_i^{(2)})$ for each observation
<code>_beta_</code>	value of $\hat{\beta}(\mathbf{x}_i^{(2)})$ for each observation
<code>_eps_</code>	value of $y_i - \mathbb{E}[Y \mathbf{X}^{(2)} = \mathbf{x}_i^{(2)}, T = t_i] = y_i - \hat{\alpha}(\mathbf{x}_i^{(2)}) - \hat{\beta}(\mathbf{x}_i^{(2)})t_i$ for each observation
<code>_mu0_</code>	value of $\psi_{\mu_0}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_mu1_</code>	value of $\psi_{\mu_1}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_ATE_</code>	value of $\psi_{\text{ATE}}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_rho00_</code>	value of $\psi_{0,0}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_rho01_</code>	value of $\psi_{0,1}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_rho10_</code>	value of $\psi_{1,0}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_rho11_</code>	value of $\psi_{1,1}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_ATT_</code>	value of $\psi_{\text{ATT}}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_deltaX_</code>	value of $\psi_{\Delta_X}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_ATU_</code>	value of $\psi_{\text{ATU}}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$
<code>_delta__</code>	value of $\psi_{\Delta}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$. NOTE: In releases through 2023.03, this variable is named <code>_delta_</code> . Because of a conflict with some reserved SAS system variables, it has been renamed <code>_delta__</code> (with two final underscores) as of release 2023.04.

- `_policy_policy-var_` value of $\psi_{\pi(s)}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$, where s is the value of a policy variable that you specify in the `POLICY=(policy-variable-list)` option in the `INFER` statement. The estimate is displayed for each policy variable.
- `_policy_comparison_policy-var_base-var_` value of $\psi_{\pi(s_c, s_b)}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$, where s_c is the value of a policy variable, `POLICY-VAR`, that you specify in the `POLICYCOMPARISON(COMPARE=(compare-list))` option in the `INFER` statement and s_b is the value of a policy variable, `BASE-VAR`, that you specify in the `POLICYCOMPARISON(BASE=(base-list))` option in the `INFER` statement. The estimate is displayed for each combination of s_c and s_b .

ODS Table Names

The DEEPCAUSAL procedure assigns a name to each table that it creates. You can use these names to refer to the tables when you use the Output Delivery System (ODS) to select tables and create output data tables.

These names are listed in [Table 15.2](#).

Table 15.2 ODS Tables Produced in the DEEPCAUSAL Procedure

ODS Table Name	Description	Option
NObs	Number of observations	Default
ParameterEstimates	Full-population parameter estimates	Default
PolicyEvaluation	Policy evaluation	POLICY=
PolicyComparison	Policy comparison	POLICYCOMPARISON=
SubPopulationEstimates	Subpopulation parameter estimates	Default

Examples: DEEPCAUSAL Procedure

Example 15.1: Treatment Effect of Unemployment Insurance Bonus on Unemployment Duration

This example investigates the incentive effect of unemployment insurance (UI) by analyzing the Pennsylvania Reemployment Bonus experiment that was conducted by the US Department of Labor from July 1988 to October 1989. In the experiment, each UI claimant was randomly assigned to the control group or to one of the bonus treatment groups. The UI claimants in the treatment groups were given a corresponding bonus if they found a job within the specified qualification period. The amount of the bonus and the length of the qualification period varied across different treatment groups; the fourth treatment group had the most generous bonus offer—the highest bonus amount for the longest qualification period. For the UI claimants in the control group, no bonus was offered.

The report of the experiment with raw data is provided by W. E. Upjohn Institute for Employment Research at <https://www.upjohn.org/data-tools/employment-research-data-center/pennsylvania-reemployment-bonus-demonstration>. The public-use data consist of two files. This example examines the records file that contains administrative data for individuals from the participant tracking system and data for selected ineligible claimants.

The following statements load the data set of the records file into the data set `mydata` and then delete the duplicated rows:

```
/*---load .dta data set---*/
proc import out=mydata file="/dept/app/doc/viya/en/src/casecon/deepcausal/Recsfile.dta"
  dbms=dta replace;
run;
/*---delete duplicated rows---*/
proc sort data = mydata out = mydata nodupkey;
  by _all_;
run;
```

The variables for duration of unemployment (`inuidur1`), treatment group code (`tg`), age group dummies (`agelt35` and `agegt54`), gender dummy (`female`), race dummies (`black`, `hispanic`, and `othrace`), number of dependents (`dep`), quarters of the experiment (`q1–q6`), major industry code (`sic`), and locations (`site1–site12`) are selected from the data set `mydata`. The other variables are ignored. The following statements select the corresponding variables and delete rows that have missing values:

```
/*---select variables---*/
data mydata;
  set mydata (keep = tg inuidur1 female black hispanic
    othrace dep q1-q6 agelt35 agegt54 site1-site12 sic);
run;

/*---delete rows with missing values---*/
data mydata;
  set mydata;
  if cmiss(of _all_) then delete;
run;
```

This example splits the occupations of the claimants into two categories: durable manufacturing (`durable`) and other occupations. The durable manufacturing dummy variable is created using the major industry code (see <https://www.osha.gov/data/sic-manual> and <https://www2.deloitte.com/us/en/insights/economy/spotlight/economics-insights-analysis-04-2019.html>). The location variables are split into three categories: areas that had a low unemployment rate and a short duration of unemployment (`lusrd`), areas that had a high unemployment rate and a short duration of unemployment (`husrd`), and other areas. Moreover, only the most generous treatment group 4 (`tg=4`) is incorporated. Treatment group 6 (`tg=6`) and treatment group 4 had only one difference: treatment group 4 was offered the job-search workshop, and treatment group 6 was not. Although claimants in treatment group 4 were offered the workshop, only a few of them attended it eventually. Thus, treatment groups 4 and 6 are merged into one treatment group. The following statements complete the previously mentioned data set modification and load the data into the data set `selectData`:

```
data selectData;
  set mydata;
  /*---durable occupation dummy---*/
  if sic in (24 14 33 34 35 36 37 38 39) then durable = 1;
```

```

else durable = 0;
/*---lud dummy---*/
if site1 = 1 or site4 = 1 or site12 = 1 then lud = 1;
else lud = 0;
/*---husd dummy---*/
if site5 = 1 or site10 = 1 or site11 = 1 then husd = 1;
else husd = 0;
if tg=0 or tg=4 or tg=6 then output; /*group 4 and 6 will be merged later*/
run;

```

The treatment group (tg=4 or 6) is then assigned a value of 1, and the control group is treatment group 0 (tg=0). Finally, the binary treatment variable tg is ready for PROC DEEPCAUSAL. The outcome variable, logUldur, which represents the log of duration of unemployment, is computed, and the ID variable is also created. The following statements complete those modifications of selectData:

```

data selectData;
  set selectData;
  logUldur = log(inuidur1); * outcome variable;
  if tg=4 or tg=6 then tg=1; * merge group 4 and 6, binary treatment variable;
  i = _N_; * row id, PROC DEEPCAUSAL requires an ID variable;
run;

```

It is worth investigating whether the subjects are randomly assigned to the treatment group across the segments of each category covariate. The following statements generate the category variable quarter, which denotes the quarter index:

```

data selectData;
  set selectData;
  if q1 = 1 then quarter = 1;
  if q2 = 1 then quarter = 2;
  if q3 = 1 then quarter = 3;
  if q4 = 1 then quarter = 4;
  if q5 = 1 then quarter = 5;
  if q6 = 1 then quarter = 6;
run;

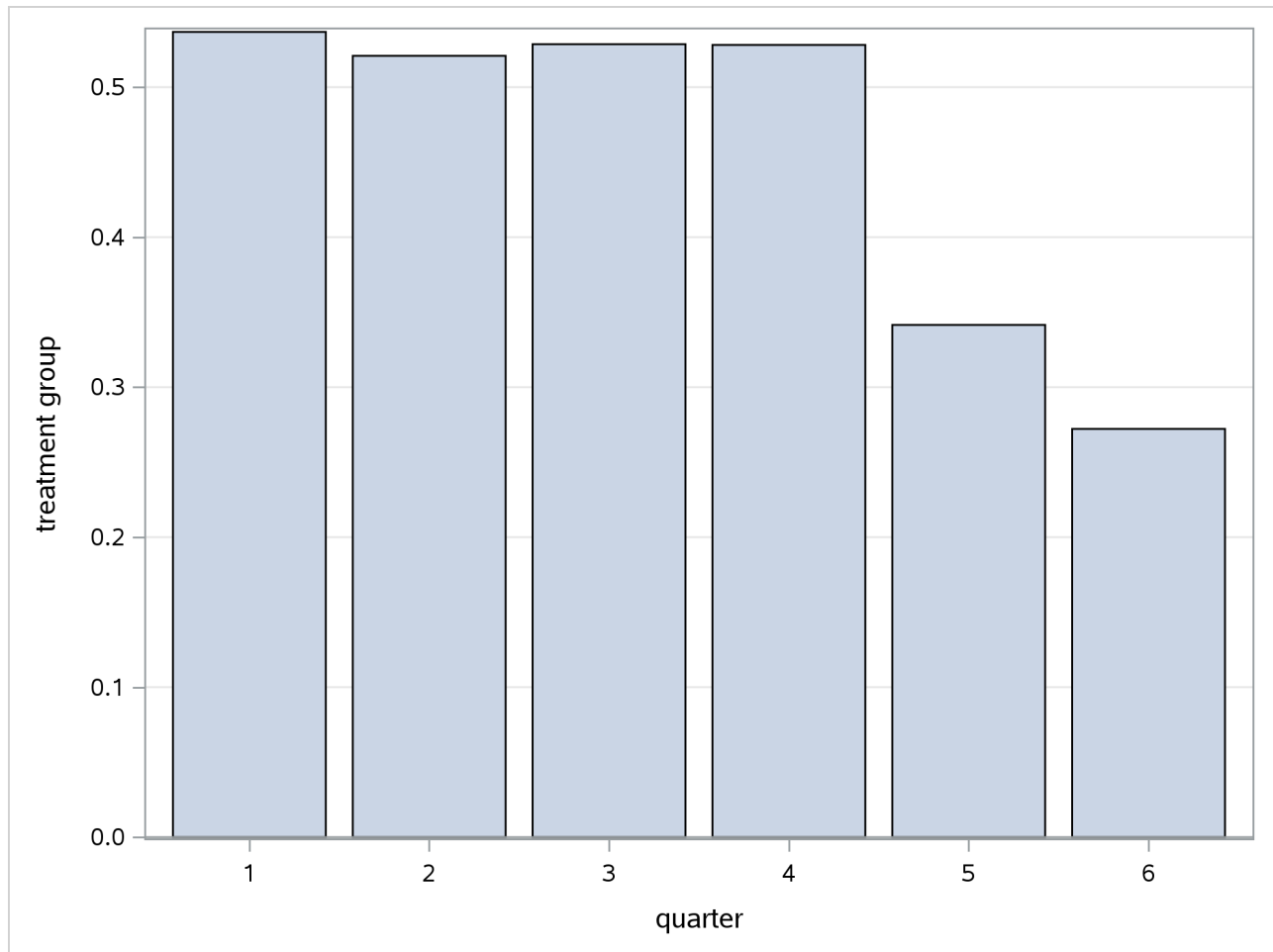
```

The following statements plot the mean of tg by quarter, which is shown in [Output 15.1.1](#). The mean of tg can be seen as the proportion of the treatment group, because tg is a binary treatment (tg=0 or 1).

```

proc sgplot data=selectData;
  vbar quarter / response=tg stat=mean;
  yaxis grid;
run;

```

Output 15.1.1 Proportion of Treatment by Quarter

Output 15.1.1 shows that the proportions of the treatment group of the fifth and sixth quarters are obviously different from the proportions of the other four quarters; that is, the proportion of the fifth or sixth quarter in the treatment group is different from proportions of the other four quarters. Thus, the subjects of the experiment are not randomly assigned to the treatment group across the six quarters.

To verify this conclusion, the following statements use PROC FREQ with the CHISQ option to test whether there is an association between the two category variables, tg and quarter:

```
proc freq data=selectData;
  tables tg*quarter / chisq cellchi2;
run;
```

The results are listed in Output 15.1.2, which shows that tg is significantly associated with quarter.

Output 15.1.2 Frequency Table and Chi-Square Test Results**The FREQ Procedure**

Frequency Cell Chi-Square Percent Row Pct Col Pct	Table of tg by quarter							
	tg(treatment group)	quarter						Total
		1	2	3	4	5	6	
	0	44	722	822	779	939	246	3552
		0.6998	6.1408	9.6928	9.0052	48.099	26.367	
		0.65	10.68	12.16	11.52	13.89	3.64	52.54
		1.24	20.33	23.14	21.93	26.44	6.93	
		46.32	47.91	47.13	47.18	65.85	72.78	
	1	51	785	922	872	487	92	3209
		0.7746	6.7971	10.729	9.9677	53.241	29.186	
		0.75	11.61	13.64	12.90	7.20	1.36	47.46
		1.59	24.46	28.73	27.17	15.18	2.87	
		53.68	52.09	52.87	52.82	34.15	27.22	
	Total	95	1507	1744	1651	1426	338	6761
		1.41	22.29	25.80	24.42	21.09	5.00	100.00

Statistic	DF	Value	Prob
Chi-Square	5	210.6996	<.0001
Likelihood Ratio Chi-Square	5	215.3130	<.0001
Mantel-Haenszel Chi-Square	1	132.3424	<.0001
Phi Coefficient		0.1765	
Contingency Coefficient		0.1738	
Cramer's V		0.1765	

In the same way, you can test whether the subjects of the experiment are randomly assigned to the treatment group across segments of other category variables, such as female.

Before you investigate the causal effect of tg by using PROC DEEPCAUSAL, the following statements calculate the descriptive statistics of the outcome variables for the control and treatment groups, respectively:

```
proc means data=selectData;
  class tg;
  var logUIdur;
run;
```

The results are shown in [Output 15.1.3](#).

Output 15.1.3 Summary Statistics for the Control and Treatment Groups**The MEANS Procedure**

Analysis Variable : logUIdur						
treatment group	N Obs	N	Mean	Std Dev	Minimum	Maximum
0	3552	3552	1.9660014	1.2567128	0	3.9512437
1	3209	3209	1.8938635	1.2391991	0	3.9512437

The statistical inference can be used to test whether the association, $\mathbb{E}[\log UIdur|tg = 1] - \mathbb{E}[\log UIdur|tg = 0]$, is significantly less than 0. However, directly considering the association statistical inference result to be ATE

is biased because the experiment is not fully randomized. Thus, the experiment is seen as an observational study.

The following DATA step loads the data set `selectData` into a data table in your session that is associated with the `mylib` libref. The DATA step assumes that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
data mylib.selectData;
    set selectData;
run;
```

The following statements use PROC DEEPCAUSAL to estimate ATE and other parameters of interest of the treatment variable, `tg`:

```
proc deepcausal data=mylib.selectData;
    id i;
    psmodel tg=female black othrace dep q2 q3 q4 q5 q6 age1t35
        age1t54 durable lused husd / dnn=(nodes=(32 32 32 32)
        train=(optimizer=(miniBatchSize=125
            regL1=0.35 regL2=0.004 maxEpochs=1000
            algorithm=adam(learningRate=0.025))
            nthreads=5 seed=12345));
    model logUIdur=female black othrace dep q2 q3 q4 q5 q6 age1t35
        age1t54 durable lused husd / dnn=(nodes=(32 32 32 32)
        train=(optimizer=(miniBatchSize=120
            regL1=0.0035 regL2=0.0035 maxEpochs=1000
            algorithm=adam(learningRate=0.0005))
            nthreads=5 seed=12345));
    infer out=mylib.oest outdetails=mylib.odetails;
run;
```

The number of observations is shown in [Output 15.1.4](#).

Output 15.1.4 Number of Observations

The DEEPCAUSAL Procedure

Number of Observations	6761
Observations with Missing Covariates	0
Observations with Missing Covariates for PS Model	0
Observations with Missing Covariates for Outcome Model	0
Observations with Missing Outcome	0

The estimates of parameters of interest are shown in [Output 15.1.5](#), which shows that ATE on log unemployment duration is significantly negative at the 5% level.

Output 15.1.5 Parameters of Interest

Population Parameter Estimates								
Parameter	Potential Outcome	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z	
Average	0	1.971554	0.020862	1.930666	2.012442	94.51	<.0001	
Average	1	1.887603	0.021720	1.845032	1.930174	86.90	<.0001	
ATE		-0.083951	0.029844	-0.142445	-0.025458	-2.81	0.0049	

Subpopulation Parameter Estimates								
Parameter	Potential Outcome	Treatment Level	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
Conditional Average	0	0	1.966001	0.031002	1.905238	2.026764	63.42	<.0001
Conditional Average	0	1	1.977701	0.032628	1.913751	2.041651	60.61	<.0001
Conditional Average	1	0	1.881947	0.030844	1.821493	1.942401	61.01	<.0001
Conditional Average	1	1	1.893863	0.032646	1.829879	1.957848	58.01	<.0001
ATT		1	-0.083837	0.029756	-0.142157	-0.025517	-2.82	0.0048
ATU		0	-0.084055	0.029949	-0.142753	-0.025356	-2.81	0.0050
Composition Effect	1		0.011917	0.046271	-0.078774	0.102607	0.26	0.7968
Observed Difference			-0.072138	0.055933	-0.181764	0.037488	-1.29	0.1971

For more information about the estimates of parameters of interest, policy evaluation, and policy comparison, see the sections “Full-Population Average Effect Parameters” on page 934 and “Subpopulation Average Effect Parameters” on page 936.

Example 15.2: Personalized Discount Policy for an Online Media Company

This example illustrates how a music subscription service from an online media company can offer targeted discounts through a personalized pricing plan based on many features that it observes about its customers to encourage them to buy more songs or become members. The main goal is to construct a policy that raises demand enough to boost overall revenue despite decreasing the price for some customers.

The data set is provided by the Microsoft research project ALICE and is available at https://msalicedatapublic.z5.web.core.windows.net/datasets/Pricing/pricing_sample.csv. The data set has 10,000 simulated observations that represent customers’ personal characteristics, such as age and log income, and online behavior history, such as previous purchase and previous online time per week. The treatment variable, t , is a binary variable that indicates whether or not a discount is applied. This variable is generated according to the values that the variable price takes in the data set. The value of t is 0 if the value of price is 1, indicating that no discount is given, and the value of t is 1 if the value of price is less than 1, indicating that a discount is applied. The outcome variable, revenue, is calculated by multiplying the number of songs purchased during the discount season by the price paid for the songs. Table 15.3 shows the names of the variables that are used in the model, their types, and their definitions. The type can be T (treatment), Y (outcome), $x^{(1)}$ (a covariate in the propensity score model), and/or $x^{(2)}$ (a covariate in the outcome model).

Table 15.3 Model Variables

Name	Type	Details
account_age	$x^{(1)}, x^{(2)}$	User's account age
age	$x^{(1)}, x^{(2)}$	User's age
avg_hours	$x^{(1)}, x^{(2)}$	Average number of hours user was online per week in the past
days_visited	$x^{(1)}, x^{(2)}$	Average number of days user visited website per week in the past
friend_count	$x^{(1)}, x^{(2)}$	Number of friends user connected to in account
has_membership	$x^{(1)}, x^{(2)}$	Whether user has membership
is_US	$x^{(1)}, x^{(2)}$	Whether user accesses website from US
songs_purchased	$x^{(1)}, x^{(2)}$	Average number of songs user purchased per week in the past
income	$x^{(1)}, x^{(2)}$	User's income
t	T	Whether a discount is applied
revenue	Y	Number of songs purchased during discount season times price paid

Assuming that you have downloaded the data set `pricing_sample` in your session that is associated with the `mylib` libref, the following statements create an ID variable that has a unique value for each observation, the treatment variable (`t`), and the outcome variable (`revenue`) in the data table `new_pricing_sample`:

```
data mylib.new_pricing_sample;
  set mylib.pricing_sample;
  id = put(_threadid_, 8.) || '_' || Put(_n_, 8.); * ID variable;
  if price<1 then t=1; else t=0; * treatment variable;
  revenue=price*demand; * outcome variable;
run;
```

The first step in policy evaluation and policy optimization is to estimate the effect of the treatment and to save to a specified output data table the details of the estimation, including $\alpha(\cdot)$, $\beta(\cdot)$, $p(\cdot)$, the residual, and the influence functions for each unit. You can do this by using the following statements:

```
/*--- Estimate the treatment effect and save the estimation details ---*/
proc deepcausal data=mylib.new_pricing_sample;
  id id;
  psmodel t = account_age age avg_hours days_visited friends_count
    has_membership is_US songs_purchased income /
    dnn=(nodes=(32 32 32 32))
    train=(optimizer=(miniBatchSize=500 regL1=0.0001 maxEpochs=32000
    algorithm=adam) nthreads=20 seed=12345 recordseed=67890));
  model revenue = account_age age avg_hours days_visited friends_count
    has_membership is_US songs_purchased income /
    dnn=(nodes=(32 32 32 32))
    train=(optimizer=(miniBatchSize=500 regL1=0.001 maxEpochs=32000
    algorithm=adam) nthreads=20 seed=12345 recordseed=67890));
  infer out=mylib.oest outdetails=mylib.odetails;
run;
```

For this example, the same covariates are used in both the propensity score model and the outcome model. The model estimation details are saved in the output data table `odetails`.

The estimation results are shown in [Output 15.2.1](#). The estimate of the average treatment effect, ATE, is negative and statistically significant, suggesting that the discount, on average, causes revenue that is generated

by the whole population to decrease. However, the effect of the discount on the revenue among the customers who received a discount, which is measured by the parameter ATT (average treatment effect on the treated), is considerably different from the ATE estimate. It might suggest that identifying the characteristics of customers to whom the discount matters would help construct the optimum policy so that it uses the fewest resources and earns the most profit.

Output 15.2.1 Parameters of Interest

The DEEPCAUSAL Procedure

Population Parameter Estimates							
Parameter	Potential Outcome	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
Average	0	14.323260	0.075167	14.175936	14.470584	190.55	<.0001
Average	1	13.577143	0.057725	13.464004	13.690282	235.20	<.0001
ATE		-0.746117	0.024569	-0.794272	-0.697962	-30.37	<.0001

Subpopulation Parameter Estimates							
Parameter	Potential Outcome	Treatment Level	Estimate	Standard Error	95% Confidence Limits		Z Pr > Z
Conditional Average	0	0	18.353066	0.230375	17.901539	18.804594	79.67 <.0001
Conditional Average	0	1	11.225712	0.135104	10.960913	11.490510	83.09 <.0001
Conditional Average	1	0	16.576263	0.204324	16.175796	16.976730	81.13 <.0001
Conditional Average	1	1	11.271842	0.120335	11.035990	11.507693	93.67 <.0001
ATT		1	0.046130	0.027921	-0.008594	0.100854	1.65 0.0985
ATU		0	-1.776803	0.036937	-1.849198	-1.704409	-48.10 <.0001
Composition Effect	1		-5.304422	0.305935	-5.904044	-4.704799	-17.34 <.0001
Observed Difference			-7.081225	0.330048	-7.728108	-6.434342	-21.46 <.0001

In policy optimization, the goal is to maximize the expected utility of a policy. For the definition of the expected utility function and how it is estimated, as well as details such as the definition of a policy rule, see the section “[Policy Evaluation and Comparison](#)” on page 937. In this example, because a positive treatment effect is preferred, anyone whose $\beta(x_i)$ value is positive should be given the treatment (that is, a discount). This function is sometimes referred to as the individual treatment effect (ITE).

The variable `_beta_` in the data table `odetails` that is obtained during the previous estimation contains the estimated values for this function for each individual. You can construct an optimal policy by offering a discount to individuals whose `_beta_` value is positive. For comparison, the following DATA steps create the optimal policy, `s2`, along with the other policies: `s0`, offer no one a discount; and `s1`, offer everyone a discount.

```
data mylib.discountPolicy;
  set mylib.odetails;
  s0 = 0; * treat none;
  s1 = 1; * treat all;
  s2 = _beta_ >= 0; * treat only individuals whose ITEs are positive;
  keep id t account_age age avg_hours days_visited
        friends_count has_membership is_US songs_purchased
        income revenue s0 s1 s2;
run;
```


You can do policy evaluation by specifying the POLICY= option in the INFER statement and do policy comparison by specifying the POLICYCOMPARISON= option in the same statement. The following statements estimate the model and do policy evaluation and comparison:

```

/*--- Policy evaluation and comparison ---*/
proc deepcausal data=mylib.discountPolicy;
  id id;
  psmodel t = account_age age avg_hours days_visited friends_count
              has_membership is_US songs_purchased income /
              dnn=(nodes=(32 32 32 32))
              train=(optimizer=(miniBatchSize=500 regL1=0.0001
              maxEpochs=32000 algorithm=adam) nthreads=20
              seed=12345 recordseed=67890));
  model revenue = account_age age avg_hours days_visited friends_count
                  has_membership is_US songs_purchased income /
                  dnn=(nodes=(32 32 32 32))
                  train=(optimizer=(miniBatchSize=500 regL1=0.001
                  maxEpochs=32000 algorithm=adam) nthreads=20
                  seed=12345 recordseed=67890));
  infer policy=(t s0 s1 s2) policyComparison=(base=(t)
          compare=(s0 s1 s2))
          out=mylib.oest2 outdetails=mylib.odetails2;
run;

```

The option POLICY=(t s0 s1 s2) requests that the policy evaluation be done for the initial discount policy (t), the policy of giving no one a discount (s0), the policy of giving everyone a discount (s1), and the personalized discount policy of giving a discount only to those whose `_beta_` value is positive (s2). The suboption COMPARE=(s0 s1 s2) in the POLICYCOMPARISON= option indicates that the policies s0, s1, and s2 are to be compared to the base policy, t, which is specified by the suboption BASE=(t), also in the POLICYCOMPARISON= option.

The policy evaluation results are shown in [Output 15.2.2](#).

Output 15.2.2 Policy Evaluation

The DEEPCAUSAL Procedure

Policy Evaluation						
Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
t	14.383749	0.068443	14.249603	14.517895	210.16	<.0001
s0	14.309670	0.074717	14.163227	14.456112	191.52	<.0001
s1	13.592299	0.058081	13.478463	13.706135	234.02	<.0001
s2	14.835283	0.069198	14.699658	14.970909	214.39	<.0001

The optimal policy, s2, indeed has the highest estimate. When the discount is given to customers whose ITE is positive, the average outcome, which in this example is the revenue for each person, is higher than the average outcome that is obtained by the base policy. This is the largest revenue that is generated, and it is generated by the personalized discount policy.

[Output 15.2.3](#) shows the output for the policy comparison. Negative values indicate that the corresponding discount policy is worse than the base policy, and positive values indicate that the corresponding discount policy is better than the base policy. As expected, s2, giving the discount to the right person, produces the

best results.

Output 15.2.3 Policy Comparison

Policy Comparison							
Comparison Policy	Base Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
s0	t	-0.074079	0.017349	-0.108082	-0.040077	-4.27	<.0001
s1	t	-0.791450	0.014168	-0.819218	-0.763681	-55.86	<.0001
s2	t	0.451535	0.015455	0.421243	0.481826	29.22	<.0001

References

- Farrell, M. H., Liang, T., and Misra, S. (2020). “Deep Learning for Individual Heterogeneity.” <https://arxiv.org/abs/2010.14694>.
- Farrell, M. H., Liang, T., and Misra, S. (2021). “Deep Neural Networks for Estimation and Inference.” *Econometrica* 89:181–213.

Chapter 16

DEEPPRICE Procedure

Contents

Overview: DEEPPRICE Procedure	954
Using CAS Sessions and CAS Engine Librefs	955
Getting Started: DEEPPRICE Procedure	956
A Demand-Price Example When the Price Is Not Endogenous	956
A Demand-Price Example When the Price Is Endogenous	961
Syntax: DEEPPRICE Procedure	966
Functional Summary	967
PROC DEEPPRICE Statement	968
DNN Statement	969
ID Statement	977
INFER Statement	977
INSTRUMENT Statement	978
MODEL Statement	978
SCORE Statement	979
TMODEL Statement	981
Details: The DEEPPRICE Procedure	982
Causal Inference Overview	982
Estimating Unknown Functions in Treatment and Outcome Models	983
Estimating Parameters of Interest	986
Missing Values	991
Data Table Output	991
ODS Table Names	994
Examples: DEEPPRICE Procedure	994
Example 16.1: Personalized Customer Segmentation for an Online Media Company	994
Example 16.2: Functional Forms and Analyzing Price and Demand	1010
References	1025

Overview: DEEPPRICE Procedure

The DEEPPRICE procedure estimates the average causal effect and performs policy evaluation and policy comparison by using deep neural networks (DNNs) when the treatment variable is continuous. DNNs overcome several technical difficulties in the big data era, including the following:

- the huge amount of discrete or continuous potential covariates
- the unknown nonlinear relationships among the covariates, the outcome, and the treatment variables

A problem in applying DNNs to causal inference is interpretability. To solve this problem, the DEEPPRICE procedure applies DNNs via a two-step semiparametric framework (see the section “[Details: The DEEPPRICE Procedure](#)” on page 982) and provides inferential results for the parameters of interest through the corresponding influence functions.

The DEEPPRICE procedure can estimate two types of causal effects (or parameters of interest): the average intercept and the average slope. For more information, see the section “[Average Intercept and Slope Parameters](#)” on page 987. The procedure can also perform policy evaluation and policy comparison. For more information, see the section “[Policy Evaluation and Comparison](#)” on page 990. PROC DEEPPRICE supports the SCORE statement, which enables you to save causal model specifications and estimation results for scoring and policy evaluation and comparison without needing to reestimate the DNNs for these models.

Tools for causal analysis of nonrandomized data are available in the following procedures:

- The CAUSALGRAPH procedure enables you to analyze graphical causal models and to construct sound statistical strategies for causal effect estimation. For more information, see Chapter 37, “The CAUSALGRAPH Procedure” (*SAS/STAT User’s Guide*).
- The PSMATCH procedure enables you to perform propensity score analyses and to assess covariate balance. For more information, see Chapter 101, “The PSMATCH Procedure” (*SAS/STAT User’s Guide*).
- The CAUSALTRT procedure enables you to estimate a causal effect. For more information, see Chapter 39, “The CAUSALTRT Procedure” (*SAS/STAT User’s Guide*).
- The CAUSALMED procedure enables you to decompose a (total) causal effect into direct and indirect effects. For more information, see Chapter 38, “The CAUSALMED Procedure” (*SAS/STAT User’s Guide*).
- The DEEPCAUSAL procedure enables you to estimate the average causal effect and perform policy evaluation and policy comparison by using deep neural networks (DNNs) when the treatment variable is binary. For more information, see Chapter 15, “[DEEPCAUSAL Procedure](#).”
- The MODEL, PANEL, QLIM, and TMODEL procedures in SAS/ETS and the CPANEL and CQLIM procedures in SAS Econometrics support endogenous variables and instrumental variables. For more information, see Chapter 24, “MODEL Procedure” (*SAS/ETS User’s Guide*), Chapter 25, “PANEL Procedure” (*SAS/ETS User’s Guide*), Chapter 27, “QLIM Procedure” (*SAS/ETS User’s Guide*), Chapter 39, “TMODEL Procedure” (*SAS/ETS User’s Guide*), Chapter 11, “CPANEL Procedure,” and Chapter 12, “CQLIM Procedure.”

- The VARMAX procedure supports Granger causality tests. For more information, see Chapter 42, “VARMAX Procedure” (*SAS/ETS User’s Guide*).

PROC DEEPPRICE requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: DEEPPRICE Procedure

This section provides two examples that demonstrate some of the capabilities of the DEEPPRICE procedure.

A Demand-Price Example When the Price Is Not Endogenous

This example illustrates how the DEEPPRICE procedure enables you to estimate the parameters of interest and perform policy evaluation and policy comparison when the treatment variable is not endogenous. In this case, the instrumental variable is not needed. This example also demonstrates how you can save the model information and use it later to score other new data.

A simulated data set is generated, based on the following assumptions:

- The treatment variable, scaled flight price (denoted by p), is continuous, and its value ranges from 0.01 to 75, inclusive.
- The outcome variable, demand (denoted by y), is continuous and nonnegative.
- There are two covariates:
 - The customer type (denoted by s), is discrete, and its value ranges from 1 to 10, inclusive.
 - The time (denoted by t) is continuous, and its value ranges from 0 to 12, exclusive.
- The disturbance in the treatment model is independent of the disturbance in the outcome model; this implies that the treatment variable is not endogenous.

The treatment model is given as

$$\begin{aligned}
 p &= \eta_0(t) + \eta, \quad \eta \stackrel{\text{iid}}{\sim} N(0, 1) \\
 \eta_0(t) &= 5 + 0.3\psi(t) \\
 \psi(t) &= 2 \left(\frac{50}{3} \left(\frac{t-6}{12} \right)^4 + \exp \left(-400 \left(\frac{t-6}{12} \right)^2 \right) + \frac{t}{12} - \frac{41}{20} \right)
 \end{aligned}$$

where $\psi(\cdot)$ describes the seasonality.

The outcome model is given as

$$\begin{aligned}
 y &= \alpha(s, t) + \beta(s, t)p + \epsilon, \quad \epsilon \stackrel{\text{iid}}{\sim} N(0, 0.25) \\
 \alpha(s, t) &= 30 + 0.1s\psi(t) \\
 \beta(s, t) &= 0.1s\psi(t) - 0.2
 \end{aligned}$$

Note that ϵ and η are uncorrelated, which implies that the scaled flight price is not endogenous.

The following statements simulate 12,500 samples that are based on the treatment and outcome models that were just presented:

```

%macro deepPriceGenerateData(n,rho,seed);
  data deepPriceDGP;
    call streaminit('pcg', &seed.);
    rho = &rho.;
    do i = 1 to &n.;
      s = floor(rand('Uniform')*10+1);
      t = rand('Uniform')*12;
      z = rand('Normal',0,1);
      eta = rand('Normal',0,1);
      psi = 2*((t-6)/12*(t-6)/12*(t-6)/12*(t-6)/12*50/3
        +exp(-400*(t-6)/12*(t-6)/12)+t/12-41/20);
      eta0 = 5+0.3*psi;
      eta1 = 0.1*psi;
      if(rho ^= 0) then p = eta0 + eta1*z + eta;
      else p = eta0 + eta;
      plb = 0.01;
      pub = 75;
      if(p<plb) then p=plb;
      if(p>pub) then p=pub;

      eps = rand('Normal',rho*eta,0.25);
      alpha = 30 + 0.1*s*psi;
      beta = 0.1*s*psi - 0.2;
      if(beta>0) then beta=0;
      y0 = alpha + beta*p;
      y = y0 + eps;
      if(y<0) then y = 0;

      * the realized revenue;
      R = y * p;
      R0 = y0 * p;

      * the best price to maximize the revenue,
        which cannot be obtained without knowing DGP;
      if(beta<0) then pstar = -alpha/beta/2;
      else pstar = pub;
      ystar = alpha + beta * pstar;
      * the maximized expected revenue;
      Rstar = ystar * pstar;

      output;
    end;
  run;
%mend;

%let n=12500;
%let rho = 0;
%let seed=12346;
%deepPriceGenerateData(&n.,&rho.,&seed.);

```

The following DATA steps load the first 10,000 observations of the data set deepPriceDGP into the data table simPrice_train and the next 2,500 observations into the data table simPrice_test in your session that is associated with the mylib libref. The DATA steps assume that your libref is named mylib, but you can

substitute any appropriately defined libref.

```
data mylib.simPrice_train;
  set deepPriceDGP(obs=10000 keep=i s t p y R pstar);
run;

data mylib.simPrice_test;
  set deepPriceDGP(firstobs=10001 keep=i s t p y R pstar);
run;
```

The unknown functions $\eta_0(\cdot)$, $\alpha(\cdot)$, and $\beta(\cdot)$ are estimated through two DNNs by using the data set `simPrice_train`. The parameters of interest are then estimated, and policy evaluation and policy comparison are performed. The following statements use PROC DEEPPRICE to perform these tasks and store the model information and parameter learning results:

```
proc deepprice data=mylib.simPrice_train;
  id i;
  tmodel p=s t / dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=2000 regL1=0.001 maxEpochs=2000
      algorithm=adam) nthreads=5 seed=12346 recordseed=12346));
  model y=s t / dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=2000 regL1=0.001 maxEpochs=4000
      algorithm=adam) nthreads=5 seed=12346 recordseed=12346));
  infer policy=(pstar) policyComparison=(base=(p) compare=(pstar))
    out=mylib.oest outdetails=mylib.odetails;
  score outt=mylib.outtdata outa=mylib.outadata outb=mylib.outbdata;
run;
```

Note that specifying the `SEED=` and `RECORDSEED=` option values does not guarantee that you get identical results each time you run the DEEPPRICE procedure in single-machine or distributed computing mode. This is because the execution environment is the combination of the operating environment and the number of threads (single-machine mode) or the number of worker nodes, together with the number of threads that are used for execution on each worker node (distributed computing mode).

The `OUTT=` option writes the treatment model information to the data table `outtdata`, the `OUTA=` option writes the $\alpha(\cdot)$ estimation information to the data table `outadata`, and the `OUTB=` option writes the $\beta(\cdot)$ estimation information to the data table `outbdata`. The number of observations is shown in Figure 16.1.

Figure 16.1 Number of Observations

The DEEPPRICE Procedure

Number of Observations	10000
Observations with Missing Covariates	0
Observations with Missing Covariates for Treatment Model	0
Observations with Missing Covariates for Outcome Model	0
Observations with Missing Outcome	0
Observations with Missing Treatment	0

The estimates of parameters of interest are shown in Figure 16.2.

Figure 16.2 Parameters of Interest

Parameter Estimates						
Parameter	Estimate	Standard	95%		Z	Pr > Z
		Error	Confidence Limits			
Average Intercept	28.442737	0.032558	28.378925	28.506549	873.61	<.0001
Average Slope	-1.515271	0.011544	-1.537896	-1.492646	-131.27	<.0001

The policy evaluation estimates the expected demand of a policy. The policy evaluation results are shown in Figure 16.3.

Figure 16.3 Policy Evaluation

Policy Evaluation						
Policy	Estimate	Standard	95%		Z	Pr > Z
		Error	Confidence Limits			
pstar	14.671826	0.107646	14.460845	14.882808	136.30	<.0001

The policy comparison estimates the difference in expected demand of two policies. The policy comparison results are shown in Figure 16.4.

Figure 16.4 Policy Comparison

Policy Comparison						
Comparison Policy	Base Policy	Estimate	Standard Error	95% Confidence Limits		Z Pr > Z
pstar	p	-7.273572	0.111963	-7.493016	-7.054129	-64.96 <.0001

According to the comparison of the two policies, the policy p in the observed data generates more demand on average than the policy $pstar$; however, this does not mean that p generates more revenue than $pstar$! In fact, in the real world, although $pstar$ is theoretically the best policy to maximize revenue, it cannot be available because it requires the unknown functions $\alpha(\cdot)$ and $\beta(\cdot)$. Because of the availability of the true data generating process (DGP), the revenue that the optimized policies generate that is suggested by the plug-in estimator and influence-function estimator can be compared to that of the best-in-theory infeasible policy $pstar$. The following statements compare the actual function values in the true DGP to the estimates of $\eta_0(\cdot)$, $\alpha(\cdot)$, and $\beta(\cdot)$, which are `_that_`, `_alpha_`, and `_beta_`, respectively, in the `oDetails` data table in your session; compare the revenues that are generated through different policies; and use PROC MEANS to get the bias and variance of each estimator:

```
proc sort data=mylib.odetails out=odetails_tmp;
  by i;
run;

data odetails;
  merge odetails_tmp(in=a) deepPriceDGP(in=b where=(i<=10000));
  by i;
  diff_eta0 = _that_ - eta0;
  diff_alpha = _alpha_ - alpha;
  diff_beta = _beta_ - beta;
  * revenue difference between observed policy and best policy;
  diff_R = R0 - Rstar;
```

```

* pipstar, the optimized policy suggested by plug-in estimator;
if(_beta_<0) then pipstar = -_alpha/_beta_/2;
else pipstar = pub;
if(pipstar>pub) then pipstar = pub;
if(pipstar<plb) then pipstar = plb;
piystar = alpha + beta * pipstar;
if(piystar<0) then piystar=0;
piRstar = piystar * pipstar;
* revenue difference between plug-in optimized policy and best policy;
diff_piR = piRstar - Rstar;
* ifpstar, the optimized policy suggested by influence-function
  estimator;
if(_if_beta_<0) then ifpstar = -_if_alpha/_if_beta_/2;
else ifpstar = pub;
if(ifpstar>pub) then ifpstar = pub;
if(ifpstar<plb) then ifpstar = plb;
ifystar = alpha + beta * ifpstar;
if(ifystar<0) then ifystar=0;
ifRstar = ifystar * ifpstar;
* revenue difference between influence-function optimized policy and
  best policy;
diff_ifR = ifRstar - Rstar;
run;

proc means data=odetails;
  var diff_eta0 diff_alpha diff_beta diff_R diff_ifR diff_piR;
run;

```

The results are displayed in Figure 16.5, which shows that all three estimated functions are unbiased.

Figure 16.5 Summary Statistics of Difference between Actual Value and Estimate

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
diff_eta0	10000	0.0513895	0.1463623	-0.4907291	0.2826009
diff_alpha	10000	-0.1021674	0.6435525	-2.4314195	1.5739562
diff_beta	10000	0.0683319	0.1840845	-0.4170375	0.8771066
diff_R	10000	-115.4171558	154.0205778	-996.2989116	-7.750115E-6
diff_ifR	10000	-44.9515312	100.6869697	-653.3786359	-3.722889E-9
diff_piR	10000	-24.8457316	80.2110640	-554.8558027	-1.849463E-8

You can apply the previously saved treatment model, the $\alpha(\cdot)$ estimation information, and the $\beta(\cdot)$ estimation information to a new data set by specifying the INT=, INA=, and INB= options, respectively, in the SCORE statement. The following statements use the outtdata, outadata, and outbdata data tables as input data tables and then estimate the parameters of interest and perform the policy evaluation and comparison by using the simPrice_test data table:

```

proc deepprice data=mylib.simPrice_test;
  id i;
  infer policy=(pstar) policyComparison=(base=(p) compare=(pstar))
    out=mylib.oest2 outdetails=mylib.odetails2;
  score int=mylib.outtdata ina=mylib.outadata inb=mylib.outbdata;
run;

```

When you specify the INT=, INA=, and INB= options in the SCORE statement, you do not need to specify the TMODEL or MODEL statement. The estimates of the parameters of interest and the results of the policy evaluation and policy comparison are shown in [Figure 16.6](#).

Figure 16.6 Parameters of Interest, Policy Evaluation, and Policy Comparison

The DEEPPRICE Procedure							
Parameter Estimates							
Parameter	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z	
Average Intercept	28.348928	0.071672	28.208453	28.489403	395.53	<.0001	
Average Slope	-1.507769	0.024143	-1.555090	-1.460449	-62.45	<.0001	

Policy Evaluation							
Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z	
pstar	14.662468	0.166886	14.335377	14.989560	87.86	<.0001	

Policy Comparison							
Comparison Policy	Base Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
pstar	p	-7.222689	0.178589	-7.572718	-6.872660	-40.44	<.0001

For more information about the estimates of parameters of interest, policy evaluation, and policy comparison, see the sections “[Average Intercept and Slope Parameters](#)” on page 987 and “[Policy Evaluation and Comparison](#)” on page 990.

A Demand-Price Example When the Price Is Endogenous

This example illustrates how the DEEPPRICE procedure estimates the parameters of interest and performs policy evaluation and policy comparison when the treatment variable is endogenous. In this case, the instrumental variable should be used. This example also demonstrates how you can save the model information and use it later to score other new data.

A simulated data set is generated, based on the following assumptions:

- The treatment variable, scaled flight price (denoted by p), is continuous, and its value ranges from 0.01 to 75, inclusive.
- The outcome variable, demand (denoted by y), is continuous and nonnegative.
- The instrumental variable, normalized oil price (denoted by z), is continuous and available.
- There are two covariates:
 - The customer type (denoted by s) is discrete, and its value ranges from 1 to 10, inclusive.
 - The time (denoted by t) is continuous, and its value ranges from 0 to 12, exclusive.

- The disturbance in the treatment model is correlated with the disturbance in the outcome model. This implies that the treatment variable is endogenous.

The treatment model is given as

$$\begin{aligned}
 p &= \eta_0(t) + \eta_1(t)z + \eta, \quad \eta \stackrel{\text{iid}}{\sim} N(0, 1) \\
 \eta_0(t) &= 5 + 0.3\psi(t) \\
 \eta_1(t) &= 0.1\psi(t) \\
 \psi(t) &= 2 \left(\frac{50}{3} \left(\frac{t-6}{12} \right)^4 + \exp \left(-400 \left(\frac{t-6}{12} \right)^2 \right) + \frac{t}{12} - \frac{41}{20} \right)
 \end{aligned}$$

where $\psi(\cdot)$ describes the seasonality.

The outcome model is given as

$$\begin{aligned}
 y &= \alpha(s, t) + \beta(s, t)p + \epsilon, \quad \epsilon \stackrel{\text{iid}}{\sim} N(2\eta, 0.25) \\
 \alpha(s, t) &= 30 + 0.1s\psi(t) \\
 \beta(s, t) &= 0.1s\psi(t) - 0.2
 \end{aligned}$$

Note that ϵ and η are correlated, which indicates that the scaled flight price is endogenous.

The following statements simulate 12,500 samples that are based on the treatment and outcome models that were just presented:

```

%let n=12500;
%let rho = 2;
%let seed=12346;
%deepPriceGenerateData(&n., &rho., &seed.);

```

The following DATA steps load the first 10,000 observations of the data set `deepPriceDGP` into the data table `simPrice_train` and the next 2,500 observations into the data table `simPrice_test` in your session that is associated with the `mylib` libref. The DATA steps assume that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```

data mylib.simPrice_train;
  set deepPriceDGP(obs=10000 keep=i s t z p y R pstar);
run;

data mylib.simPrice_test;
  set deepPriceDGP(firstobs=10001 keep=i s t z p y R pstar);
run;

```

The unknown functions $\eta_0(\cdot)$, $\eta_1(\cdot)$, $\alpha(\cdot)$, and $\beta(\cdot)$ are estimated through two DNNs by using the `simPrice_train` data table. The parameters of interest are then estimated, and the policy evaluation and policy comparison are performed. The instrumental variable is introduced into the model by using the `INSTRUMENT` statement. The following statements use PROC DEEPPRICE to perform these tasks and store the model information and parameter learning results:

```

proc deepprice data=mylib.simPrice_train;
  id i;
  instrument z;
  tmodel p=s t / dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=2000 regL1=0.001 maxEpochs=2000
      algorithm=adam) nthreads=5 seed=12346 recordseed=12346));
  model y=s t / dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=2000 regL1=0.001 maxEpochs=8000
      algorithm=adam) nthreads=5 seed=12346 recordseed=12346));
  infer policy=(pstar) policyComparison=(base=(p) compare=(pstar))
    out=mylib.oest outdetails=mylib.odetails;
  score outeta0=mylib.outeta0data outeta1=mylib.outeta1data
    outa=mylib.outadata outb=mylib.outbdata;
run;

```

The OUTETA0= option writes the $\eta_0(\cdot)$ estimation information to the data table outeta0data, the OUTETA1= option writes the $\eta_1(\cdot)$ estimation information to the data table outeta1data, the OUTA= option writes the $\alpha(\cdot)$ estimation information to the data table outadata, and the OUTB= option writes the $\beta(\cdot)$ estimation information to the data table outbdata. The number of observations is shown in Figure 16.7.

Figure 16.7 Number of Observations

The DEEPPRICE Procedure

Number of Observations	10000
Observations with Missing Covariates	0
Observations with Missing Covariates for Treatment Model	0
Observations with Missing Covariates for Outcome Model	0
Observations with Missing Outcome	0
Observations with Missing Treatment	0
Observations with Missing Instrumental Variable	0

The estimates of parameters of interest are shown in Figure 16.8.

Figure 16.8 Parameters of Interest

Parameter Estimates						
Parameter	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
Average Intercept	28.832408	0.713265	27.434435	30.230382	40.42	<.0001
Average Slope	-1.594396	0.154868	-1.897932	-1.290860	-10.30	<.0001

The policy evaluation estimates the expected demand of a policy. The policy evaluation results are shown in Figure 16.9.

Figure 16.9 Policy Evaluation

Policy Evaluation						
Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
pstar	10.807473	6.861139	-2.640113	24.255058	1.58	0.1152

The policy comparison estimates the difference in expected demand between two policies. The policy comparison results are shown in Figure 16.10.

Figure 16.10 Policy Comparison

Policy Comparison							
Comparison Policy	Base Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
pstar	p	-10.975870	6.871267	-24.443305	2.491566	-1.60	0.1102

According to the comparison of two policies, the policy p in the observed data generates more demand on average than the policy $pstar$; however, this does not mean that p generates more revenue than $pstar$! In fact, in the real world, although $pstar$ is theoretically the best policy to maximize revenue, it cannot be available because it needs the unknown functions $\alpha(\cdot)$ and $\beta(\cdot)$. Because of the availability of the true data generating process (DGP), the revenue that the optimized policies generate that is suggested by the plug-in estimator and influence-function estimator can be compared to that of the best-in-theory infeasible policy $pstar$. The following statements compare the actual function values in the true DGP to the estimates of $\eta_0(\cdot)$, $\eta_1(\cdot)$, $\alpha(\cdot)$, and $\beta(\cdot)$, which are `_eta0_`, `_eta1_`, `_alpha_`, and `_beta_`, respectively, in the `oDetails` data table in your session; compare the revenues that are generated through different policies; and use PROC MEANS to get the bias and variance of each estimator:

```
proc sort data=mylib.odetails out=odetails_tmp;
  by i;
run;

data odetails;
  merge odetails_tmp(in=a) deepPriceDGP(in=b where=(i<=10000));
  by i;
  diff_eta0 = _eta0_ - eta0;
  diff_eta1 = _eta1_ - eta1;
  diff_alpha = _alpha_ - alpha;
  diff_beta = _beta_ - beta;
  * revenue difference between observed policy and best policy;
  diff_R = R0 - Rstar;
  * ifpstar, the optimized policy suggested by influence-function
    estimator;
  if(_if_beta_ < 0) then ifpstar = -_if_alpha_/_if_beta_/2;
  else ifpstar = pub;
  if(ifpstar > pub) then ifpstar = pub;
  if(ifpstar < plb) then ifpstar = plb;
  ifystar = alpha + beta * ifpstar;
  if(ifystar < 0) then ifystar = 0;
  ifRstar = ifystar * ifpstar;
  * revenue difference between influence-function optimized policy and
    best policy;
  diff_ifR = ifRstar - Rstar;
  * pipstar, the optimized policy suggested by plug-in estimator;
  if(_beta_ < 0) then pipstar = -_alpha_/_beta_/2;
  else pipstar = pub;
  if(pipstar > pub) then pipstar = pub;
  if(pipstar < plb) then pipstar = plb;
  piystar = alpha + beta * pipstar;
```

```

    if(piystar<0) then piystar=0;
    piRstar = piystar * pipstar;
    * revenue difference between plug-in optimized policy and best policy;
    diff_piR = piRstar - Rstar;
run;

proc means data=odetails;
    var diff_eta0 diff_eta1 diff_alpha diff_beta diff_R diff_ifR diff_piR;
run;

```

The results are displayed in Figure 16.11, which shows that all four estimated functions are unbiased and also that the plug-in optimized policy is the best feasible policy.

Figure 16.11 Summary Statistics for Differences
The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
diff_eta0	10000	-0.0054662	0.1393846	-0.5600274	0.2398572
diff_eta1	10000	-0.0101667	0.0514571	-0.2134025	0.0734339
diff_alpha	10000	-0.0288887	2.4017872	-4.2549535	8.2332300
diff_beta	10000	0.0440188	0.5654596	-1.8312735	1.3122532
diff_R	10000	-115.5029685	153.9612042	-996.3512062	-2.001858E-7
diff_ifR	10000	-132.0524553	148.7798031	-990.7514537	-2.105923E-8
diff_piR	10000	-76.3508287	133.3256657	-813.2234595	-4.053138E-8

You can apply the previously saved $\eta_0(\cdot)$ estimation information, $\eta_1(\cdot)$ estimation information, $\alpha(\cdot)$ estimation information, and $\beta(\cdot)$ estimation information to a new data set by specifying the INETA0=, INETA1=, INA=, and INB= options, respectively, in the SCORE statement. The following statements use the outeta0data, outeta1data, outadata, and outbdata data tables as input data tables and then estimate the parameters of interest and perform the policy evaluation and comparison by using the simPrice_test data table:

```

proc deepprice data=mylib.simPrice_test;
    id i;
    instrument z;
    infer policy=(pstar) policyComparison=(base=(p) compare=(pstar))
        out=mylib.oest2 outdetails=mylib.odetails2;
    score ineta0=mylib.outeta0data ineta1=mylib.outeta1data
        ina=mylib.outadata inb=mylib.outbdata;
run;

```

When you specify the INETA0=, INETA1=, INA=, and INB= options in the SCORE statement, you do not need to specify the TMODEL or MODEL statement. The estimates of parameters of interest and the results of the policy evaluation and policy comparison are shown in Figure 16.12.

Figure 16.12 Parameters of Interest, Policy Evaluation, and Policy Comparison

The DEEPPRICE Procedure							
Parameter Estimates							
Parameter	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z	
Average Intercept	30.844464	1.176019	28.539509	33.149419	26.23	<.0001	
Average Slope	-2.030962	0.259695	-2.539955	-1.521970	-7.82	<.0001	

Policy Evaluation						
Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
pstar	-4.124391	9.355596	-22.461022	14.212240	-0.44	0.6593

Policy Comparison							
Comparison Policy	Base Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
pstar	p	-26.269043	9.476454	-44.842550	-7.695535	-2.77	0.0056

For more information about the estimates of parameters of interest, policy evaluation, and policy comparison, see the sections “[Average Intercept and Slope Parameters](#)” on page 987 and “[Policy Evaluation and Comparison](#)” on page 990.

Syntax: DEEPPRICE Procedure

The following statements are available in the DEEPPRICE procedure. Items within angle brackets (<>) are optional.

```

PROC DEEPPRICE options ;
  DNN options ;
  ID variable ;
  INFER <options> ;
  INSTRUMENT variable ;
  MODEL outcome-variable <=covariates> </options> ;
  SCORE options ;
  TMODEL treatment-variable <=covariates> </options> ;

```


Functional Summary

The statements and options available in the DEEPPRICE procedure are summarized in Table 16.1. The sections after the table provide a description of each statement in alphabetical order, following the PROC DEEPPRICE statement.

Table 16.1 Functional Summary

Description	Statement	Option
Input Data Table Options		
Specifies the input data table	PROC DEEPPRICE	DATA=
Specifies the input data table for $\alpha(\cdot)$ estimation information	SCORE	INA=
Specifies the input data table for $\beta(\cdot)$ estimation information	SCORE	INB=
Specifies the input data table for $\eta_0(\cdot)$ estimation information	SCORE	INETA0=
Specifies the input data table for $\eta_1(\cdot)$ estimation information	SCORE	INETA1=
Specifies the input data table for outcome model information	SCORE	INO=
Specifies the input data table for treatment model information	SCORE	INT=
Output Data Table Options		
Writes the estimation of the parameters of interest and the policy evaluation and comparison results to an output data set	INFER	OUTEST=
Writes the details of the estimation, including $\alpha(\cdot)$, $\beta(\cdot)$, $p(\cdot)$, the residual, and influence functions for each unit, to an output data set	INFER	OUTDETAILS=
Writes the $\alpha(\cdot)$ estimation information to the specified output data set	SCORE	OUTA=
Writes the $\beta(\cdot)$ estimation information to the specified output data set	SCORE	OUTB=
Writes the $\eta_0(\cdot)$ estimation information to the specified output data set	SCORE	OUTETA0=
Writes the $\eta_1(\cdot)$ estimation information to the specified output data set	SCORE	OUTETA1=
Writes the outcome model information to the specified output data set	SCORE	OUTO=
Writes the treatment model information to the specified output data set	SCORE	OUTT=
ID Variable		
Specifies the identifying variable	ID	

Table 16.1 *continued*

Description	Statement	Option
Instrumental Variable		
Specifies the instrumental variable	INSTRUMENT	
Deep Neural Network (DNN) Options		
Specifies the parameters of the DNN for the treatment model	TMODEL	DNN=
Specifies the parameters of the DNN for the outcome model	MODEL	DNN=
Specifies the number of nodes for all hidden layers of the DNN for the treatment and outcome models	DNN	NODES=
Specifies the training options of the DNN for the treatment and outcome models	DNN	TRAIN=
Outcome Model Variables		
Specifies the outcome variable	MODEL	<i>outcome-variable</i>
Specifies the covariates for the outcome model	MODEL	<i>covariates</i>
Treatment Model Variables		
Specifies the treatment variable	TMODEL	<i>treatment-variable</i>
Specifies the covariates for the treatment model	TMODEL	<i>covariates</i>
Inference Options		
Specifies the list of policy variables to be evaluated	INFER	POLICY=
Specifies the parameters for policy comparison	INFER	POLICYCOMPARISON=

PROC DEEPPRICE Statement

PROC DEEPPRICE *options* ;

The PROC DEEPPRICE statement invokes the DEEPPRICE procedure. You must specify the following *option*:

DATA=*libref.data-table*

names the input data table for PROC DEEPPRICE to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

data-table specifies the name of the input data table.

NOTE: The input data table must not contain any of the reserved variables (case-insensitive): *_PSHAT_*, *_YHAT_*, *_Y0_*, *_Y1_*, *_AVGPX_*, *_px_*, *_alpha_*, *_beta_*, *_eps_*, *_mu0_*, *_mu1_*, *_ate_*, *_rho00_*, *_rho01_*, *_rho10_*, *_rho11_*, *_att_*, *_atu_*, *_deltaX_*, *_delta_*, *_that_*, *_eta0_*, *_eta1_*, *_iv_rename_*, *_DL_PRED_*, *_epst_*, *_epsy_*, *_treatment_rename_*, *_alphanatilda_*, *_betatilda_*, *_if_alpha_*, or *_if_beta_*.

DNN Statement

DNN < *options* > ;

The DNN statement specifies the parameters for the deep neural network (DNN). It includes options that are related to the DNNs for the treatment model and outcome model. You can specify the following *options*:

NODES=(*numeric-list*)

specifies the number of nodes for each hidden layer in the DNN, where each value in the *numeric-list* must be a positive number and values are separated by commas or spaces. The length of the list specifies the number of layers, and each value specifies the number of nodes for the corresponding layer. If the length of the list is denoted by n , then the i th ($1 \leq i \leq n$) value denotes the number of nodes for the i th layer. By default, NODES=(32 32 32 32), which yields four hidden layers, with 32 nodes in each layer.

TRAIN=(*train-options*)

specifies the options that you can use to train the DNN. You can specify one or more of the following *train-options*:

GPU=(< (*gpu-options*) >)

specifies that the DNN perform calculations by using graphics processing unit (GPU) hardware. You can specify the following *gpu-options*:

DEVICE=(*numeric-list*)

DEVICES=(*numeric-list*)

specifies a list of GPU devices to use.

PRECISION=FP16 | FP32

specifies the computational precision in forward-backward computations. You can specify the following values:

FP16	uses half-precision (16-bit floating-point) computations to train and score the DNN. The lower computational precision in forward-backward computations can encourage tensor-core engagements.
FP32	uses single-precision (32-bit floating-point) computations to train and score the DNN.

By default, PRECISION=FP32.

USEEXCLUSIVE

specifies that only GPU devices be used.

USETensorRT

enables the use of the TensorRT software development kit for fast inference.

MISSING=MAX | MEAN | MIN | NONE

specifies the policy to use for replacing missing input covariate values with imputed values. You can specify one of the following policies:

MAX	replaces missing values with the maximum of the corresponding covariate.
MEAN	replaces missing values with the mean of the corresponding covariate.
MIN	replaces missing values with the minimum of the corresponding covariate.
NONE	ignores observations that have missing values.

By default, MISSING=MEAN. The MISSING= option is for DNN training only. In the inference that is performed later, observations that have missing covariates are removed.

NTHREADS=number

specifies the number of threads to use for the DNN.

OPTIMIZATION=(optimizer-options)**OPTIMIZE=(optimizer-options)****OPTIMIZER=(optimizer-options)**

specifies the settings for the optimization algorithm and optimization mode, as well as other settings, such as a seed, the maximum number of epochs, and so on. You can specify the following *optimizer-options*:

ALGORITHM=algorithm-keyword<(common-algorithm-options)>

specifies the optimization algorithm and the corresponding options. First, you can specify some options common to all the optimization algorithms, *common-algorithm-options*, in the parentheses that follow the *algorithm-keyword*. They are as follows:

CLIPGRADMAX=number

specifies the maximum gradient value. All gradient values that exceed the specified maximum value are set to the specified maximum value. The *number* should be greater than or equal to 0.

CLIPGRADMIN=number

specifies the minimum gradient value. All gradient values that are less than the specified minimum value are set to the specified minimum value. The *number* should be less than or equal to 0.

FCMPEARLYSTOPPING=symbol-name**FCMPES=symbol-name**

specifies the function compiler (FCMP) early stopping function.

FCMPLEARNINGRATE=symbol-name**FCMPLR=symbol-name**

specifies the function compiler (FCMP) learning rate function.

GAMMA=number

specifies the gamma value for the learning rate policy. The *number* must be between 0 and 100, exclusive. By default, GAMMA=0.1.

LEARNINGRATE=number

specifies the learning rate (a positive value) for stochastic gradient descent (SGD). By default, LEARNINGRATE=0.001.

LEARNINGRATEPOLICY=FIXED | INV | MULTISTEP | POLY | STEP**LRPOLICY=FIXED | INV | MULTISTEP | POLY | STEP**

specifies the learning rate policy. You can specify one of the following values:

FIXED	specifies a fixed learning rate.
INV	sets the learning rate option value after each epoch, according to the initial learning rate, the value of the GAMMA= option (γ), and the value of the POWER= option. The rate is calculated as $\text{learningRate} * (1 + \gamma * \text{currentEpoch})^{-\text{power}}$.
MULTISTEP	sets the learning rate after each of the epochs that you specify in the STEPS= option. The updated learning rate is the product of the previous learning rate value and the GAMMA= option value.
POLY	sets the learning rate option value after each epoch, according to the initial learning rate, the maximum number of epochs, and the value of the POWER= option. The rate is calculated as $\text{learningRate} * (1 - \text{currentEpoch}/\text{maxEpochs})^{\text{power}}$.
STEP	sets the learning rate option by multiplying the current learning rate option value by the GAMMA= option value. The number of steps is specified in the STEPSIZE= option. The configured learning rate is recalculated for each group of epochs, according to the step size.

By default, LEARNINGRATEPOLICY=FIXED.

NOUSELOCKING

computes gradients asynchronously with multiple threads.

POWER=number

specifies the power for the learning rate policy. The *number* must be nonnegative. By default, POWER=0.75.

STEPS=(numeric-list)

specifies a list of epoch counts. When the current epoch matches one of the specified steps, the learning rate is multiplied by the value of the GAMMA= option. For example, if you specify STEPS=(5 9 13), then the learning rate is multiplied by the GAMMA= option value after the 5th, 9th, and 13th epochs.

STEPSIZE=number

specifies the step size (a value greater than or equal to 1) when the learning rate policy is set to STEP. By default, STEPSIZE=10.

For the ALGORITHM= option, you can specify one of the following *algorithm-keywords* and its associated options:

ADAM<(adam-options)>**ADAMSGD<(adam-options)>**

specifies the adaptive moment estimation (ADAM) optimization algorithm. For the *adam-options*, you can specify one or more of the *common-algorithm-options*. You can also specify the following *adam-options*:

BETA1=number

specifies the exponential decay rate for the first moment in an ADAM learning algorithm. The *number* must be in the range [0, 1). By default, BETA1=0.9.

BETA2=number

specifies the exponential decay rate for the second moment in an ADAM learning algorithm. The *number* must be in the range [0, 1). By default, BETA2=0.999.

By default, ALGORITHM=ADAM.

COMPRESSION=(compression-options)

makes communication between workers more sparse for faster training by using gradient sparsification, a distributed training technique that makes stochastic gradients more sparse to reduce communication costs. You can specify one or more of the following *compression-options*:

LOCALMOMENTUM=number

determines the strength of aggregation on the masked gradient elements when compression is used. The *number* must be in the range [0, 1]. By default, LOCALMOMENTUM=0.95.

SAMPLERATIO=number

determines the number of samples for gradients in each layer. The *number* must be in the range $[10^{-5}, 0.2]$. By default, SAMPLERATIO=0.05.

SPARSITYRATIO=number

determines the number of samples for gradients in each layer. The *number* must be in the range $[0, 1]$. By default, SPARSITYRATIO=0.75.

DROPOUT=number

specifies the probability that the output of a neuron in a fully connected layer will be set to 0 during training. This probability is recalculated each time an observation is processed. The *number* must be in the range $[0, 1)$. By default, DROPOUT=0.

DROPOUTINPUT=number

specifies the probability that an input variable will be set to 0 during training. This probability is recalculated each time an observation is processed. The *number* must be in the range $[0, 1)$. By default, DROPOUTINPUT=0.

DROPOUTTYPE=INVERTED | STANDARD

specifies what type of dropout to use. You can specify one of the following types:

INVERTED	uses the inverted dropout, in which activations of some neurons are set to 0 and the remaining activations are scaled.
STANDARD	uses the standard dropout, in which activations of some neurons are set to 0 and the remaining activations are not scaled.

By default, DROPOUTTYPE=STANDARD.

FCONV=number

specifies the relative function convergence criterion. If the relative loss error, $\text{abs}[(\text{previous_loss} - \text{current_loss})/\text{previous_loss}]$, does not result in a change in the objective function, the optimization stops. The *number* must be greater than or equal to 0. By default, FCONV=0, and the relative function convergence is not checked.

FREEZEBATCHNORMSTATS**FREEZEBNSTATS**

freezes statistics of all batch normalization layers.

IGNORETRAININGERROR

continues model training without interruption by ignoring the issue of training data observations containing invalid or missing variable data. If you omit this option, training stops and PROC DEEPPRICE terminates when bad input data are found.

MAXEPOCHS=number

specifies the maximum number of epochs. The *number* must be greater than or equal to 1. By default, MAXEPOCHS=1. For SGD with a single-machine server, or for a session that uses one worker on a distributed server, one epoch is reached when the optimizer passes through the data one time. For a session that uses more than one worker, one epoch is reached when all the workers exchange the weights with the controller one time. The SYNCFREQ= option specifies the number of times that each worker passes through the

data before exchanging weights with the controller. For the L-BFGS optimization algorithm with a full batch, each iteration might process more than one epoch, and the final number of epochs might exceed the parameter value for the maximum number of epochs.

MINIBATCHBUFSIZE=*number*

specifies the number of observations to buffer in memory (including input data and intermediate calculations) before processing the remaining records in the minibatch. The *number* must be greater than or equal to 1 and must be smaller than the *number* that you specify for the MINIBATCHSIZE= option. The MINIBATCHBUFSIZE= option setting is ignored when GPUs are used.

MINIBATCHSIZE=*number*

specifies the number of observations per thread in a minibatch. The parameter controls the number of observations that are used

per worker in each thread to compute the gradient, prior to updating the weights. Larger values use more memory. The *number* is greater than or equal to 1. By default, MINIBATCHSIZE=1. When you use synchronous SGD (the default), the total minibatch size is calculated as miniBatchSize * number of threads * number of workers. When you use asynchronous SGD by specifying the ELASTICSYNCFREQ= option, each worker trains its own local model. In this case, the total minibatch size for each worker is calculated as miniBatchSize * number of threads. You can specify values for either the MINIBATCHSIZE= or TOTALMINIBATCHSIZE= option, but not for both.

MODE=*optimization-mode*

specifies the optimization mode and the corresponding options. You can specify one of the following values:

DOWNPOUR

specifies the downpour optimization mode.

ELASTIC<(elastic-options)>

specifies the elastic optimization mode. You can specify the following *elastic-options*:

ALPHA=*number*

ELASTICALPHA=*number*

specifies the significance level (a value between 0 and 1, inclusive) that is used for elastic SGD. By default, ALPHA=0.

ELASTICSYNCFREQ=*number*

SYNCFREQ=*number*

specifies the frequency (a nonnegative value) for communication between the workers and controller for exchanging weights. You can exchange the weights more often than once each epoch by specifying a *number* that is less than the number of batches in an epoch. If the *number* is greater than the number of batches in an epoch, then the weights are exchanged once each epoch. By default, ELASTICSYNCFREQ=0.

SYNCHRONOUS<(SYNCFREQ=*number*)>

specifies the synchronous optimization mode. The SYNCFREQ= option, which is not required, specifies the synchronization frequency for SGD in terms of epochs. By default, SYNCFREQ=1.

By default, MODE=SYNCHRONOUS.

NOBNSRCLAYERWARNINGS

suppresses the warning if the batch normalization source layer has an atypical type or activation setting.

REGL1=*number*

specifies the weight for the L1 regularization term. The *number* must be greater than or equal to 0. By default, REGL1=0. At the default setting, PROC DEEPPRICE does not perform L1 regularization. Initial L1 weight values should be small (such as 1E-3). You can combine L1 regularization with L2 regularization.

REGL2=*number*

specifies the weight for the L2 regularization term. The *number* must be greater than or equal to 0. By default, REGL2=0. At the default setting, PROC DEEPPRICE does not perform L2 regularization. Initial L2 weight values should be small (such as 1E-3). You can combine L2 regularization with L1 regularization.

SEED=*number*

specifies the random number seed value for the random number generator in the L-BFGS optimization algorithm. Seed values less than or equal to 0 generate random number streams by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences. By default, SEED=0.

SNAPSHOTFREQ=*number*

specifies the frequency for generating snapshots of the neural weights and storing the weights in a weight table during the training process. When you specify asynchronous SGD by specifying the ELASTICSYNCFREQ= option, PROC DEEPPRICE synchronizes all the weights before writing out the weights. The *number* must be greater than or equal to 0. By default, SNAPSHOTFREQ=0.

STAGNATION=*number*

specifies the number of iterations to complete without improvement before stopping the optimization early. The *number* must be greater than or equal to 0. By default, STAGNATION=0; this turns off the stagnation monitoring.

THRESHOLD=*number*

specifies the threshold that is used to determine whether the loss error or validation scores are improving or stagnating across iterations. The iteration does not improve when the magnitude of the score change between successive iterations is less than or equal to the absolute value of current_score * threshold value. When an iteration does not improve, the stagnation counter increments. Otherwise, the stagnation counter is set to 0. The *number* must be greater than or equal to 0. By default, THRESHOLD=10⁻⁸.

TOTALMINIBATCHSIZE=(totalminibatchsize-options)

specifies the number of observations in a minibatch. You can specify one or more of the following *totalminibatchsize-options*:

NONOTIFY

suppresses the message that is printed if the total minibatch size is rounded up.

ROUND

rounds up the total minibatch size.

SIZE=number**N=number**

specifies the number of observations in a minibatch. The computation for the total minibatch size is $nWorkers * miniBatchSize * nThreads$. For example, a two-worker scenario with a minibatch size of 10 and 32 threads per worker has a total minibatch size of $2 * 32 * 10 = 640$ records.

You can use these options to control the number of observations that the DNN uses to compute the gradient prior to updating the weights. Larger values use more memory. If the specified size cannot be evenly divided by the number of threads (if you are using asynchronous SGD) or by the number of threads * number of workers (if you are using synchronous SGD), then PROC DEEPPRICE terminates with an error unless you specify the ROUND option. In that case, the total minibatch size is rounded up so that it is evenly divided. You can specify values for either the MINIBATCHSIZE= or TOTALMINIBATCHSIZE= option, but not for both.

RECORDSEED=number

specifies the random number seed for random record selection within a worker. Records are read in the order in which they are written in memory. Seed values that are less than or equal to 0 generate random number streams by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences. By default, RECORDSEED=0; this disables random record selection.

SEED=number

specifies the random number seed value for the random number generator in SGD. Seed values that are less than or equal to 0 generate random number streams by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences. By default, SEED=0.

TARGETMISSING=MAX | MEAN | MIN | NONE

specifies the policy to use for replacing missing outcome variables with imputed values. You can specify one of the following policies:

MAX	replaces missing values with the maximum.
MEAN	replaces missing values with the mean value.
MIN	replaces missing values with the minimum.
NONE	ignores observations that have missing target variables.

By default, TARGETMISSING=MEAN. The TARGETMISSING= option is for DNN training only. In the inference that is performed later, observations that have missing outcome variables have missing values for residuals and for any estimates that need residuals.

ID Statement

ID *variable* ;

The ID statement specifies the ID variable in the data set. The *variable* must have a unique value for each row in the input data table. If it does not, PROC DEEPPRICE issues an error.

INFER Statement

INFER *options* ;

The INFER statement specifies the parameters for inference. You can specify one or more of the following *options*:

OUT=*libref.data-table*

OUTEST=*libref.data-table*

writes the estimation of the parameters of interest and the policy evaluation and policy comparison results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

OUTDETAILS=*libref.data-table*

writes the details of the estimation, including $\alpha(\cdot)$, $\beta(\cdot)$, $p(\cdot)$, the residual, and the influence functions for each unit, to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

POLICY=(*policy-variable-list*)

specifies the list of policy variables to be evaluated. All symbol names that the user specifies in the *policy-variable-list* must correspond to the policy variables; the symbol names are separated by spaces.

POLICYCOMPARISON=(*comparison-options*)

specifies the options for policy comparison. You must specify the following *comparison-options*:

BASE=(*base-list*)

specifies a list of policy variables, each of which is used as a base policy in the policy comparison. All symbol names that the user specifies in the *base-list* must correspond to the policy variables; the symbol names are separated by spaces.

COMPARE=(compare-list)

specifies a list of policy variables, each of which is compared to each base policy that you specify in the BASE= option. All symbol names that the user specifies in the *compare-list* must correspond to the policy variables; the symbol names are separated by spaces.

INSTRUMENT Statement

INSTRUMENT *variable* ;

The INSTRUMENT statement specifies the instrumental variable in the data table. The *variable* must be numeric in the input data table.

The instrumental variable cannot be the same as any of the following: the ID variable that you specify in the ID statement; the treatment variables or covariates that you specify in the TREATMENT statement; or the outcome variable or covariates that you specify in the MODEL statement.

When you are scoring by using the INT=, INETA0=, or INETA1= option in the SCORE statement for a treatment model that contains the instrumental variable, you must specify the INSTRUMENT statement in exactly same way (by using the same instrumental variable name) that you specify the INSTRUMENT statement used in the previous call of PROC DEEPPRICE that generates the corresponding OUTT=, OUTETA0=, or OUTETA1= table.

MODEL Statement

MODEL *outcome-variable* <=*covariates*> </*options*> ;

The MODEL statement specifies options that are related to the outcome model and the DNN to fit the outcome model. The *outcome-variable* specifies the outcome variable of the outcome model. The *covariates* specify the covariates of the outcome model. If you omit the *covariates*, no DNN is trained for the outcome model. For more information about the outcome model, see the section “[Outcome Model](#)” on page 984. Covariate names must be separated by spaces. The treatment variable is automatically included in the outcome model as an independent variable.

You can specify the following *option* after a forward slash (/):

DNN=(dnn-options)

specifies the options of the DNN for the outcome model. If you omit the DNN= option and specify the DNN statement, the DNN for the outcome model has the options that you specify in the DNN statement. If you omit both the DNN= option and the DNN statement, the DNN for the outcome model uses the default option values. If you specify both the DNN= option and the DNN statement, the DNN statement is ignored for the outcome model. For the *dnn-options*, you can specify one or more options that are listed in the section “[DNN Statement](#)” on page 969. If no covariates are specified in the MODEL statement, the DNN= option is ignored.

SCORE Statement

SCORE options ;

The SCORE statement specifies options that are related to scoring data. You can specify the following options:

INA=*libref.data-table*

INALPHA=*libref.data-table*

specifies the input data table that contains information about $\alpha(\cdot)$ estimation. The data table must be an output data table that the OUTA= option produces in a previous call of PROC DEEPPRICE. When you specify the INA= option, you must also specify the INB= option, and the MODEL statement and the OUTA=, OUTB=, and OUTO= options in the SCORE statement are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

INB=*libref.data-table*

INBETA=*libref.data-table*

specifies the input data table that contains information about $\beta(\cdot)$ estimation. The data table must be an output data table that the OUTB= option produces in a previous call of PROC DEEPPRICE. When you specify the INB= option, you must also specify the INA= option, and the MODEL statement and the OUTA=, OUTB=, and OUTO= options in the SCORE statement are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

INETA0=*libref.data-table*

specifies the input data table that contains information about $\eta_0(\cdot)$ estimation. If you omit the INSTRUMENT statement, this option is ignored. The data table must be an output data table that the OUTETA0= option produces in a previous call of PROC DEEPPRICE. When you specify the INETA0= option, you must also specify the INETA1= option, and the TMODEL statement and the OUTETA0=, OUTETA1=, and OUTT= options in the SCORE statement are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

INETA1=*libref.data-table*

specifies the input data table that contains information about $\eta_1(\cdot)$ estimation. If you omit the INSTRUMENT statement, this option is ignored. The data table must be an output data table that the OUTETA1= option produces in a previous call of PROC DEEPPRICE. When you specify the INETA1= option, you must also specify the INETA0= option, and the TMODEL statement and the OUTETA0=, OUTETA1=, and OUTT= options in the SCORE statement are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

INO=libref.data-table**INOUTCOME=libref.data-table**

specifies the input data table that contains information about the outcome model. The data table must be an output data table that the OUTO= option produces in a previous call of PROC DEEPPRICE. When you specify the INO= option, the MODEL statement and the INA=, INB=, OUTA=, OUTB=, and OUTO= options in the SCORE statement are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

INT=libref.data-table**INPROPNESITYSCORE=libref.data-table**

specifies the input data table that contains information about the treatment model. The data table must be an output data table that the OUTT= option produces in a previous call of PROC DEEPPRICE. When you specify the INT= option, the TMODEL statement and the OUTT= option in the SCORE statement are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

OUTA=libref.data-table**OUTALPHA=libref.data-table**

writes the $\alpha(\cdot)$ estimation information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

OUTB=libref.data-table**OUTBETA=libref.data-table**

writes the $\beta(\cdot)$ estimation information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

OUTETA0=libref.data-table

writes the $\eta_0(\cdot)$ estimation information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

OUTETA1=libref.data-table

writes the $\eta_1(\cdot)$ estimation information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

OUTO=libref.data-table**OUTOUTCOME=libref.data-table**

writes the outcome model information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

OUTT=libref.data-table**OUTPROPENSITYSCORE=libref.data-table**

writes the treatment model information to the specified output data table. The output data table consists of binary large object (BLOB) columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 955.

TMODEL Statement

TMODEL *treatment-variable* <=*covariates*> </options> ;

The TMODEL statement specifies options that are related to the treatment model and the DNN that is used to fit the treatment model. The *treatment-variable* specifies the treatment variable of the treatment model. It corresponds to one numeric variable in the input data table. The *covariates* specify the covariates of the treatment model. All covariate names are separated by spaces. If you omit the *covariates* and also omit the INSTRUMENT statement, no DNN is trained for the treatment model. For more information about the treatment model, see the section “Treatment Model” on page 983.

You can specify the following *option* after a forward slash (/):

DNN=(dnn-options)

specifies the options of the DNN that is used to fit the treatment model. If you omit the DNN= option and specify the DNN statement, the DNN for the treatment model has the options that you specify in the DNN statement. If you omit both the DNN= option and the DNN statement, the DNN for the treatment model uses the default option values. If you specify both the DNN= option and the DNN statement, the DNN statement is ignored for the treatment model. For the *dnn-options*, you can specify

one or more options that are listed in the section “DNN Statement” on page 969. If no *covariates* are specified in the TMODEL statement, the DNN= option is ignored.

Details: The DEEPPRICE Procedure

Causal Inference Overview

Causal inference is the science of why. Many classical statistics models, such as linear regression, show how the outcome variable is associated with the covariates. It is well known that correlation does not equal causation. However, it is often not known whether a change in the outcome is caused by a change in the treatment. Randomized experiments provide a solution, but there are many constraints that often make the experiment impossible. For example, designing and then conducting the experiment is time-consuming and expensive. Moreover, an experiment might encounter some ethical issues that prevent the experiment from being conducted. Therefore, causal inference based on observational data is indispensable to answering the causal questions.

The DEEPPRICE procedure is designed primarily for observational data when the subjects are not randomly assigned to the treatment conditions. But in some situations the randomized trials need to be treated as observational studies. For example, you can randomly assign subjects to the treatment group, or you can use block randomization when there are several confounding variables. However, after the experiment, what if you discover a few more confounding variables? Or worse yet, what if the treatment and control groups do not have similar distributions across the segments of one or more new confounding variables? In these cases, instead of doing the experiment again, you need to estimate the causal effect by treating the experiment as an observational study. Moreover, selection bias can affect the estimation of the treatment effect. For example, suppose that there is a new version of a smartphone app, and the relevant notifications are sent to randomly selected individuals, who can be seen as the treatment group. In that case, the effect of the new app version on a certain outcome, such as the average time spent on the app every day, is worth investigating. The issue is that not everyone in the treatment group will download the new version. In the treatment group, those who download the new version are different from those who do not, and then selection bias arises. To avoid that bias, the control group can be compared to all subjects in the treatment group, no matter whether a particular individual downloads the app. The estimated treatment effect is no doubt diluted because some individuals in the treatment group are not actually treated. Again, one solution is to treat the experiment as an observational study.

PROC DEEPPRICE exploits deep neural networks (DNNs) for causal inference. It applies deep learning to causal inference in a two-step semiparametric framework: the nonparametric step and the parametric step. In the nonparametric step, two DNNs are used to estimate the unknown functions in the treatment model and outcome model. In the parametric step, parameters of interest are estimated through the corresponding influence functions.

For more information about using DNNs, see *SAS Viya: Deep Learning Programming Guide*.

Estimating Unknown Functions in Treatment and Outcome Models

There are two main models in causal inference: the treatment model and the outcome model. Two approximate DNNs are constructed to estimate these two models. The DEEPPRICE procedure supports one continuous treatment and one continuous outcome.

Treatment Model

Without an Instrumental Variable

The treatment model estimates the treatment value conditional on some covariates:

$$\mathbb{E}[T|\mathbf{X}^{(1)} = \mathbf{x}^{(1)}] = \eta(\mathbf{x}^{(1)})$$

where T denotes the continuous treatment and $\mathbf{x}^{(1)}$ is a vector of the observed covariates. The superscript (1) in $\mathbf{X}^{(1)}$ denotes the covariates in Set 1, which are the covariates used in the treatment model. The function $\eta(\cdot)$ is one unknown function to be estimated by a DNN.

The DNN for the treatment model, which is a regression predictor, predicts the continuous treatment by using the covariates in Set 1. It trains the observations in the input data set. The DNN is a fully connected multilayer perceptron (MLP). The input layer of that DNN corresponds to $\mathbf{X}^{(1)}$, and the output layer corresponds to the observed treatment variable. The activation functions for the hidden layers are all rectified linear unit (ReLU) functions, and the activation function for the output layer is an identity function. The loss function is the sum of squared error function.

If no covariates are used to predict the treatment ($\mathbf{X}^{(1)} = \emptyset$), then no DNN needs to be trained for the treatment model. The treatment model is predicted by

$$\mathbb{E}[T|\mathbf{X}^{(1)} = \emptyset] = \mathbb{E}[T] = \eta(\emptyset)$$

The parameter $\eta(\emptyset)$ is estimated through the sample mean of the treatment variable, $\frac{1}{n} \sum_{i=1}^n t_i$, where t_i denotes the treatment value of the i th observation and n denotes the number of observations.

With an Instrumental Variable

The treatment model estimates the treatment value conditional on some covariates and the instrumental variable:

$$\mathbb{E}[T|\mathbf{X}^{(1)} = \mathbf{x}^{(1)}, Z = z] = \eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z$$

where T denotes the continuous treatment, Z denotes the instrumental variable, and $\mathbf{x}^{(1)}$ is a vector of the observed covariates. The superscript (1) in $\mathbf{X}^{(1)}$ denotes the covariates in Set 1, which are the covariates used in the treatment model. The functions $\eta_0(\cdot)$ and $\eta_1(\cdot)$ are two unknown functions to be estimated by a DNN.

The DNN for the treatment model, which is a regression predictor, predicts the continuous treatment by using the covariates in Set 1 and the instrumental variable. It trains the observations in the input data set. The DNN is a fully connected multilayer perceptron (MLP). The input layer of that DNN corresponds to $\mathbf{X}^{(1)}$. The activation functions for those hidden layers are rectified linear unit (ReLU) functions. After those hidden layers, there is a parameter layer with two neurons that correspond to $\eta_0(\mathbf{x}^{(1)})$ and $\eta_1(\mathbf{x}^{(1)})$. The activation functions for this parameter layer are linear functions. Subsequently, z is included, and $\eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z$ is calculated as the result of the final output layer, which corresponds to the observed treatment variable. The activation function for the output layer is an identity function. The loss function is the sum of squared error function.

If no covariates are used to predict the treatment ($\mathbf{X}^{(1)} = \emptyset$), then no DNN needs to be trained for the treatment model. The treatment model is predicted by

$$\mathbb{E}[T|\mathbf{X}^{(1)} = \emptyset, Z = z] = \mathbb{E}[T|Z = z] = \eta_0(\emptyset) + \eta_1(\emptyset)z$$

The parameters $\eta_0(\emptyset)$ and $\eta_1(\emptyset)$ are the intercept and slope coefficients, respectively, in the linear projection of the treatment T on the instrumental variable Z , and they are estimated through ordinary least squares (OLS) regression; that is,

$$\eta_1(\emptyset) = \frac{\sum_{i=1}^n z_i t_i}{\sum_{i=1}^n z_i^2}$$

$$\eta_0(\emptyset) = \frac{1}{n} \sum_{i=1}^n t_i - \frac{\eta_1(\emptyset)}{n} \sum_{i=1}^n z_i$$

where t_i denotes the treatment value of the i th observation, z_i denotes the instrumental variable value of the i th observation, and n denotes the number of observations.

Outcome Model

Without an Instrumental Variable

The outcome model describes how the outcome depends on the treatment and some covariates in Set 2, which are the covariates used in the outcome model. The covariates in Set 1 might differ from those in Set 2. The DEEPPRICE procedure supports only one continuous outcome variable, and the outcome model is as follows:

$$\mathbb{E}[Y|\mathbf{X}^{(2)} = \mathbf{x}^{(2)}, T = t] = G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t)$$

where Y is the outcome, t is the treatment, and $G(\cdot)$ is a known function. For example, $G(u)$ might be the identity function; that is, $G(u) = u$. The superscript (2) in $\mathbf{X}^{(2)}$ denotes the covariates in Set 2. In PROC DEEPPRICE, $G(\cdot)$ is the identity function. The functions $\alpha(\cdot)$ and $\beta(\cdot)$ are unknown functions to be estimated by a DNN.

The DNN for the outcome model, which is a regression predictor, predicts the outcome by using the covariates in Set 2 and the treatment variable. It trains the observations in the input data set. The input layer of that DNN corresponds to $\mathbf{X}^{(2)}$. After the hidden layers, there is a parameter layer with two neurons that correspond to $\alpha(\mathbf{x}^{(2)})$ and $\beta(\mathbf{x}^{(2)})$. The activation functions for this parameter layer are linear functions. Subsequently, t is included, and $G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t)$ is calculated as the result of the final output layer. $G(\cdot)$ is set to the identity function in PROC DEEPPRICE, so $\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t$ is output as the prediction.

If no covariates are used to predict the outcome ($\mathbf{X}^{(2)} = \emptyset$), then no DNN needs to be trained for the outcome model. The outcome model is predicted by

$$\mathbb{E}[Y|\mathbf{X}^{(2)} = \emptyset, T = t] = \mathbb{E}[Y|T = t]$$

Moreover, when $\mathbf{X}^{(2)} = \emptyset$, the functions $\alpha(\mathbf{x}^{(2)})$ and $\beta(\mathbf{x}^{(2)})$ are constants and are computed by

$$\beta(\emptyset) = \frac{\sum_{i=1}^n t_i y_i}{\sum_{i=1}^n t_i^2}$$

$$\alpha(\emptyset) = \frac{1}{n} \sum_{i=1}^n y_i - \frac{\beta(\emptyset)}{n} \sum_{i=1}^n t_i$$

where y_i denotes the outcome value of the i th observation, t_i denotes the treatment value of the i th observation, and n denotes the number of observations.

With an Instrumental Variable

The outcome model describes how the outcome depends on the treatment and some covariates in Set 2, which are the covariates used in the outcome model. The covariates in Set 1 might differ from those in Set 2. The DEEPPRICE procedure supports only one continuous outcome variable, and the outcome model is as follows:

$$\mathbb{E}[Y | \mathbf{X}^{(2)} = \mathbf{x}^{(2)}, T = t] = G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t)$$

where Y is the outcome, t is the treatment, and $G(\cdot)$ is a known function. For example, $G(u)$ might be the identity function; that is, $G(u) = u$. The superscript (2) in $\mathbf{X}^{(2)}$ denotes the covariates in Set 2. In PROC DEEPPRICE, $G(\cdot)$ is the identity function. The functions $\alpha(\cdot)$ and $\beta(\cdot)$ are unknown functions to be estimated by a DNN. However, because the treatment variable is endogenous and the instrumental variable is required, the two-stage regression is applied: first, the treatment variable is predicted by using the covariates in Set 1 and the instrumental variable (see the section “With an Instrumental Variable” on page 983); then the treatment variable t is replaced by its prediction $\hat{t}$. That is,

$$\mathbb{E}[Y | \mathbf{X}^{(2)} = \mathbf{x}^{(2)}, T = \hat{t}] = G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})$$

The DNN for the outcome model, which is a regression predictor, predicts the outcome by using the covariates in Set 2 and the treatment variable. It trains the observations in the input data set. The input layer of that DNN corresponds to $\mathbf{X}^{(2)}$. After the hidden layers, there is a parameter layer with two neurons that correspond to $\alpha(\mathbf{x}^{(2)})$ and $\beta(\mathbf{x}^{(2)})$. The activation functions for this parameter layer are linear functions. Subsequently, t is included, and $G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})$ is calculated as the result of the final output layer. $G(\cdot)$ is set to the identity function in PROC DEEPPRICE, so $\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t}$ is output as the prediction.

If no covariates are used to predict the outcome ($\mathbf{X}^{(2)} = \emptyset$), then no DNN needs to be trained for the outcome model. The outcome model is predicted by

$$\mathbb{E}[Y | \mathbf{X}^{(2)} = \emptyset, T = \hat{t}] = \mathbb{E}[Y | T = \hat{t}]$$

Moreover, when $\mathbf{X}^{(2)} = \emptyset$, the functions $\alpha(\mathbf{x}^{(2)})$ and $\beta(\mathbf{x}^{(2)})$ are constants and are computed by

$$\begin{aligned}\beta(\emptyset) &= \frac{\sum_{i=1}^n \hat{t}_i y_i}{\sum_{i=1}^n \hat{t}_i^2} \\ \alpha(\emptyset) &= \frac{1}{n} \sum_{i=1}^n y_i - \frac{\beta(\emptyset)}{n} \sum_{i=1}^n \hat{t}_i\end{aligned}$$

where y_i denotes the outcome value of the i th observation, $\hat{t}_i$ denotes the predicted treatment value of the i th observation, and n denotes the number of observations.

Estimating Parameters of Interest

A parameter of interest can be defined through the following structural model:

$$\theta_0 = \mathbb{E}[H(y, \mathbf{X}, T, \alpha(\mathbf{X}^{(2)}), \beta(\mathbf{X}^{(2)}); t^*)]$$

where θ_0 is a certain parameter of interest, $H(\cdot)$ is a known function that corresponds to θ_0 , $\mathbf{X}$ is the union of $\mathbf{X}^{(1)}$ and $\mathbf{X}^{(2)}$, and t^* is a fixed treatment value of interest. To obtain the expected value, a naive approach is to plug the DNN-based estimates $\hat{\alpha}(\mathbf{x}^{(2)})$ and $\hat{\beta}(\mathbf{x}^{(2)})$ into the structural model of θ_0 . This estimator, which is known as the plug-in estimator, is given by

$$\hat{\theta}_0 = \frac{1}{n} \sum_{i=1}^n H(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}); t^*)$$

where n is the sample size and the subscript i stands for the i th observation. For example, if T is continuous, then θ_0 is the estimator of the average slope when $H(y, \mathbf{X}, T, \alpha(\mathbf{X}^{(2)}), \beta(\mathbf{X}^{(2)}); t^*) = \beta(\mathbf{X}^{(2)})$. Then the plug-in estimator of the average slope is given by

$$\hat{\theta}_0 = \frac{1}{n} \sum_{i=1}^n \hat{\beta}(\mathbf{x}_i^{(2)})$$

This estimator might be highly biased, because the covariates might have both effects on the outcome and treatment assignment. One way to solve this problem is to use an estimator based on the influence function. With one treatment variable, Farrell, Liang, and Misra (2020) propose the following influence function for the case in which there is no instrumental variable:

$$\begin{aligned} \psi(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) + \nabla \mathbf{H}(\mathbf{x}) \Lambda(\mathbf{x})^{-1} (1, t)' (y - G(\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})t)) \end{aligned}$$

where $\nabla \mathbf{H}(\mathbf{x}) = (\frac{\partial H}{\partial \alpha}, \frac{\partial H}{\partial \beta})$, $\Lambda(\mathbf{x}) = \mathbb{E}[\dot{G}(1, T)'(1, T) | \mathbf{X} = \mathbf{x}]$, and $\dot{G} = [dG(u)/du]_{u=\alpha(\mathbf{x}^{(2)})+\beta(\mathbf{x}^{(2)})t}$.

Because PROC DEEPPRICE sets $G(\cdot)$ to the identity function and supports one continuous treatment, $\dot{G} = 1$ and the inverse of $\Lambda(\mathbf{x})$ can be computed directly. The preceding influence function can be simplified as follows:

$$\begin{aligned} \psi(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) \\ + \frac{\dot{H}_\alpha(\mathbf{x})(\lambda_2(\mathbf{x}) - \lambda_1(\mathbf{x})t) + \dot{H}_\beta(\mathbf{x})(\lambda_0(\mathbf{x})t - \lambda_1(\mathbf{x}))}{\lambda_2(\mathbf{x})\lambda_0(\mathbf{x}) - \lambda_1(\mathbf{x})^2} (y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t) \end{aligned}$$

where $\lambda_j(\mathbf{x}) = \mathbb{E}[T^j | \mathbf{X} = \mathbf{x}]$, $j = 0, 1, 2$; $\dot{H}_\alpha(\mathbf{x}) = \frac{\partial H}{\partial \alpha}$; and $\dot{H}_\beta(\mathbf{x}) = \frac{\partial H}{\partial \beta}$. When the treatment variable is continuous,

$$\begin{aligned} \lambda_0(\mathbf{x}) &= \mathbb{E}[T^0 | \mathbf{X} = \mathbf{x}] = 1 \\ \lambda_1(\mathbf{x}) &= \mathbb{E}[T^1 | \mathbf{X} = \mathbf{x}] = \eta_0(\mathbf{x}) \\ \lambda_2(\mathbf{x}) &= \mathbb{E}[T^2 | \mathbf{X} = \mathbf{x}] = \sigma_t^2 + \eta_0^2(\mathbf{x}) \end{aligned}$$

where σ_t^2 is the variance of the treatment (regression) model and is used to approximate the conditional variance.

The estimator for the parameter of interest is given by

$$\hat{\theta}_0 = \frac{\sum_{i=1}^n \psi(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

This estimator is doubly robust, which means that the estimator is unbiased when either the treatment model or the outcome model is correct. The standard error of $\hat{\theta}_0$ is

$$\sigma(\hat{\theta}_0) = \frac{\sqrt{\sum_{i=1}^n (\psi(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*) - \hat{\theta}_0)^2}}{n}$$

For the case in which there is an instrumental variable, the influence function becomes

$$\begin{aligned} & \psi(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z, t^*) \\ &= H(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z, t^*) \\ &+ \nabla \mathbf{H}(\mathbf{x}) \begin{pmatrix} \Lambda_Z^{-1}(1, z)'(y - (\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})) \\ \Lambda_Z^{-1}(1, z)'(t - (\eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z)) \end{pmatrix} \end{aligned}$$

where $\nabla \mathbf{H}(\mathbf{x}) = (\frac{\partial H}{\partial \tilde{\alpha}}, \frac{\partial H}{\partial \tilde{\beta}}, \frac{\partial H}{\partial \eta_0}, \frac{\partial H}{\partial \eta_1})$; $\Lambda_Z = \begin{pmatrix} 1 & \mu_z \\ \mu_z & \sigma_z^2 + \mu_z^2 \end{pmatrix}$ if you assume that z is independent of all covariates in $\mathbf{x}$; and μ_z and σ_z^2 are the sample mean and variance of z , respectively. Also, $\tilde{\alpha}(\mathbf{x}) = \alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\eta_0(\mathbf{x}^{(1)})$ and $\tilde{\beta}(\mathbf{x}) = \beta(\mathbf{x}^{(2)})\eta_1(\mathbf{x}^{(1)})$; and $\hat{t} = \eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z$.

For ease of presentation, the missing value is not indicated in the preceding equations. In practice, if $\psi(\cdot)$ is a missing value for $i = 1, \dots, n$, then it is ignored in both the denominator and numerator of the equation of $\hat{\theta}_0$ or $\sigma(\hat{\theta}_0)$. This policy is applied to all parameters of interest that are described in the sections “[Average Intercept and Slope Parameters](#)” on page 987 and “[Policy Evaluation and Comparison](#)” on page 990.

Next, the calculation of some common parameters of interest is introduced.

Average Intercept and Slope Parameters

There are two types of average parameters:

- $\mathbb{E}[\alpha(X)]$, the average intercept
- $\mathbb{E}[\beta(X)]$, the average slope

The two average parameters are estimated as follows:

- Average intercept, $\mathbb{E}[\alpha(X)]$

- Without an instrumental variable:

The influence function, denoted by $\psi_\alpha(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*)$, is defined by setting $H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = H_\alpha(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = \alpha(\mathbf{x}^{(2)})$ in the influence

function. When $G(\cdot)$ is set to the identity function and the treatment is continuous, the influence function is simplified as

$$\begin{aligned}\psi_\alpha(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ &= \alpha(\mathbf{x}^{(2)}) + \frac{\lambda_2(\mathbf{x}) - \lambda_1(\mathbf{x})t}{\lambda_2(\mathbf{x})\lambda_0(\mathbf{x}) - \lambda_1(\mathbf{x})^2} (y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t) \\ &= \alpha(\mathbf{x}^{(2)}) + \frac{\sigma_t^2 - \eta_0(\mathbf{x}^{(1)})(t - \eta_0(\mathbf{x}^{(1)}))}{\sigma_t^2} (y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t)\end{aligned}$$

Then $\mathbb{E}[\alpha(X)]$ is estimated by

$$\mathbb{E}[\alpha(X)] = \frac{\sum_{i=1}^n \psi_\alpha(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

– With an instrumental variable:

The influence function, denoted by $\psi_\alpha(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z, t^*)$, is defined by setting

$$\begin{aligned}H(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z; t^*) \\ &= H_\alpha(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z; t^*) \\ &= \alpha(\mathbf{x}^{(2)}) = \tilde{\alpha}(\mathbf{x}) - \frac{\tilde{\beta}(\mathbf{x})}{\eta_1(\mathbf{x}^{(1)})} \eta_0(\mathbf{x}^{(1)})\end{aligned}$$

in the influence function. When $G(\cdot)$ is set to the identity function and the treatment is continuous, the influence function is simplified as

$$\begin{aligned}\psi_\alpha(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z, t^*) \\ &= \tilde{\alpha}(\mathbf{x}) - \frac{\tilde{\beta}(\mathbf{x})}{\eta_1(\mathbf{x}^{(1)})} \eta_0(\mathbf{x}^{(1)}) + \nabla \mathbf{H}_\alpha(\mathbf{x}) \begin{pmatrix} \Lambda_Z^{-1}(1, z)'(y - (\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})) \\ \Lambda_Z^{-1}(1, z)'(t - (\eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z)) \end{pmatrix} \\ &= \tilde{\alpha}(\mathbf{x}) - \frac{\tilde{\beta}(\mathbf{x})}{\eta_1(\mathbf{x}^{(1)})} \eta_0(\mathbf{x}^{(1)}) \\ &\quad + (1)(\sigma_z^{-2}(\sigma_z^2 + \mu_z^2 - \mu_z z))(y - (\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})) \\ &\quad + (-\frac{\eta_0(\mathbf{x}^{(1)})}{\eta_1(\mathbf{x}^{(1)})})(-\mu_z + z)(y - (\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})) \\ &\quad + (-\frac{\tilde{\beta}(\mathbf{x})}{\eta_1(\mathbf{x}^{(1)})})(\sigma_z^{-2}(\sigma_z^2 + \mu_z^2 - \mu_z z))(t - (\eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z)) \\ &\quad + (-\frac{\tilde{\beta}(\mathbf{x})}{\eta_1^2(\mathbf{x}^{(1)})} \eta_0(\mathbf{x}^{(1)}))(-\mu_z + z)(t - (\eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z))\end{aligned}$$

where $\nabla \mathbf{H}_\alpha(\mathbf{x}) = (1, -\frac{\eta_0(\mathbf{x}^{(1)})}{\eta_1(\mathbf{x}^{(1)})}, -\frac{\tilde{\beta}(\mathbf{x})}{\eta_1(\mathbf{x}^{(1)})}, \frac{\tilde{\beta}(\mathbf{x})}{\eta_1^2(\mathbf{x}^{(1)})} \eta_0(\mathbf{x}^{(1)}))$ and $\mu_z = \frac{\sum_{i=1}^n z_i}{n}$. Then $\mathbb{E}[\alpha(X)]$ is estimated by

$$\mathbb{E}[\alpha(X)] = \frac{\sum_{i=1}^n \psi_\alpha(y_i, \mathbf{x}_i, t_i, z_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \hat{\eta}_0(\mathbf{x}^{(1)}), \hat{\eta}_1(\mathbf{x}^{(1)}), \Lambda_Z, t^*)}{n}$$

where $\hat{\alpha}(\mathbf{x}_i^{(2)}) = \hat{\alpha}(\mathbf{x}^{(2)}) + \hat{\beta}(\mathbf{x}^{(2)})\hat{\eta}_0(\mathbf{x}^{(1)})$ and $\hat{\beta}(\mathbf{x}_i^{(2)}) = \hat{\beta}(\mathbf{x}^{(2)})\hat{\eta}_1(\mathbf{x}^{(1)})$.

• Average slope, $\mathbb{E}[\beta(X)]$

– Without an instrumental variable:

The influence function, denoted by $\psi_\beta(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*)$, is defined by setting $H(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = H_\beta(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}); t^*) = \beta(\mathbf{x}^{(2)})$ in the influence function. When $G(\cdot)$ is set to the identity function and the treatment is continuous, the influence function is simplified as

$$\begin{aligned} \psi_\beta(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ &= \beta(\mathbf{x}^{(2)}) + \frac{\lambda_0(\mathbf{x})t - \lambda_1(\mathbf{x})}{\lambda_2(\mathbf{x})\lambda_0(\mathbf{x}) - \lambda_1(\mathbf{x})^2} (y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t) \\ &= \beta(\mathbf{x}^{(2)}) + \frac{t - \eta_0(\mathbf{x}^{(1)})}{\sigma_t^2} (y - \alpha(\mathbf{x}^{(2)}) - \beta(\mathbf{x}^{(2)})t) \end{aligned}$$

Then $\mathbb{E}[\beta(X)]$ is estimated by

$$\mathbb{E}[\beta(X)] = \frac{\sum_{i=1}^n \psi_\beta(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

– With an instrumental variable:

The influence function, denoted by $\psi_\beta(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z, t^*)$, is defined by setting

$$\begin{aligned} H(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z; t^*) \\ &= H_\beta(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z; t^*) \\ &= \beta(\mathbf{x}^{(2)}) = \frac{\tilde{\beta}(\mathbf{x})}{\eta_1(\mathbf{x}^{(1)})} \end{aligned}$$

in the influence function. When $G(\cdot)$ is set to the identity function and the treatment is continuous, the influence function is simplified as

$$\begin{aligned} \psi_\beta(y, \mathbf{x}, t, z, \tilde{\alpha}(\mathbf{x}), \tilde{\beta}(\mathbf{x}), \eta_0(\mathbf{x}^{(1)}), \eta_1(\mathbf{x}^{(1)}), \Lambda_Z, t^*) \\ &= \frac{\tilde{\beta}(\mathbf{x})}{\eta_1(\mathbf{x}^{(1)})} + \nabla \mathbf{H}_\beta(\mathbf{x}) \begin{pmatrix} \Lambda_Z^{-1}(1, z)'(y - (\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})) \\ \Lambda_Z^{-1}(1, z)'(t - (\eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z)) \end{pmatrix} \\ &= \frac{\tilde{\beta}(\mathbf{x})}{\eta_1(\mathbf{x}^{(1)})} \\ &\quad + (0)(\sigma_z^{-2}(\sigma_z^2 + \mu_z^2 - \mu_z z))(y - (\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})) \\ &\quad + \left(\frac{1}{\eta_1(\mathbf{x}^{(1)})}\right)(-\mu_z + z)(y - (\alpha(\mathbf{x}^{(2)}) + \beta(\mathbf{x}^{(2)})\hat{t})) \\ &\quad + (0)(\sigma_z^{-2}(\sigma_z^2 + \mu_z^2 - \mu_z z))(t - (\eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z)) \\ &\quad + \left(-\frac{\tilde{\beta}(\mathbf{x})}{\eta_1^2(\mathbf{x}^{(1)})}\right)(-\mu_z + z)(t - (\eta_0(\mathbf{x}^{(1)}) + \eta_1(\mathbf{x}^{(1)})z)) \end{aligned}$$

where $\nabla \mathbf{H}_\beta(\mathbf{x}) = (0, \frac{1}{\eta_1(\mathbf{x}^{(1)})}, 0, -\frac{\tilde{\beta}(\mathbf{x})}{\eta_1^2(\mathbf{x}^{(1)})})$ and $\mu_z = \frac{\sum_{i=1}^n z_i}{n}$. Then $\mathbb{E}[\beta(X)]$ is estimated by

$$\mathbb{E}[\beta(X)] = \frac{\sum_{i=1}^n \psi_\beta(y_i, \mathbf{x}_i, t_i, z_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \hat{\eta}_0(\mathbf{x}^{(1)}), \hat{\eta}_1(\mathbf{x}^{(1)}), \Lambda_Z, t^*)}{n}$$

where $\hat{\alpha}(\mathbf{x}_i^{(2)}) = \hat{\alpha}(\mathbf{x}^{(2)}) + \hat{\beta}(\mathbf{x}^{(2)})\hat{\eta}_0(\mathbf{x}^{(1)})$ and $\hat{\beta}(\mathbf{x}_i^{(2)}) = \hat{\beta}(\mathbf{x}^{(2)})\hat{\eta}_1(\mathbf{x}^{(1)})$.

Policy Evaluation and Comparison

First, a policy is defined to be a rule that assigns a given set of information, represented by the covariates $\mathbf{X}$, to a treatment value; that is, a policy is a known map function that is defined by

$$s(\mathbf{X}) : \mathbf{X} \rightarrow \mathbb{R}$$

The expected utility of a policy, denoted by $\pi(s)$, is given by

$$\pi(s) = \mathbb{E}[\alpha(\mathbf{X}) + \beta(\mathbf{X})s(\mathbf{X})]$$

To estimate $\pi(s)$, the influence function is replaced by

$$\begin{aligned} \psi_{\pi(s)}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = s(\mathbf{x})\psi_{\beta}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) + \psi_{\alpha}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \end{aligned}$$

Then $\pi(s)$ is estimated by

$$\pi(s) = \frac{\sum_{i=1}^n \psi_{\pi(s)}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

The difference in expected utility of two policies is defined by

$$\pi(s_1, s_0) = \mathbb{E}[(s_1(\mathbf{X}) - s_0(\mathbf{X}))\beta(\mathbf{X})]$$

To estimate $\pi(s_1, s_0)$, the influence function is replaced by

$$\begin{aligned} \psi_{\pi(s_1, s_0)}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \\ = (s_1(\mathbf{x}) - s_0(\mathbf{x}))\psi_{\beta}(y, \mathbf{x}, t, \alpha(\mathbf{x}^{(2)}), \beta(\mathbf{x}^{(2)}), \Lambda(\mathbf{x}), t^*) \end{aligned}$$

Then $\pi(s_1, s_0)$ is estimated by

$$\pi(s_1, s_0) = \frac{\sum_{i=1}^n \psi_{\pi(s_1, s_0)}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)}{n}$$

Missing Values

Treatment Variable

In the DNN training stage of the treatment model, the missing treatment values are replaced by imputed values. (For more information, see the [TARGETMISSING=](#) option.) In the inference stage, the observations that have missing treatment values have missing values for residuals of the treatment model and for any estimates that need such residuals.

Outcome Variable

In the DNN training stage of the outcome model, the missing outcome values are replaced by imputed values. (For more information, see the [TARGETMISSING=](#) option.) In the inference stage, the observations that have missing outcome values have missing values for residuals of the outcome model and for any estimates that need such residuals.

Covariate

In the DNN training stage of the treatment model or the outcome model, the missing covariate values are replaced by imputed values. (For more information, see the [MISSING=](#) option.) In the inference stage, the observations that have missing covariates are ignored.

Data Table Output

The DEEPPRICE procedure can create the data tables that you specify in the [OUT=](#) and [OUTDETAILS=](#) options in the [INFER](#) statement. The information in each table is described in the following sections.

OUT= Data Table Generated by the INFER Statement

The estimate and the corresponding standard error of each parameter of interest are displayed in this table. For more information about the parameters of interest, see the section “[Average Intercept and Slope Parameters](#)” on page 987. The output data table contains the following variables:

NOBS	number of observations in the input data table
_N_AVERAGE_INTERCEPT_	number of observations in the estimation of $\mathbb{E}[\alpha(X)]$
_EST_AVERAGE_INTERCEPT_	estimate of $\mathbb{E}[\alpha(X)]$
_STD_AVERAGE_INTERCEPT_	standard error of $\mathbb{E}[\alpha(X)]$
_N_AVERAGE_SLOPE_	number of observations in the estimation of $\mathbb{E}[\beta(X)]$
_EST_AVERAGE_SLOPE_	estimate of $\mathbb{E}[\beta(X)]$
_STD_AVERAGE_SLOPE_	standard error of $\mathbb{E}[\beta(X)]$
_N_POLICY_POLICY-VAR_	number of observations in the estimation of $\pi(s)$, where s is the value of a policy variable, POLICY-VAR , that you specify in the POLICY= option in the INFER statement. The estimate is displayed for each policy variable.

<code>_EST_POLICY_POLICY-VAR_</code>	estimate of $\pi(s)$, where s is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>POLICY=</code> option in the <code>INFER</code> statement. The estimate is displayed for each policy variable.
<code>_STD_POLICY_POLICY-VAR_</code>	standard error of $\pi(s)$, where s is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>POLICY=</code> option in the <code>INFER</code> statement. The standard error is displayed for each policy variable.
<code>_N_POLICY_COMPARISON_POLICY-VAR_BASE-VAR_</code>	number of observations in the estimation of $\pi(s_c, s_b)$, where s_c is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>COMPARE=</code> suboption of the <code>POLICYCOMPARISON=</code> option in the <code>INFER</code> statement and s_b is the value of a policy variable, <code>BASE-VAR</code> , that you specify in the <code>BASE=</code> suboption of the <code>POLICYCOMPARISON=</code> option in the <code>INFER</code> statement. The estimate is displayed for each combination of s_c and s_b .
<code>_EST_POLICY_COMPARISON_POLICY-VAR_BASE-VAR_</code>	estimate of $\pi(s_c, s_b)$, where s_c is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>COMPARE=</code> suboption of the <code>POLICYCOMPARISON=</code> option in the <code>INFER</code> statement and s_b is the value of a policy variable, <code>BASE-VAR</code> , that you specify in the <code>BASE=</code> suboption of the <code>POLICYCOMPARISON=</code> option in the <code>INFER</code> statement. The estimate is displayed for each combination of s_c and s_b .
<code>_STD_POLICY_COMPARISON_POLICY-VAR_BASE-VAR_</code>	standard error of $\pi(s_c, s_b)$, where s_c is the value of a policy variable, <code>POLICY-VAR</code> , that you specify in the <code>COMPARE=</code> suboption of the <code>POLICYCOMPARISON=</code> option in the <code>INFER</code> statement and s_b is the value of a policy variable, <code>BASE-VAR</code> , that you specify in the <code>BASE=</code> suboption of the <code>POLICYCOMPARISON=</code> option in the <code>INFER</code> statement. The standard error is displayed for each combination of s_c and s_b .

OUTDETAILS= Data Table Generated by the INFER Statement

The output data first consist of the variables in the input data table and the details in the estimation, including $\alpha(\cdot)$, $\beta(\cdot)$, $p(\cdot)$, the residual, and the influence functions for each observation in the input data table. For more information about the influence functions for parameters of interest, the treatment prediction model, and the outcome prediction model, see the sections “[Estimating Unknown Functions in Treatment and Outcome Models](#)” on page 983 and “[Estimating Parameters of Interest](#)” on page 986. In the following list, i denotes the observation index. Moreover, the output data table contains the following variables in addition to the input variables:

<code>_THAT_</code>	value of $\mathbb{E}[T \mathbf{X}^{(1)} = \mathbf{x}_i^{(1)}] = \hat{\eta}_0(\mathbf{x}_i^{(1)})$ for each observation when there is no instrumental variable, and value of $\mathbb{E}[T \mathbf{X}^{(1)} = \mathbf{x}_i^{(1)}, Z = z_i] = \hat{\eta}_0(\mathbf{x}_i^{(1)}) + \hat{\eta}_1(\mathbf{x}_i^{(1)})z_i$ for each observation when there is the instrumental variable Z
<code>_YHAT_</code>	value of $\mathbb{E}[Y \mathbf{X}^{(2)} = \mathbf{x}_i^{(2)}, T = t_i] = \hat{\alpha}(\mathbf{x}_i^{(2)}) + \hat{\beta}(\mathbf{x}_i^{(2)})t_i$ for each observation when there is no instrumental variable, and value of $\mathbb{E}[Y \mathbf{X}^{(2)} = \mathbf{x}_i^{(2)}, T = \hat{t}_i] = \hat{\alpha}(\mathbf{x}_i^{(2)}) + \hat{\beta}(\mathbf{x}_i^{(2)})\hat{t}_i$ for each observation, where $\hat{t}_i = \hat{\eta}_0(\mathbf{x}_i^{(1)}) + \hat{\eta}_1(\mathbf{x}_i^{(1)})z_i$, when there is the instrumental variable Z

<code>_SIGMAT_</code>	standard error of the residuals in the treatment model, $\sigma_t = \sqrt{\frac{\sum_{i=1}^N (t_i - \hat{\eta}_0(\mathbf{x}_i^{(1)}))^2}{n}}$ when there is no instrumental variable or $\sigma_t = \sqrt{\frac{\sum_{i=1}^N (t_i - (\hat{\eta}_0(\mathbf{x}_i^{(1)}) + \hat{\eta}_1(\mathbf{x}_i^{(1)})z_i))^2}{n}}$ when there is the instrumental variable Z , where n the total number of observations. The value is identical for all observations.
<code>_epst_</code>	value of the residual in the treatment model for each observation, $\eta_i = t_i - \hat{\eta}_0(\mathbf{x}_i^{(1)})$ when there is no instrumental variable, and $\eta_i = t_i - (\hat{\eta}_0(\mathbf{x}_i^{(1)}) + \hat{\eta}_1(\mathbf{x}_i^{(1)})z_i)$ when there is the instrumental variable Z
<code>_epsy_</code>	value of the residual in the outcome model, $\varepsilon_i = y_i - (\hat{\alpha}(\mathbf{x}_i^{(2)}) + \hat{\beta}(\mathbf{x}_i^{(2)})t_i)$ when there is no instrumental variable, and $\varepsilon_i = y_i - (\hat{\alpha}(\mathbf{x}_i^{(2)}) + \hat{\beta}(\mathbf{x}_i^{(2)})\hat{t}_i)$ when there is the instrumental variable Z , where $\hat{t}_i = \hat{\eta}_0(\mathbf{x}_i^{(1)}) + \hat{\eta}_1(\mathbf{x}_i^{(1)})z_i$
<code>_eta0_</code>	value of $\hat{\eta}_0(\mathbf{x}_i^{(1)})$ for each observation. This variable is available only when there is the instrumental variable Z .
<code>_eta1_</code>	value of $\hat{\eta}_1(\mathbf{x}_i^{(1)})$ for each observation. This variable is available only when there is the instrumental variable Z .
<code>_alpha_</code>	value of $\hat{\alpha}(\mathbf{x}_i^{(2)})$ for each observation
<code>_beta_</code>	value of $\hat{\beta}(\mathbf{x}_i^{(2)})$ for each observation
<code>_alphatilde_</code>	value of $\hat{\tilde{\alpha}}(\mathbf{x})$ for each observation. This variable is available only when there is the instrumental variable Z .
<code>_betatilde_</code>	value of $\hat{\tilde{\beta}}(\mathbf{x})$ for each observation. This variable is available only when there is the instrumental variable Z .
<code>_if_alpha_</code>	value of $\psi_\alpha(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$ for each observation when there is no instrumental variable, and value of $\psi_\alpha(y_i, \mathbf{x}_i, t_i, z_i, \hat{\tilde{\alpha}}_i(\mathbf{x}_i), \hat{\tilde{\beta}}_i(\mathbf{x}_i), \hat{\eta}_0(\mathbf{x}_i^{(1)}), \hat{\eta}_1(\mathbf{x}_i^{(1)}), \Lambda_Z, t^*)$ for each observation when there is the instrumental variable Z
<code>_if_beta_</code>	value of $\psi_\beta(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$ for each observation when there is no instrumental variable, and value of $\psi_\beta(y_i, \mathbf{x}_i, t_i, z_i, \hat{\tilde{\alpha}}_i(\mathbf{x}_i), \hat{\tilde{\beta}}_i(\mathbf{x}_i), \hat{\eta}_0(\mathbf{x}_i^{(1)}), \hat{\eta}_1(\mathbf{x}_i^{(1)}), \Lambda_Z, t^*)$ for each observation when there is the instrumental variable Z
<code>_policy_policy-var_</code>	value of $\psi_{\pi(s)}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$, where s is the value of a policy variable that you specify in the POLICY= option in the INFER statement. The estimate is displayed for each policy variable.
<code>_policy_comparison_policy-var_base-var_</code>	value of $\psi_{\pi(s_c, s_b)}(y_i, \mathbf{x}_i, t_i, \hat{\alpha}(\mathbf{x}_i^{(2)}), \hat{\beta}(\mathbf{x}_i^{(2)}), \Lambda(\mathbf{x}_i), t^*)$, where s_c is the value of a policy variable, POLICY-VAR, that you specify in the COMPARE= suboption of the POLICYCOMPARISON= option in the INFER statement and s_b is the value of a policy variable, BASE-VAR, that you specify in the BASE= suboption of the POLICYCOMPARISON= option in the INFER statement. The estimate is displayed for each combination of s_c and s_b .

ODS Table Names

The DEEPPRICE procedure assigns a name to each table that it creates. You can use these names to refer to the tables when you use the Output Delivery System (ODS) to select tables and create output data tables.

These names are listed in Table 16.2.

Table 16.2 ODS Tables Produced in the DEEPPRICE Procedure

ODS Table Name	Description	Option
NObs	Number of observations	Default
ParameterEstimates	Parameter estimates	Default
PolicyComparison	Policy comparison	POLICYCOMPARISON=
PolicyEvaluation	Policy evaluation	POLICY=

Examples: DEEPPRICE Procedure

Example 16.1: Personalized Customer Segmentation for an Online Media Company

This example illustrates how a music subscription service from an online media company can offer targeted discounts through a personalized pricing plan based on many features that the company observes about its customers to encourage them to buy more songs.

The data set is provided by the Microsoft research project ALICE and is available at https://econmldata.azurewebsites.net/datasets/Pricing/pricing_sample.csv. The data set has 10,000 simulated observations that represent users' personal characteristics, such as age, log income, and account age, as well as users' online behavior history, including previous purchases and previous online time per week. The treatment variable, price, is the price that the customer was exposed to during the discount season. A value of 1 indicates that no discount is given, and a value less than 1 indicates that a discount is applied. The outcome variable, demand, is the number of songs that the customer purchased during the discount season. Table 16.3 shows the names of the variables that are used in the model, their types, and their definitions. The type can be T (treatment), Y (outcome), $x^{(1)}$ (a covariate in the treatment model), and/or $x^{(2)}$ (a covariate in the outcome model).

Table 16.3 Model Variables

Name	Type	Details
account_age	$x^{(1)}$, $x^{(2)}$	User's account age
age	$x^{(1)}$, $x^{(2)}$	User's age
avg_hours	$x^{(1)}$, $x^{(2)}$	Average number of hours per week user was online in the past
days_visited	$x^{(1)}$, $x^{(2)}$	Average number of days per week user visited website in the past
friend_count	$x^{(1)}$, $x^{(2)}$	Number of friends user connected to in account
has_membership	$x^{(1)}$, $x^{(2)}$	Whether user has membership

Table 16.3 *continued*

Name	Type	Details
is_US	$x^{(1)}, x^{(2)}$	Whether user accesses website from the United States
songs_purchased	$x^{(1)}, x^{(2)}$	Average number of songs user purchased per week in the past
income	$x^{(1)}, x^{(2)}$	User's income
price	T	Price that user was exposed to during the discount season (baseline price * small discount)
demand	Y	Number of songs user purchased during the discount season

This example has three goals:

1. Estimate the parameters of interest, such as average partial effect of price on demand.
2. Estimate the price elasticity of demand for songs, and assess its variation with users' characteristics.
3. Conduct a policy analysis that aims at increasing revenue despite decreasing the price for some users.

Assuming that you have downloaded the data set `pricing_sample` in your session that is associated with the `mylib` libref, the following statements create an ID variable that has a unique value for each observation in the data table `new_pricing_sample`:

```
data mylib.new_pricing_sample;
  set mylib.pricing_sample;
  id = put(_threadid_,8.) || '_' || Put(_n_,8.); * ID variable;
run;
```

The first step in policy evaluation and policy optimization is to estimate the effect of the treatment and to save to a specified output data table the details of the estimation, including $\alpha(\cdot)$ and $\beta(\cdot)$, the residual, and the influence functions for each unit. You can do this by using the following statements, which also store the estimated treatment model information, $\alpha(\cdot)$ information, and $\beta(\cdot)$ information in the data tables `outtdata`, `outadata`, and `outbdata`, respectively, to be used later for policy evaluation:

```
/*--- Estimate the treatment effect and save the estimation details ---*/
proc deepprice data=mylib.new_pricing_sample;
  id id;
  tmodel price = account_age age avg_hours days_visited friends_count
    has_membership is_US songs_purchased income /
    dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=500 regL1=0.0001 maxEpochs=32000
    algorithm=adam) nthreads=20 seed=12345 recordseed=67890));
  model demand = account_age age avg_hours days_visited friends_count
    has_membership is_US songs_purchased income /
    dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=500 regL1=0.001 maxEpochs=32000
    algorithm=adam) nthreads=20 seed=12345 recordseed=67890));
  infer out=mylib.oest outdetails=mylib.odetails;
  score outt=mylib.outtdata outa=mylib.outadata outb=mylib.outbdata;
run;
```

For this example, the same covariates are used in both the treatment model and the outcome model. The model estimation details are saved in the output data table `odetails`.

The estimation results are shown in [Output 16.1.1](#). The estimate of the average partial effect of price on demand is negative, as expected, and statistically significant.

Output 16.1.1 Parameters of Interest
The DEEPPRICE Procedure

Parameter	Parameter Estimates					Z	Pr > Z
	Estimate	Standard Error	95% Confidence Limits				
Average Intercept	23.629698	0.215064	23.208180	24.051217	109.87	<.0001	
Average Slope	-9.235895	0.244447	-9.715001	-8.756788	-37.78	<.0001	

Another parameter of interest is the price elasticity of demand, which measures the degree of sensitivity of demand to price. In general, demand decreases when price increases for most products, but the amount by which demand decreases is greater for some products than for others. Price elasticity measures the percentage change in demand when there is a 1% increase in price. In this example, a price elasticity of -3 means that a 1% increase in price will lead to a 3% decrease in the number of songs purchased.

Heterogeneity in price sensitivity is introduced among users by allowing the price elasticity to depend on users' characteristics x_i , so that the heterogeneous elasticity function is defined for the i th user as

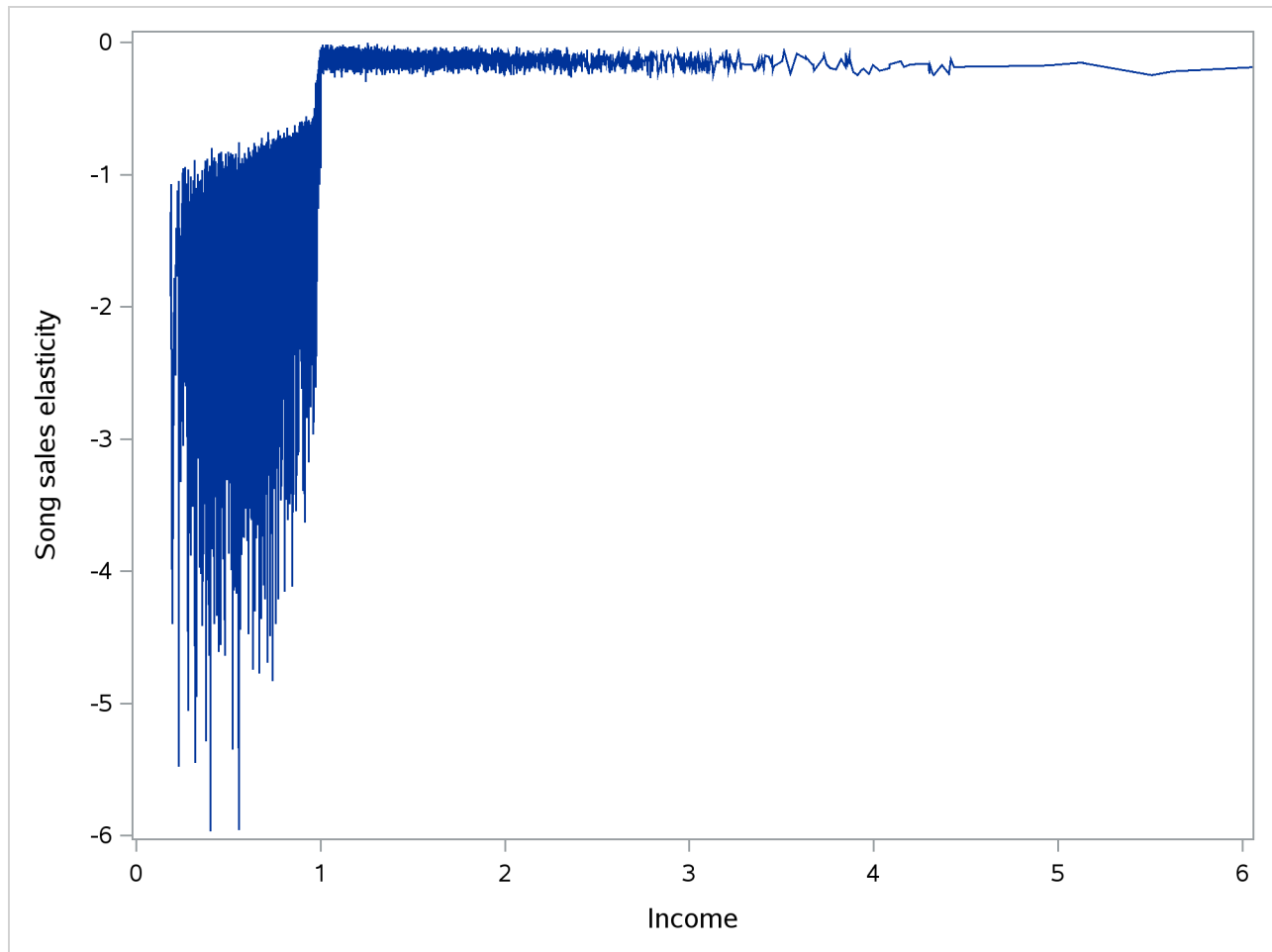
$$\text{elasticity}(x_i) = \beta(x_i) \frac{\text{price}_i}{\text{demand}_i}$$

It is of particular interest to assess how price elasticity varies among users with respect to their income and other personal characteristics. The variable `_beta_` in the data table `odetails` that is obtained during the previous estimation contains the estimated values for this function for each user. Price elasticity is computed for each user by using the following statements:

```
data odetails;
  set mylib.odetails;
  elasticity = _beta_ *price/demand;
run;
```

The estimated price elasticities can then be plotted against various characteristics of users. The following statements plot the relationship between elasticity and income, which is shown in [Output 16.1.2](#):

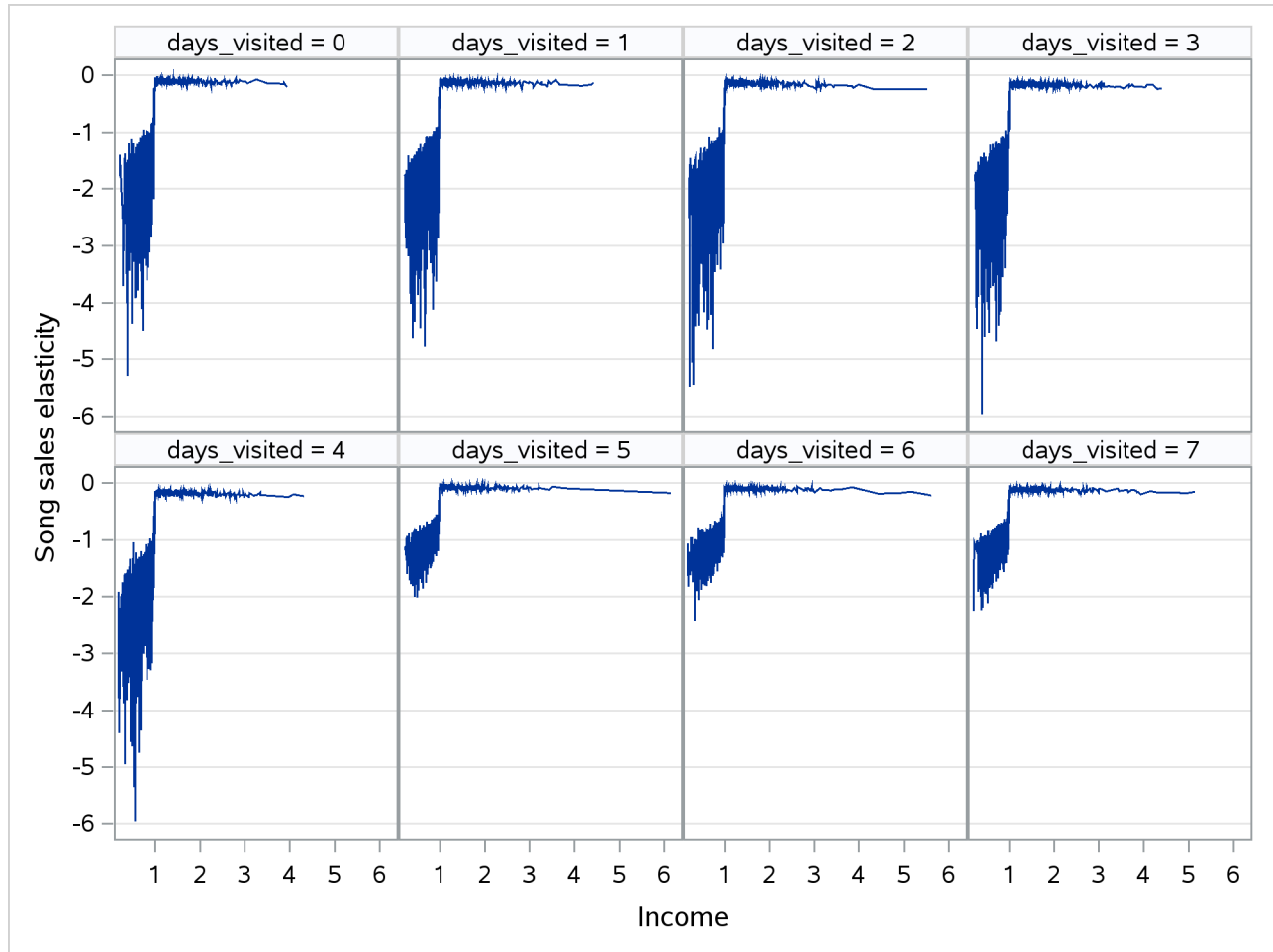
```
proc sort data=odetails ;
  by income;
run;
proc sgplot data=odetails ;
  series x=income y=elasticity /legendlabel='Sales elasticity prediction';
  xaxis label='Income' values=(0 to 6 by 1);
  yaxis label='Song sales elasticity';
run;
```

Output 16.1.2 Song Sales Price Elasticity vs. Income

Output 16.1.2 shows a remarkable heterogeneity among users with respect to income for their response to a price increase. Users whose income is less than 1 (51% of all users) are more sensitive to a price increase; if price goes up by 1%, their number of songs purchased falls by 1% to 5%, compared to around 0.15% for higher-income users.

The following statements plot the relationship between elasticity, income, and days_visited, which is shown in Output 16.1.3:

```
proc sort data=odetails ;
  by days_visited income;
run;
proc sgpanel data=odetails;
  panelby days_visited/layout=panel columns=4 rows=2 uniscale=column;
  series x=income y=elasticity;
  colaxis label='Income' valueshint values=(0 1 2 3 4 5 6);
  rowaxis label='Song sales elasticity' grid;
run;
```

Output 16.1.3 Song Sales Price Elasticity vs. Income and Days Visited

Output 16.1.3 further shows that low-income users who visited the website less often in the past (less than five days a week) are even more price-sensitive than low-income users who visited the website more often. The discussion next turns to policy evaluation and comparison.

In policy optimization, the goal is to maximize the expected utility of a policy. For the definition of the expected utility function and how it is estimated, as well as details such as the definition of a policy rule, see the section “Policy Evaluation and Comparison” on page 990. In this example, the policy decisions are to choose user-specific prices (price_i) in order to maximize the revenue, which is defined as

$$\text{Revenue}(\text{price}_i, \mathbf{x}_i) = (\alpha(\mathbf{x}_i) + \beta(\mathbf{x}_i) * \text{price}_i) * \text{price}_i$$

The optimal price for each user is therefore derived as $\text{price}_i^* = -\frac{\alpha(\mathbf{x}_i)}{2\beta(\mathbf{x}_i)}$. Let $\mathbf{s}1$ be the optimal policy that sets each user’s price as price_i^* . This optimal policy is highly personalized, because it sets user-specific prices during the discount season, but those prices can exceed the original prices. It is therefore more meaningful to consider an optimized discount personalized policy, $\mathbf{s}1d$, that is equal to $\mathbf{s}1$ if $\mathbf{s}1$ does not exceed the original price, and equal to 1 otherwise.

Although the optimal pricing policy $\mathbf{s}1$ guarantees the maximum revenue and is highly personalized, the online media company might want to consider a price discount that targets a group of users or all users. Whether the company’s revenue will rise or fall depends on the price elasticity of songs’ demand. Microeconomic theory

predicts that lowering prices will increase revenue if demand is price-elastic (elasticity < -1) and decrease revenue if demand is price-inelastic (elasticity > -1).

Output 16.1.2 and **Output 16.1.3** show that price elasticity is heterogeneous among users; the most price-sensitive users are those whose income is less than 1 and who spend on average less than five days a week on the website. This suggests a policy that offers this type of user a price discount. Let **s2** be this policy. For comparison, also consider policies **s3** (offer everyone a discount) and **s0** (offer no one a discount).

To select the discount rates, revenues were computed for policies **s2** and **s3** and for each of the six discount values 5%, 10%, 15%, 20%, 25%, and 30%. For policy **s2**, a 5% discount did not improve the base policy price, and although a 10%, 15%, 20%, 25%, or 30% discount generated more revenue than the base policy, there was no significant difference among the five discount rates. A 10% discount was therefore selected for policy **s2**. Using a similar analysis, a 5% discount was selected for policy **s3**.

Policies **s2** and **s3** offer the same discount to all users or groups of users. Versions of these policies (**s4** and **s5**) that randomly select with equal probability a discount rate from the set (5%, 10%, 15%, 20%, 25%, and 30%) are also considered.

The following DATA step creates policies **s0**, **s1**, **s1d**, **s2**, **s3**, **s4**, and **s5** in the data table **odetails**, which also contains the variable **elasticity** that was previously computed:

```
data mylib.discountPolicy;
  array prob[6] _temporary_ (6*0.1666667);
  call streaminit(54321);
  set odetails;
  s0 = 1;
  s1 = -_alpha_/(2*_beta_);
  if s1 > 1 then s1d = 1;
  else s1d = s1;
  if (income < 1 and days_visited < 5) then s2 = 0.9;
  else s2 = 1;
  s3 = 0.95;
  if (income < 1 and days_visited < 5) then s4c = rand("Table", of prob[*]);
  else s4c = 0;
  s4=. ;
  do k = 1 to 7;
    if (s4c = k-1) then s4 = 1 - 0.05*(k-1);
  end;
  s5c = rand("Table", of prob[*]);
  s5=. ;
  do k = 1 to 7;
    if (s5c = k-1) then s5 = 1 - 0.05*(k-1);
  end;
  keep id price account_age age avg_hours days_visited
      friends_count has_membership is_US songs_purchased
      income demand s0 s1 s1d s2 s3 s4 s5 elasticity;
run;
```

Before policy evaluation and comparison are conducted, the variation of the optimal prices with respect to users' price sensitivity and income is assessed.

In the following code, the DATA step defines the variables **elasticity_type** and **price_discount**, and the PROC steps plot the relationship between **price_discount**, **elasticity_type**, and **income**, which

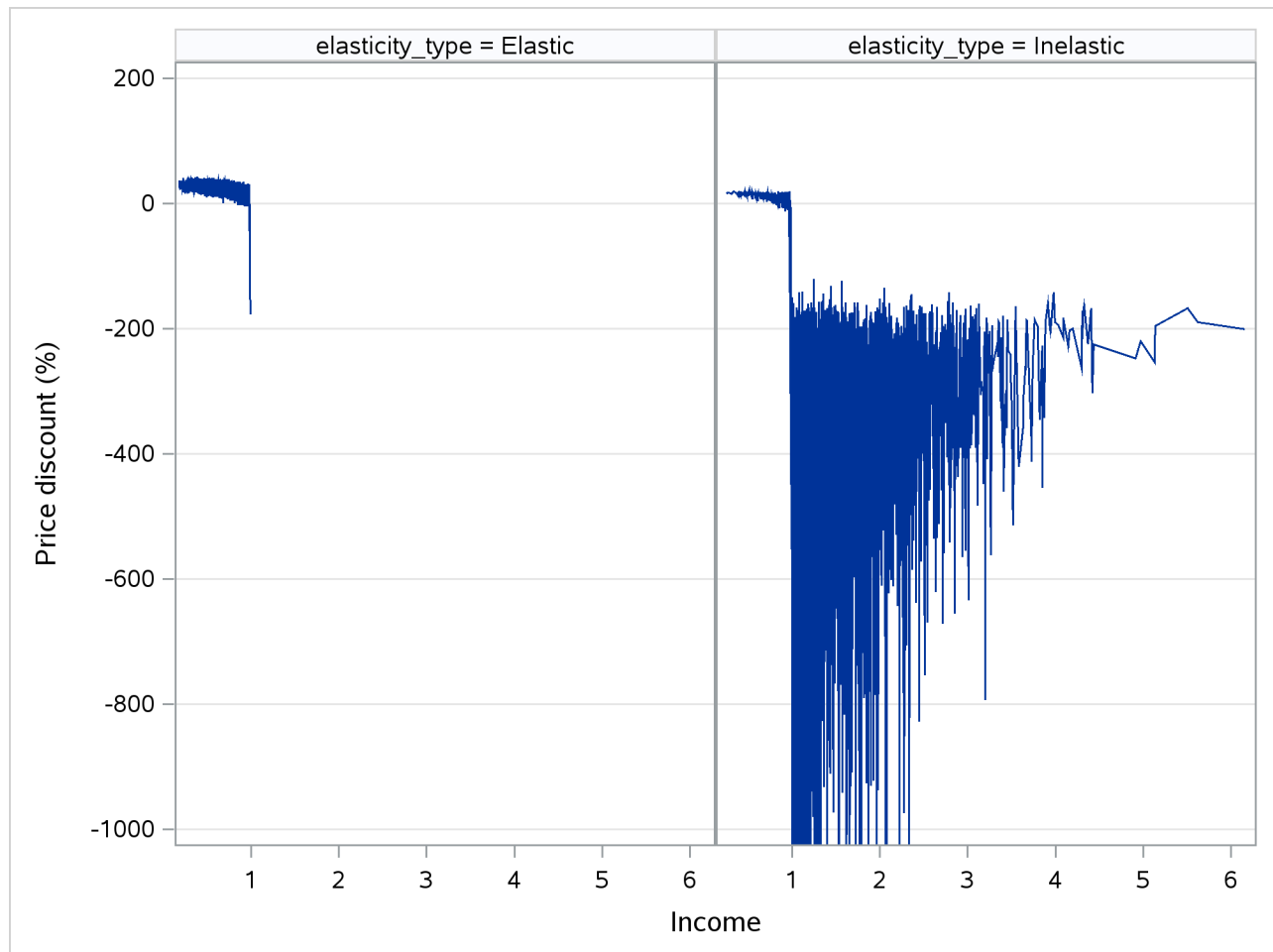
is shown in **Output 16.1.4**:

```

data discountPolicy2;
  set mylib.discountPolicy;
  if elasticity > -1 then elasticity_type = 'Inelastic';
  else if elasticity < -1 then elasticity_type = 'Elastic';
  else if elasticity = -1 then elasticity_type = 'Unit Elastic';
  price_discount = ((1-s1)/1)*100;
run;
proc sort data=discountPolicy2 ;
  by elasticity_type income;
run;
proc sgpanel data=discountPolicy2;
  panelby elasticity_type / layout=panel columns=2 rows=1 uniscale=column;
  series x=income y=price_discount;
  colaxis label='Income' valueshint values=(0 1 2 3 4 5 6);
  rowaxis label='Price discount (%)' grid max=200 min=-1000;
run;

```

Output 16.1.4 Price Discount Distribution by Elasticity and Income



Output 16.1.4 shows that the optimal policy offers a positive discount to users whose income is less than 1 and to nearly all price-sensitive users (elastic group).

You can do policy evaluation by specifying the POLICY= option in the INFER statement, and you can do policy comparison by specifying the POLICYCOMPARISON= option in the INFER statement. The previously estimated treatment model, $\alpha(\cdot)$ information, and $\beta(\cdot)$ information are applied to the new data set by specifying the INT=, INA=, and INB= options, respectively, in the SCORE statement. The following statements estimate the model and do policy evaluation and comparison:

```
/*--- Policy evaluation and comparison ---*/
proc deepprice data=mylib.discountPolicy;
  id id;
  infer out=mylib.oest2 outdetails=mylib.odetails2
    policy=(price s0 s1 s1d s2 s3 s4 s5)
    policyComparison=(base=(price)
      compare=(s0 s1 s1d s2 s3 s4 s5));
  score int=mylib.outtdata ina=mylib.outadata inb=mylib.outbdata ;
run;
```

The POLICY= option performs the policy evaluation for the initial price policy (price) and policies s0, s1, s1d, s2, s3, s4, and s5. The COMPARE= suboption of the POLICYCOMPARISON= option compares policies s0, s1, s1d, s2, s3, s4, and s5 to the base policy, price, which is specified by the BASE= suboption of the POLICYCOMPARISON= option.

The policy evaluation results are shown in [Output 16.1.5](#).

Output 16.1.5 Policy Evaluation

The DEEPPRICE Procedure

Policy Evaluation						
Policy	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
price	14.648911	0.073155	14.505529	14.792293	200.24	<.0001
s0	14.393804	0.077702	14.241511	14.546096	185.24	<.0001
s1	12.017483	0.659866	10.724169	13.310798	18.21	<.0001
s1d	16.271219	0.059482	16.154637	16.387800	273.55	<.0001
s2	14.869897	0.069841	14.733011	15.006784	212.91	<.0001
s3	14.855598	0.072180	14.714128	14.997069	205.81	<.0001
s4	15.220208	0.069391	15.084205	15.356211	219.34	<.0001
s5	15.985290	0.070627	15.846863	16.123717	226.33	<.0001

The optimal policy s1 has the lowest expected demand, whereas the optimized discount policy s1d and the policy s5, which offer each user a randomly selected discount, generate the most demand. Policies s2, s3, and s4 generate slightly more demand than s0 (no user gets a discount).

[Output 16.1.6](#) shows the output for the policy comparison. Policies s1d and s5 are better than the current price policy with respect to demand.

Output 16.1.6 Policy Comparison

Policy Comparison						
Comparison Policy	Base Policy	Estimate	Standard Error	95% Confidence Limits		Z Pr > Z
s0	price	-0.255108	0.035676	-0.325032	-0.185183	-7.15 <.0001
s1	price	-2.631428	0.671764	-3.948062	-1.314794	-3.92 <.0001
s1d	price	1.622307	0.041549	1.540872	1.703742	39.05 <.0001
s2	price	0.220986	0.029949	0.162287	0.279685	7.38 <.0001
s3	price	0.206687	0.027870	0.152063	0.261311	7.42 <.0001
s4	price	0.571296	0.037946	0.496924	0.645669	15.06 <.0001
s5	price	1.336379	0.035783	1.266245	1.406512	37.35 <.0001

Output 16.1.5 and Output 16.1.6 evaluate and compare the different policies on the basis of demand. However, it would be better for the comparison of the policies to be based on revenue.

The following statements evaluate and compare the different policies by computing the average revenue per user and the associated confidence interval for each policy in the data table odetails2:

```
data odetails2;
  set mylib.odetails2;
  if (_alpha+_beta*price>0) then revenue_price = price*(_alpha+_beta*price);
  else revenue_price = 0;
  if (_alpha+_beta*s0>0) then revenue_s0 = s0*(_alpha+_beta*s0);
  else revenue_s0 = 0;
  if (_alpha+_beta*s1>0 and s1>0) then revenue_s1 = s1*(_alpha+_beta*s1);
  else revenue_s1 = 0;
  if (_alpha+_beta*s1d>0 and s1d>0) then revenue_s1d = s1d*(_alpha+_beta*s1d);
  else revenue_s1d = 0;
  if (_alpha+_beta*s2>0) then revenue_s2 = s2*(_alpha+_beta*s2);
  else revenue_s2 = 0;
  if (_alpha+_beta*s3>0) then revenue_s3 = s3*(_alpha+_beta*s3);
  else revenue_s3 = 0;
  if (_alpha+_beta*s4>0) then revenue_s4 = s4*(_alpha+_beta*s4);
  else revenue_s4 = 0;
  if (_alpha+_beta*s5>0) then revenue_s5 = s5*(_alpha+_beta*s5);
  else revenue_s5 = 0;
run;
proc means data=odetails2 noprint;
  var revenue_price revenue_s0 revenue_s1 revenue_s1d revenue_s2 revenue_s3
      revenue_s4 revenue_s5;
  output out=odetails2_stats(drop=_type_ _freq_) mean= STD= N=/autoname;
run;
data odetails2_stats;
  set odetails2_stats;
  s0_price_Estimate = revenue_s0_mean - revenue_price_mean;
  s1_price_Estimate = revenue_s1_mean - revenue_price_mean;
  s1d_price_Estimate = revenue_s1d_mean - revenue_price_mean;
  s2_price_Estimate = revenue_s2_mean - revenue_price_mean;
  s3_price_Estimate = revenue_s3_mean - revenue_price_mean;
  s4_price_Estimate = revenue_s4_mean - revenue_price_mean;
  s5_price_Estimate = revenue_s5_mean - revenue_price_mean;
```

```

revenue_price_StdErr = revenue_price_StdDev/sqrt(revenue_price_N);
revenue_s0_StdErr = revenue_s0_StdDev/sqrt(revenue_s0_N);
revenue_s1_StdErr = revenue_s1_StdDev/sqrt(revenue_s1_N);
revenue_s1d_StdErr = revenue_s1d_StdDev/sqrt(revenue_s1d_N);
revenue_s2_StdErr = revenue_s2_StdDev/sqrt(revenue_s2_N);
revenue_s3_StdErr = revenue_s3_StdDev/sqrt(revenue_s3_N);
revenue_s4_StdErr = revenue_s4_StdDev/sqrt(revenue_s4_N);
revenue_s5_StdErr = revenue_s5_StdDev/sqrt(revenue_s5_N);

s0_price_StdErr = sqrt(revenue_s0_StdErr**2 + revenue_price_StdErr**2);
s1_price_StdErr = sqrt(revenue_s1_StdErr**2 + revenue_price_StdErr**2);
s1d_price_StdErr = sqrt(revenue_s1d_StdErr**2 + revenue_price_StdErr**2);
s2_price_StdErr = sqrt(revenue_s2_StdErr**2 + revenue_price_StdErr**2);
s3_price_StdErr = sqrt(revenue_s3_StdErr**2 + revenue_price_StdErr**2);
s4_price_StdErr = sqrt(revenue_s4_StdErr**2 + revenue_price_StdErr**2);
s5_price_StdErr = sqrt(revenue_s5_StdErr**2 + revenue_price_StdErr**2);

run;
proc transpose data=odetails2_stats(drop=revenue_price_N revenue_s1_N
    revenue_s1d_N revenue_s2_N revenue_s3_N revenue_s4_N revenue_s5_N
    revenue_s0_N revenue_price_StdDev revenue_s1_StdDev revenue_s1d_StdDev
    revenue_s2_StdDev revenue_s3_StdDev revenue_s4_StdDev revenue_s5_StdDev
    revenue_s0_StdDev) OUT=odetails2_stats;

run;
data odetails2_stats2;
    set odetails2_stats;
    ComparisonPolicy=scan(_name_,1,'_');
    BasePolicy=scan(_name_,2,'_');
    stat=scan(_name_,3,'_');
    if ComparisonPolicy = 'revenue' then Policy = BasePolicy;
    drop _name_;

run;
proc sort data=odetails2_stats2;
    by ComparisonPolicy BasePolicy Policy;

run;
proc transpose data=odetails2_stats2 out=odetails2_stats2(drop=_name_);
    by ComparisonPolicy BasePolicy Policy;
    id stat;
    var coll;

run;
data odetails2_stats2;
    format ProbZ pvalue6.4;
    set odetails2_stats2;
    if Estimate =. then Estimate=Mean;
    Z = Estimate/StdErr;
    LowerCL = Estimate - StdErr*quantile('NORMAL',.975);
    UpperCL = Estimate + StdErr*quantile('NORMAL',.975);
    ProbZ = 2*(1-cdf('NORMAL',abs(Z)));

run;

```

The table that shows policy evaluation based on revenue is printed by the following statements and shown in [Output 16.1.7](#). As expected, the optimal personalized policy s1 generates the most revenue. The optimized personalized discount policy s1d and the personalized policies s2 and s4 generate comparable revenue, which is also the second-highest revenue.

```
proc print data=odetails2_stats2 noobs;
  var Policy Estimate StdErr LowerCL UpperCL ProbZ;
  where Policy ~= '';
run;
```

Output 16.1.7 Policy Evaluation Based on Revenue

	Policy	Estimate	StdErr	LowerCL	UpperCL	ProbZ
price		15.0378	0.06639	14.9077	15.1679	<.0001
s0		15.1765	0.07319	15.0331	15.3200	<.0001
s1		34.4007	0.33318	33.7477	35.0537	<.0001
s1d		15.6764	0.06750	15.5441	15.8087	<.0001
s2		15.4018	0.07024	15.2641	15.5395	<.0001
s3		14.8507	0.06671	14.7199	14.9814	<.0001
s4		15.4653	0.06937	15.3294	15.6013	<.0001
s5		13.7614	0.05384	13.6558	13.8669	<.0001

Output 16.1.8 shows the output for the policy comparison based on revenue. The personalized policies s1, s1d, s2, and s4 generate more average revenue per user than the base policy price. The policy s0 (offer no user a discount) is equivalent to price, and policies that offer every user a discount (s3 and s5) are worse than price, with s5 being the worst. The following statements print the policy comparison results, as shown in Output 16.1.8:

```
proc print data=odetails2_stats2 noobs;
  var ComparisonPolicy BasePolicy Estimate StdErr LowerCL UpperCL ProbZ;
  where Policy = '';
run;
```

Output 16.1.8 Policy Comparison Based on Revenue

	ComparisonPolicy	BasePolicy	Estimate	StdErr	LowerCL	UpperCL	ProbZ
s0	price		0.1387	0.09881	-0.0549	0.3324	0.1603
s1	price		19.3629	0.33973	18.6970	20.0287	<.0001
s1d	price		0.6385	0.09467	0.4530	0.8241	<.0001
s2	price		0.3640	0.09665	0.1745	0.5534	0.0002
s3	price		-0.1872	0.09411	-0.3716	-0.0027	0.0467
s4	price		0.4275	0.09602	0.2393	0.6157	<.0001
s5	price		-1.2764	0.08548	-1.4440	-1.1089	<.0001

In fact, the data generating process (DGP) behind the observed demand in the pricing_sample data set is known, and the objectives of this final section are to assess how well the estimated intercept and slope functions approximate the truth and to compare the different policies by using true DGP revenues.

The true DGP is defined as follows:

$$\alpha(\mathbf{x}) = 20 + 0.5 * \text{avg_hours} + 5 * (\text{days_visited} > 4)$$

$$\beta(\mathbf{x}) = -3 - 14 * (\text{income} < 1)$$

$$\text{demand} = \alpha(\mathbf{x}) + \beta(\mathbf{x}) * \text{price}$$

The following statements compute the true intercept and slope functions, as well as the revenue that is generated per user by the different price policies and by the true DGP demand. The optimal policies $s1star$ and $s1dstar$ are the true DGP counterparts of the estimated policies $s1$ and $s1d$. The $s1star$ policy is the best revenue-maximizing policy in theory, but it is not available in practice because it depends on the unknown functions $\alpha(\cdot)$ and $\beta(\cdot)$.

```
data odetails2_DGP;
  set mylib.odetails2;
  alpha_dgp = 20 + 0.5*avg_hours + 5*(days_visited > 4);
  beta_dgp = -3 -14*(income < 1);

  s1star = -alpha_dgp/(2*beta_dgp);
  if s1star>1 then s1dstar = 1; else s1dstar = s1star;

  if (alpha_dgp+beta_dgp*price>0) then
    revenue_price = price*(alpha_dgp+beta_dgp*price);
  else revenue_price = 0;
  if (alpha_dgp+beta_dgp*s0>0) then
    revenue_s0 = s0*(alpha_dgp+beta_dgp*s0);
  else revenue_s0 = 0;
  if (alpha_dgp+beta_dgp*s1>0 and s1>0) then
    revenue_s1 = s1*(alpha_dgp+beta_dgp*s1);
  else revenue_s1 = 0;
  if (alpha_dgp+beta_dgp*s1d>0 and s1d>0) then
    revenue_s1d = s1d*(alpha_dgp+beta_dgp*s1d);
  else revenue_s1d = 0;
  if (alpha_dgp+beta_dgp*s1star>0 and s1star>0) then
    revenue_s1star = s1star*(alpha_dgp+beta_dgp*s1star);
  else revenue_s1star = 0;
  if (alpha_dgp+beta_dgp*s1dstar>0 and s1dstar>0) then
    revenue_s1dstar = s1dstar*(alpha_dgp+beta_dgp*s1dstar);
  else revenue_s1dstar = 0;
  if (alpha_dgp+beta_dgp*s2>0) then revenue_s2 = s2*(alpha_dgp+beta_dgp*s2);
  else revenue_s2 = 0;
  if (alpha_dgp+beta_dgp*s3>0) then revenue_s3 = s3*(alpha_dgp+beta_dgp*s3);
  else revenue_s3 = 0;
  if (alpha_dgp+beta_dgp*s4>0) then revenue_s4 = s4*(alpha_dgp+beta_dgp*s4);
  else revenue_s4 = 0;
  if (alpha_dgp+beta_dgp*s5>0) then revenue_s5 = s5*(alpha_dgp+beta_dgp*s5);
  else revenue_s5 = 0;
run;
```

The true and estimated intercept functions are then plotted using the code that follows. The results are shown in [Output 16.1.9](#). The estimated and true intercepts are highly positively correlated, and the estimated slope function reasonably approximates the true slope except in the case of high-income users. It should be noted that the estimated intercept and slope are functions of

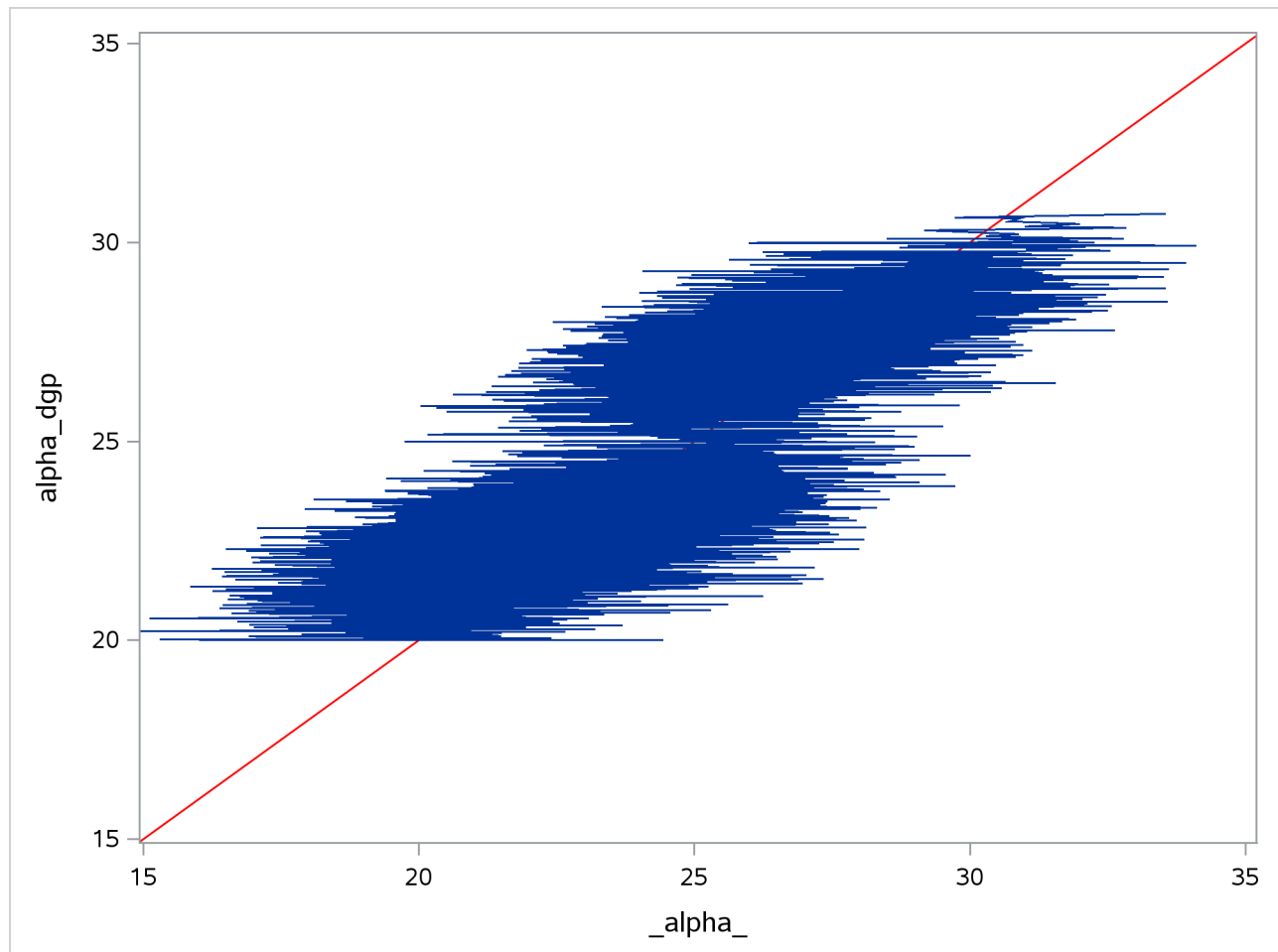
all nine users' characteristics, whereas the true intercept and slope are functions of only income, avg_hours, and days_visited. Despite this difference, the DEEPPRICE procedure is able to accurately estimate the patterns in these functions.

```

proc sort data=odetails2_DGP ;
  by alpha_dgp;
run;
proc sgplot data=odetails2_DGP noautolegend;
  lineparm x=0 y=0 slope=1/lineattrs=(color=red) ;
  series x=_alpha_ y=alpha_dgp ;
  xaxis min=15 max=35;
  yaxis min=15 max=35;
run;

```

Output 16.1.9 Estimated vs. True Intercept Function $\alpha(\cdot)$



The following statements plot the true and estimated slope functions, as shown in [Output 16.1.10](#):

```

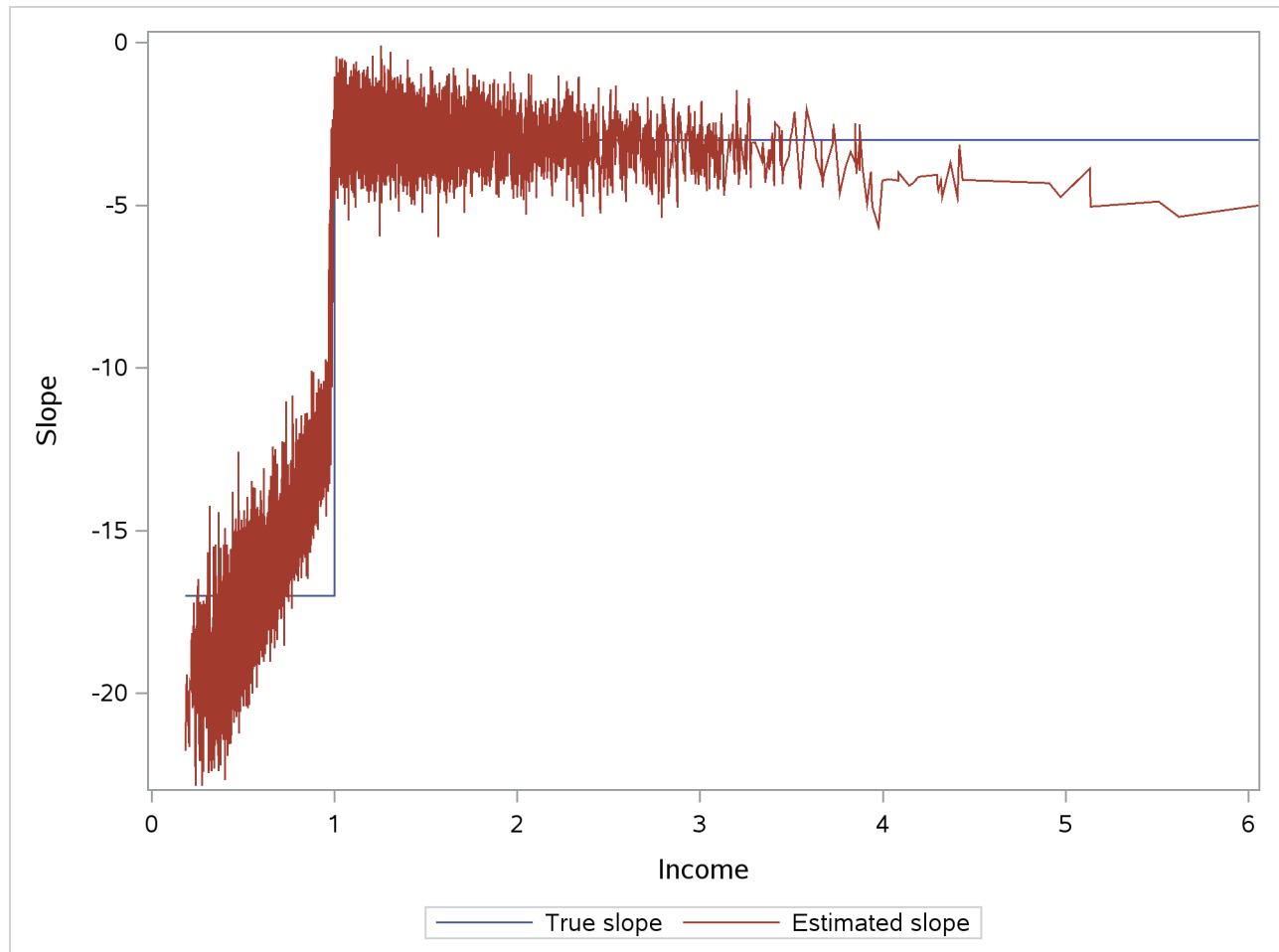
proc sort data=odetails2_DGP ;
  by income;
run;
proc sgplot data=odetails2_DGP ;
  series x=income y=beta_dgp /legendlabel='True slope';
  series x=income y=_beta_ /legendlabel='Estimated slope';
  xaxis label='Income' values=(0 to 6 by 1);

```



```
yaxis label='Slope';
run;
```

Output 16.1.10 Estimated and True Slope Function $\beta(\cdot)$ vs. Income



As with the non-DGP revenues, the following statements compute the average revenue per user for each price policy in the data table `odetails2_DGP`:

```
proc means data=odetails2_DGP noprint;
  var revenue_price revenue_s0 revenue_s1 revenue_s1star revenue_s1d
      revenue_s1dstar revenue_s2 revenue_s3 revenue_s4 revenue_s5;
  output out=odetails2_DGP_stats(drop=_type_ _freq_) mean= STD= N=/autoname;
run;
data odetails2_DGP_stats;
  set odetails2_DGP_stats;
  s0_price_Estimate = revenue_s0_mean - revenue_price_mean;
  s1_price_Estimate = revenue_s1_mean - revenue_price_mean;
  s1d_price_Estimate = revenue_s1d_mean - revenue_price_mean;
  s1star_price_Estimate = revenue_s1star_mean - revenue_price_mean;
  s1dstar_price_Estimate = revenue_s1dstar_mean - revenue_price_mean;
  s2_price_Estimate = revenue_s2_mean - revenue_price_mean;
  s3_price_Estimate = revenue_s3_mean - revenue_price_mean;
  s4_price_Estimate = revenue_s4_mean - revenue_price_mean;
```

```

s5_price_Estimate = revenue_s5_mean - revenue_price_mean;

revenue_s0_StdErr = revenue_s0_StdDev/sqrt(revenue_s0_N);
revenue_price_StdErr = revenue_price_StdDev/sqrt(revenue_price_N);
    revenue_s1_StdErr = revenue_s1_StdDev/sqrt(revenue_s1_N);
revenue_s1d_StdErr = revenue_s1d_StdDev/sqrt(revenue_s1d_N);
revenue_s1star_StdErr = revenue_s1star_StdDev/sqrt(revenue_s1star_N);
revenue_s1dstar_StdErr = revenue_s1dstar_StdDev/sqrt(revenue_s1dstar_N);
revenue_s2_StdErr = revenue_s2_StdDev/sqrt(revenue_s2_N);
revenue_s3_StdErr = revenue_s3_StdDev/sqrt(revenue_s3_N);
revenue_s4_StdErr = revenue_s4_StdDev/sqrt(revenue_s4_N);
revenue_s5_StdErr = revenue_s5_StdDev/sqrt(revenue_s5_N);

s0_price_StdErr = sqrt(revenue_s0_StdErr**2 + revenue_price_StdErr**2);
    s1_price_StdErr = sqrt(revenue_s1_StdErr**2 + revenue_price_StdErr**2);
s1d_price_StdErr = sqrt(revenue_s1d_StdErr**2 + revenue_price_StdErr**2);
s1star_price_StdErr = sqrt(revenue_s1star_StdErr**2 + revenue_price_StdErr**2);
s1dstar_price_StdErr = sqrt(revenue_s1dstar_StdErr**2 + revenue_price_StdErr**2);
s2_price_StdErr = sqrt(revenue_s2_StdErr**2 + revenue_price_StdErr**2);
s3_price_StdErr = sqrt(revenue_s3_StdErr**2 + revenue_price_StdErr**2);
s4_price_StdErr = sqrt(revenue_s4_StdErr**2 + revenue_price_StdErr**2);
s5_price_StdErr = sqrt(revenue_s5_StdErr**2 + revenue_price_StdErr**2);
run;
proc transpose data=odetails2_DGP_stats(drop= revenue_price_N revenue_s0_N
    revenue_s1_N revenue_s1d_N revenue_s1star_N revenue_s1dstar_N
        revenue_s2_N revenue_s3_N revenue_s4_N revenue_s5_N revenue_price_StdDev
        revenue_s0_StdDev revenue_s1_StdDev revenue_s1d_StdDev revenue_s1star_StdDev
        revenue_s1dstar_StdDev revenue_s2_StdDev revenue_s3_StdDev revenue_s4_StdDev
        revenue_s5_StdDev) OUT=odetails2_DGP_stats;
run;
data odetails2_DGP_stats2;
    set odetails2_DGP_stats;
    ComparisonPolicy=scan(_name_,1,'_');
    BasePolicy=scan(_name_,2,'_');
    stat=scan(_name_,3,'_');
    if ComparisonPolicy = 'revenue' then Policy = BasePolicy;
    drop _name_;
run;
proc sort data=odetails2_DGP_stats2;
    by ComparisonPolicy BasePolicy Policy;
run;
proc transpose data=odetails2_DGP_stats2
    out=odetails2_DGP_stats2(drop=_name_);
    by ComparisonPolicy BasePolicy Policy;
    id stat;
    var coll;
run;
data odetails2_DGP_stats2;
    format ProbZ pvalue6.4;
    set odetails2_DGP_stats2;
    if Estimate =. then Estimate=Mean;
    Z = Estimate/StdErr;
    LowerCL = Estimate - StdErr*quantile('NORMAL',.975);
    UpperCL = Estimate + StdErr*quantile('NORMAL',.975);

```

```

    ProbZ = 2*(1-cdf('NORMAL',abs(Z)));
run;

```

The true DGP average revenue per user and the corresponding confidence interval for each price policy are printed using the following code and shown in [Output 16.1.11](#) and [Output 16.1.12](#), respectively:

```

data odetails2_DGP_stats2_eval;
    set odetails2_DGP_stats2(where=(Policy ~= ''));
    order = _n_;
    if Policy = 's1star' then order = 4;
    if Policy = 's1d' then order = 5;
    if Policy = 's1dstar' then order = 6;
run;
proc sort data=odetails2_DGP_stats2_eval;
    by order;
run;
proc print data=odetails2_DGP_stats2_eval noobs;
    var Policy Estimate StdErr LowerCL UpperCL ProbZ;
run;

```

Output 16.1.11 Policy Evaluation Based on True DGP Revenue

Policy	Estimate	StdErr	LowerCL	UpperCL	ProbZ
price	14.3493	0.06697	14.2181	14.4806	<.0001
s0	14.2946	0.07480	14.1480	14.4412	<.0001
s1	25.9909	0.20765	25.5839	26.3979	<.0001
s1star	29.2230	0.22100	28.7898	29.6561	<.0001
s1d	14.9563	0.06759	14.8238	15.0888	<.0001
s1dstar	15.0342	0.06684	14.9032	15.1652	<.0001
s2	14.6058	0.07117	14.4663	14.7453	<.0001
s3	14.0598	0.06796	13.9266	14.1930	<.0001
s4	14.7221	0.06988	14.5851	14.8591	<.0001
s5	13.1714	0.05423	13.0651	13.2777	<.0001

The following statements print the comparison of each policy to price, as shown in [Output 16.1.12](#):

```

data odetails2_DGP_stats2_comp;
    set odetails2_DGP_stats2(where=(Policy = ''));
    order = _n_;
    if ComparisonPolicy = 's1star' then order = 4;
    if ComparisonPolicy = 's1d' then order = 5;
    if ComparisonPolicy = 's1dstar' then order = 6;
run;
proc sort data=odetails2_DGP_stats2_comp;
    by order;
run;
proc print data=odetails2_DGP_stats2_comp noobs;
    var ComparisonPolicy BasePolicy Estimate StdErr LowerCL UpperCL ProbZ;
run;

```

Output 16.1.12 Policy Comparison Based on True DGP Revenue

Comparison	Policy	BasePolicy	Estimate	StdErr	LowerCL	UpperCL	ProbZ
s0	price		-0.0547	0.10040	-0.2515	0.1421	0.5856
s1	price		11.6416	0.21818	11.2139	12.0692	<.0001
s1star	price		14.8736	0.23093	14.4210	15.3262	<.0001
s1d	price		0.6070	0.09515	0.4205	0.7935	<.0001
s1dstar	price		0.6849	0.09462	0.4994	0.8703	<.0001
s2	price		0.2564	0.09773	0.0649	0.4480	0.0087
s3	price		-0.2895	0.09542	-0.4766	-0.1025	0.0024
s4	price		0.3728	0.09679	0.1830	0.5625	0.0001
s5	price		-1.1779	0.08618	-1.3468	-1.0090	<.0001

Output 16.1.11 shows that the average revenue per user that is generated by the estimated optimal policy s1 is less than but close to that obtained using its true DGP counterpart and best policy, s1star. Without knowing the true DGP, the DEEPPRICE procedure can accurately estimate the average optimal revenue per user. The optimized discount policies s1d and s1dstar generate comparable but substantially less revenue than s1 and s1star.

In terms of policy comparison, Output 16.1.12 shows that the s1 and s1star policies generate much more revenue per user than the base policy price. The optimized discount personalized policies (s1d and s1dstar) and the personalized policies that offer only price-sensitive users a discount (s2 and s4) are better than price, whereas the policies that offer every user a discount (s3 and s5) are worse than price.

In summary, this example shows how, by taking advantage of the flexibility of PROC DEEPPRICE in specifying heterogeneous price effects as a function of users' characteristics, the online media company can set user-specified prices to achieve maximum revenue and discount policies that target a specified group of users.

Example 16.2: Functional Forms and Analyzing Price and Demand

This example demonstrates the effect of functional form misspecification on the estimated demand curve and shows how to compare demand curves across customer characteristics and purchasing time.

The data set has 10,000 simulated observations, which include customer type, customer purchase time, price, and demand. The treatment variable, price, is the selling price of a product. The outcome variable, demand, is the sales quantity of the product. Table 16.4 describes the variables that are used in the model. The type can be T (treatment), Y (outcome), $x^{(1)}$ (a covariate in the treatment model), and/or $x^{(2)}$ (a covariate in the outcome model).

Table 16.4 Variables in Model

Name	Type	Details
customertype	$x^{(1)}, x^{(2)}$	Customer type
time	$x^{(1)}, x^{(2)}$	Time of purchase of goods
demand	Y	Quantity of purchased goods
price	T	Selling price of a product

The following statements create the data set and load the data into your session:

```
data deepPriceData;
  input i customertype time price demand ;
  datalines;
1      10 3.2517  0.82502 0.59985
2       8 9.6777  1.47760 0.35881
3       1 6.4239  1.82441 2.24215
4       4 10.0217 1.56411 1.18040
5       3 11.2567 2.13610 1.39896

... more lines ...

9995   8 3.4223  1.04572 0.38379
9996   6 11.1666 1.65613 1.29208
9997   3 2.9016  1.11613 1.97467
9998   6 3.8167  0.97021 1.05249
9999   7 3.7612  1.17288 0.35869
10000  4 1.2474  1.13899 1.51599
;

data mylib.deeppricedata;
  set deeppricedata;
run;
```

It is first assumed that the demand curve is linear. The following statements use PROC DEEPPRICE to estimate the intercept and slope functions and then to save the estimation details:

```
proc deepprice data=mylib.deeppricedata;
  id i;
  tmodel price = customerType time /
    dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=500 regL1=0.0001 maxEpochs=32000
    algorithm=adam) nthreads=20 seed=12345 recordseed=67890));
  model demand = customerType time /
    dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=500 regL1=0.001 maxEpochs=32000
    algorithm=adam) nthreads=20 seed=12345 recordseed=67890));
  infer out=mylib.oest outdetails=mylib.outtabdetails;
  score outt=mylib.outtdata outa=mylib.outadata outb=mylib.outbdata;
run;
```

For this example, the same covariates are used in both the treatment model and the outcome model. The model estimation details are saved in the output data table outtabdetails.

The estimation results are shown in [Output 16.2.1](#).

Output 16.2.1 Parameter Estimates

Parameter Estimates						
Parameter	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
Average Intercept	23.629698	0.215064	23.208180	24.051217	109.87	<.0001
Average Slope	-9.235895	0.244447	-9.715001	-8.756788	-37.78	<.0001

This predicted demand might not be accurate, because the covariates might affect both price and demand. The following statements show how you can construct the grid of price and calculate the predicted demand by using the influence functions `_if_alpha_` and `_if_beta_` that you saved in the `outtabdetails` data set in the initial call to PROC DEEPPRICE and then how you can plot the demand curve:

```
data myGrid_if;
  array gridPrice[100];
  array gridY[100];
  set mylib.outtabdetails;
  do j = 1 to 100;
    gridPrice[j] = 0.7 + 2.8*j / 101;
    gridY[j] = _if_alpha_ + _if_beta_*gridPrice[j] ;
  end;
run;
```

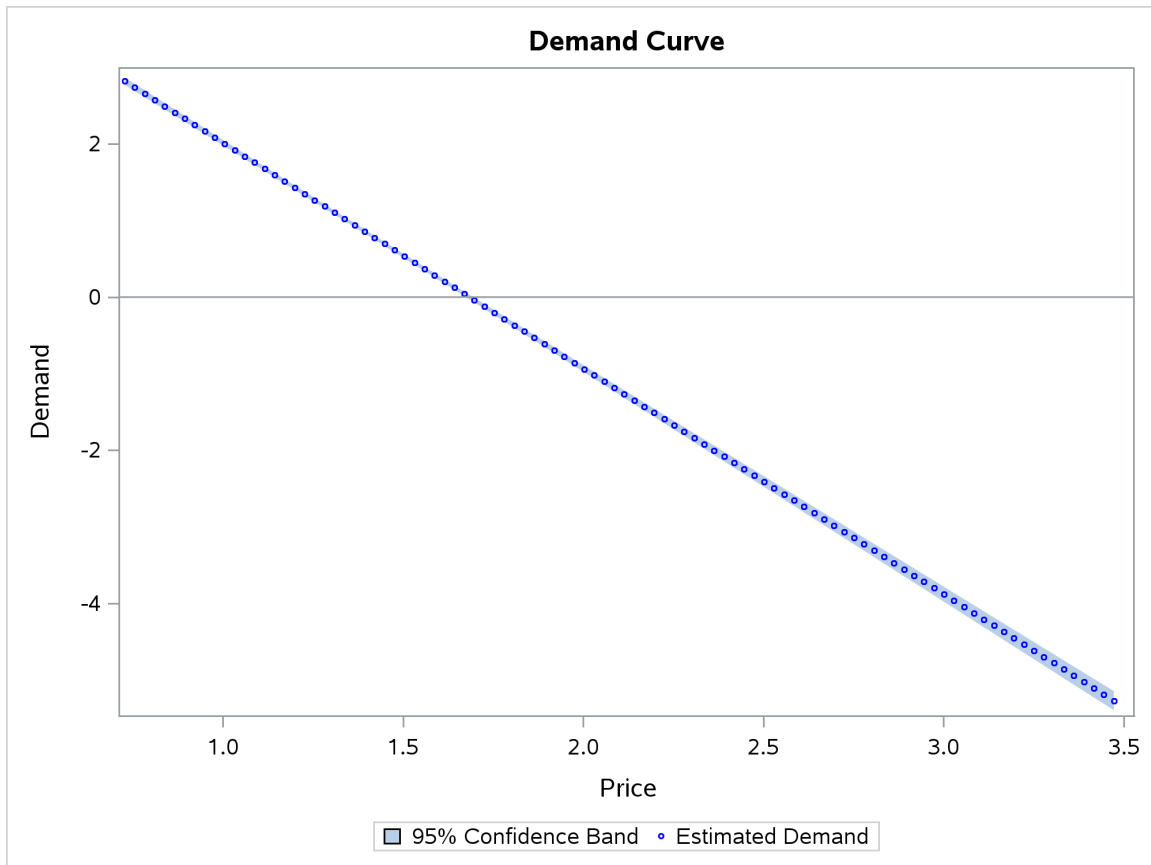
The following statements compute the average demand, its standard error, and its 95% confidence interval at each of the 100 price values and then plot the results:

```
proc univariate data=mygrid_if;
  var gridPrice1-gridPrice100 gridy1-gridy100;
  output out= estCurve_if
  mean=gridPrice1-gridPrice100 gridym1-gridym100
  std=gridPrices1-gridPrices100 gridys1-gridys100;
run;

data estCurve_if_ci;
  set estCurve_if;
  array prices[100] gridPrice1-gridPrice100;
  array demands[100] gridym1-gridym100;
  array stds[100] gridys1-gridys100;
  do i = 1 to 100;
    price=prices[i];
    estDemand=demands[i];
    estDemandLb = estDemand - 1.96*stds[i]/sqrt(10000);
    estDemandUb = estDemand + 1.96*stds[i]/sqrt(10000);
    output;
  end;
  keep price estDemand estDemandLb estDemandUb;
run;

proc sgplot data= estCurve_if_ci;
  title 'Demand Curve';
  band x=price lower=estDemandLb upper=estDemandUb /
  legendlabel="95% Confidence Band " name="band";
  scatter x=price y=estDemand / markerattrs=(color=blue size=3 symbol=circle)
  legendlabel="Estimated Demand" name="estDemand";
  refline 0;
  xaxis label='Price';
  yaxis label='Demand' ;
run;
```

The generated demand curve and 95% pointwise confidence band are shown in [Output 16.2.2](#).

Output 16.2.2 Estimated Demand Curve

The curve looks unrealistic, especially given that a large portion of the curve is below zero. This indicates that the linear demand curve assumption might be incorrect. There is another way to model price and demand: by using log transformation.

The following DATA step computes the logarithm of price and demand data:

```
data mylib.deeppricedata_log;
  set deepPriceData;
  logy=log(demand);
  logp=log(price);
run;
```

The following statements use PROC DEEPPRICE to estimate the price effect and save the estimation details to the output table outtabdetails_log:

```
proc deepprice data=mylib.deeppricedata_log;
  id i;
  tmodel logp = customerType time /
    dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=500 regL1=0.0001 maxEpochs=32000
    algorithm=adam) nthreads=20 seed=12345 recordseed=67890));
  model logy = customerType time /
    dnn=(nodes=(32 32 32 32)
    train=(optimizer=(miniBatchSize=500 regL1=0.001 maxEpochs=32000
```

```

        algorithm=adam) nthreads=20 seed=12345 recordseed=67890));
infer out=mylib.oest_log outdetails=mylib.outtabdetails_log;
score outt=mylib.outtdata_log outa=mylib.outadata_log outb=mylib.outbdata_log;
run;

```

The estimation results are shown in [Output 16.2.3](#). The average slope is negative, indicating that the impact of price on demand is negative. This is consistent with expectations.

Output 16.2.3 Parameter Estimates

Demand Curve

The DEEPPRICE Procedure

Parameter Estimates						
Parameter	Estimate	Standard Error	95% Confidence Limits		Z	Pr > Z
Average Intercept	0.628116	0.009709	0.609087	0.647145	64.69	<.0001
Average Slope	-2.859250	0.011297	-2.881392	-2.837109	-253.10	<.0001

The following DATA step constructs a grid of price values and calculates the corresponding expected demand, whose influence function is given by

$$\psi(\cdot) = \exp(\alpha(x) + \beta(x) \log(t)) \{1 + (\psi_{\alpha}(\cdot) - \alpha(x)) + \log(t)(\psi_{\beta}(\cdot) - \beta(x))\}$$

```

data myGrid_log_if;
    array gridPrice[100];
    array gridY[100];
    set mylib.outtabdetails_log;
    do j = 1 to 100;
        gridPrice[j] = 0.7 + 2.8*j / 101 ;
        gridY[j] = exp(_alpha_ + _beta_ * log(gridPrice[j]))*(1+(_if_alpha_-_alpha_)
            +log(gridPrice[j])*(_if_beta_-_beta_));
    end;
run;

```

The following PROC UNIVARIATE and DATA steps compute the average demand, its standard error, and its 95% confidence interval at each of the 100 price values:

```

proc univariate data=myGrid_log_if noprint;
    var gridPrice1-gridPrice100 gridy1-gridy100;
    output out=estCurve_log_if mean=gridPrice1-gridPrice100 gridy1-gridy100
        std=gridPrice1-gridPrice100 gridys1-gridys100;
run;

data estCurve_log_if_ic;
    set estCurve_log_if;
    array prices[100] gridPrice1-gridPrice100;
    array demands[100] gridy1-gridy100;
    array stds[100] gridys1-gridys100;
    do i = 1 to 100;
        price=prices[i];
        estDemand=demands[i];
    end;
run;

```



```

    estDemandLb = estDemand - 1.96*stds[i]/sqrt(10000);
    estDemandUb = estDemand + 1.96*stds[i]/sqrt(10000);
    output;
end;
keep price estDemand estDemandLb estDemandUb;
run;

```

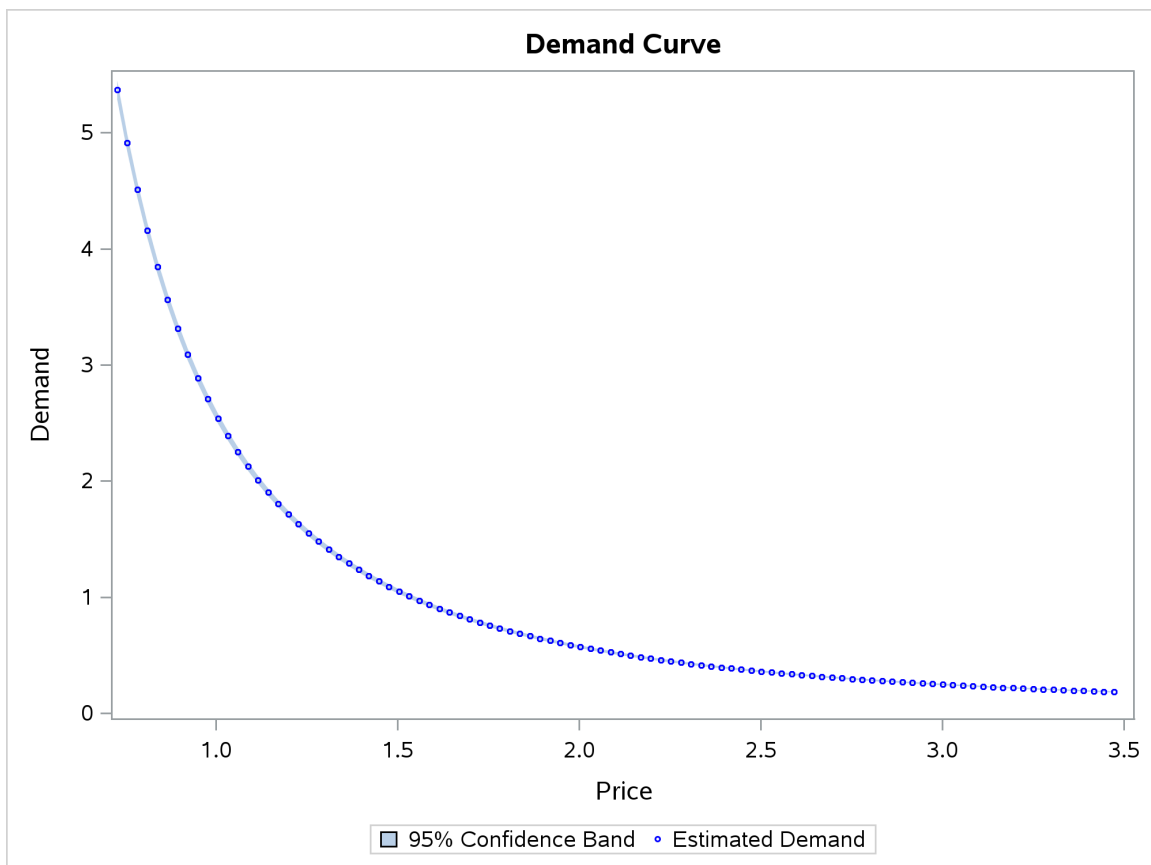
Then the demand curve and 95% pointwise confidence intervals are displayed as shown in [Output 16.2.4](#), using the following statements:

```

proc sgplot data=estCurve_log_if_ic;
  title 'Demand Curve';
  band x=price lower=estDemandLb upper=estDemandUb /
  legendlabel="95% Confidence Band " name="band";
  scatter x=price y=estDemand / markerattrs=(color=blue size=3 symbol=circle)
  legendlabel="Estimated Demand" name="estDemand";
  xaxis label='Price';
  yaxis label='Demand';
run;

```

Output 16.2.4 Estimated Demand Curve



The curve looks considerably better. As expected, the demand curve slopes downward. In fact, the data generating process (DGP) is available, and the objective of this section of the example is to assess how well the estimated demand curves approximate the truth.

The true DGP is defined as follows:

- η has a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.1$.
- The time (t) is continuous and has a uniform distribution from 0 to 12, exclusive.
- The customer type (s) is discrete and has an integer uniform distribution from 1 to 10, inclusive.
- The price (p) is continuous, and its value ranges from 0.7 to 3.5, inclusive.
- The demand (y) is continuous and nonnegative.

The treatment model is given as

$$p = \exp(1 + 0.3 * \zeta(t) + \eta)$$

$$\zeta(t) = 2 \left(\frac{50}{3} \left(\frac{t-6}{12} \right)^4 + \exp \left(-400 \left(\frac{t-6}{12} \right)^2 \right) + \frac{t}{12} - \frac{41}{20} \right)$$

where $\zeta(\cdot)$ describes the seasonality.

The outcome model is given as

$$y = \alpha(s, t) \exp(\log(p) \beta(s, t))$$

$$\alpha(s, t) = \exp(2 + 0.1 * s * \zeta(t))$$

$$\beta(s, t) = 0.1 * s * \zeta(t) - 1.5$$

The following DATA steps generate the data by using the true DGP:

```
data deepPriceDGP;
  array gridPrice[100] gridPrice1-gridPrice100;
  array gridY[100] gridY1-gridY100;
  call streaminit('pcg', 12440);
  do i = 1 to 10000;
    customertype = floor(rand('Uniform')*10+1);
    time = rand('Uniform')*12;
    eta = 0.1*rand('Normal',0,1);
    * z: seasonality;
    z = 2*(50/3*((time-6)/12)**4
      +exp(-400*((time-6)/12)**2)+time/12-41/20);
    price = exp(1+0.3*z + eta);
    a = exp(2 + 0.1*customertype*z);
    b = 0.1*customertype*z - 1.5;
    demand = a * exp(log(price) * b);
    do j = 1 to 100;
      gridPrice[j] = 0.7 + 2.8*j / 101;
      gridY[j] = a * exp( log(gridPrice[j]) * b);
    end;
  output;
```

```

    end;
run;

data mylib.deepPriceDataFull;
    set deepPriceDGP;
run;

```

Then, using the code segments that follow, you can plot the true and estimated demand curves.

First, the following code calculates the mean of the true price and demand:

```

proc univariate data=mylib.deepPriceDataFull noprint;
    var gridPrice1-gridPrice100 gridy1-gridy100;
    output out=trueCurve mean=gridPrice1-gridPrice100 gridym1-gridym100;
run;

data trueCurve2;
    set trueCurve;
    array prices[100] gridPrice1-gridPrice100;
    array demands[100] gridym1-gridym100;
    do i = 1 to 100;
        price=prices[i];
        demand=demands[i];
        output;
    end;
    keep price demand;
run;

```

The following code calculates the mean of estimated price and demand, and then the true and estimated demand curves and 95% pointwise confidence intervals are displayed in [Output 16.2.5](#) and [Output 16.2.6](#):

```

%macro createcurve(infile,outfile,bothfile);
    proc univariate data=&infile noprint;
        var gridPrice1-gridPrice100 gridy1-gridy100;
        output out=&outfile mean=gridPrice1-gridPrice100
            gridym1-gridym100 std=gridPrices1-gridPrices100 gridys1-gridys100;
    run;

    data &outfile%str(_mean);
        set &outfile;
        array prices[100] gridPrice1-gridPrice100;
        array demands[100] gridym1-gridym100;
        array stds[100] gridys1-gridys100;
        do i = 1 to 100;
            price=prices[i];
            estDemand=demands[i];
            estDemandLb = estDemand - 1.96*stds[i]/100;
            estDemandUb = estDemand + 1.96*stds[i]/100;
            output;
        end;
        keep price estDemand estDemandLb estDemandUb;
    run;

    data &bothfile;
        set trueCurve2;

```

```

    set &outfile%str(_mean);
run;

proc sgplot data=&bothfile;
    title 'True and Estimated Demand Curves';
    scatter x=price y=demand / markerattrs=(color=red size=5 symbol=trianglefilled)
    legendlabel="True Demand" name="demand";
    band x=price lower=estDemandLb upper=estDemandUb /
    legendlabel="95% Confidence Band " name="band";
    scatter x=price y=estDemand / markerattrs=(color=blue size=3 symbol=circle)
    legendlabel="Estimated Demand" name="estDemand";
    refline 0;
    xaxis label='Price';
    yaxis label='Demand';
run;
%mend;

```

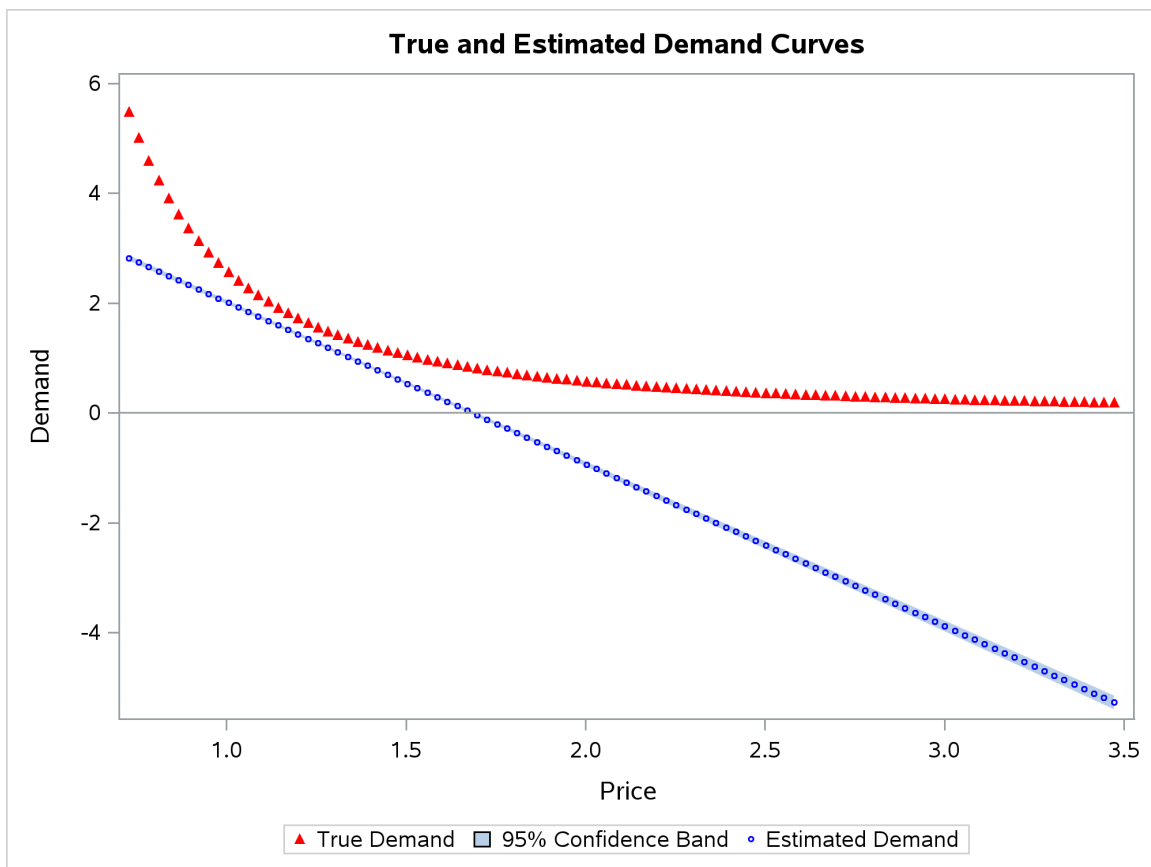
The results of using a linear demand curve assumption as follows are shown in [Output 16.2.5](#):

```

%let infile=mygrid_if;
%let outfile=estCurve_if;
%let bothfile=bothCurves_if;
%createcurve(&infile.,&outfile.,&bothfile.);

```

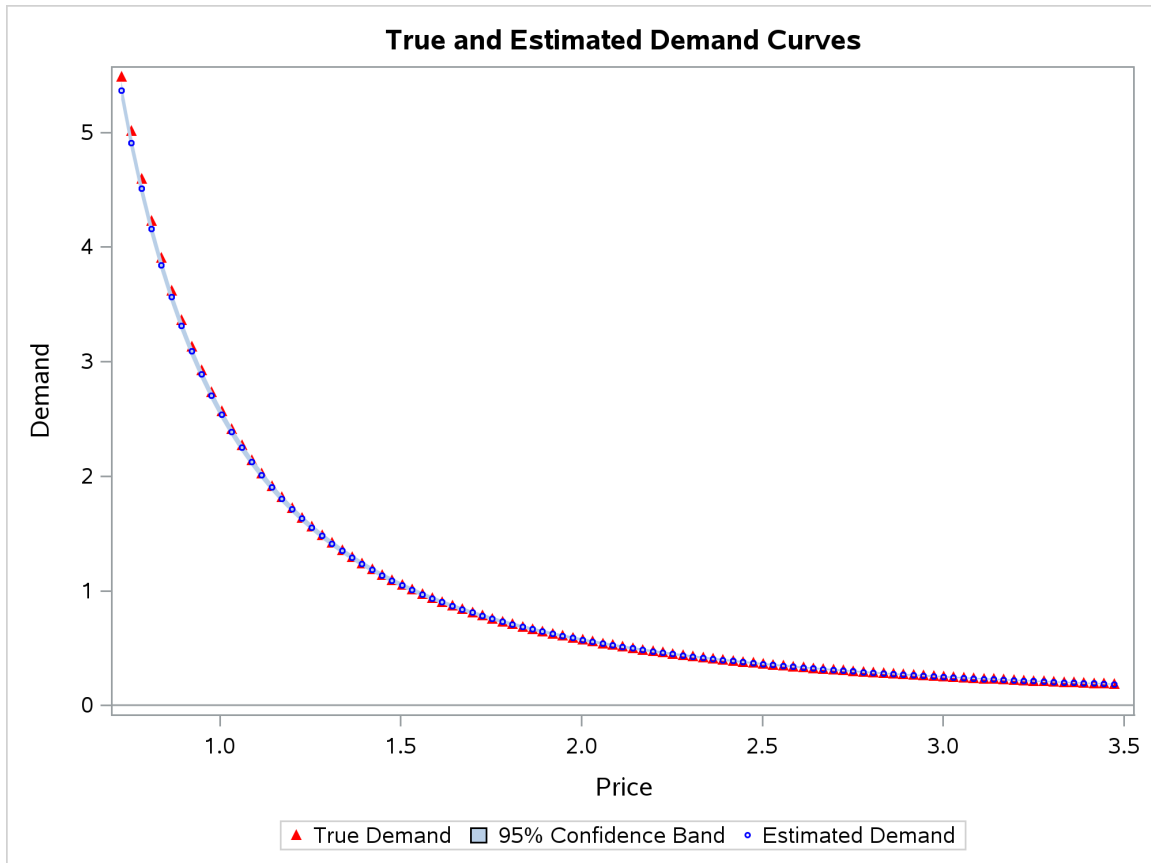
Output 16.2.5 True and Estimated Demand Curves



The results of using a log-log model as follows are shown in [Output 16.2.6](#):

```
%let infile=mygrid_log_if;
%let outfile=estCurve_log_if;
%let bothfile=bothCurves_log_if;
%createcurve(&infile.,&outfile.,&bothfile.);
```

Output 16.2.6 True and Estimated Demand Curves



The coverage rate of the 95% confidence interval for the linear curve is 0%, whereas that for the log-log curve is 93%. The linear demand curve assumption is clearly wrong for this example. However, in the real world, when there is no true demand curve or it is too expensive to get such a curve through randomized controlled trials (RCTs), how can you know what is the correct model specification? You can use the nonparametric framework, but that is beyond the scope of this example.

Considering that different types of customers and different purchase times can have an impact on the estimated results, the following code segments show how to produce the conditional demand curve. There are two situations: the first is the demand curve for different types of customers, and the second is the demand curve in different time periods. The following statements generate subsets of DGP based on various conditions:

```
%macro createtrue(n,where);
  %do j=1 %to &n;
    %let l=%eval(&j-1);
    proc univariate data=mylib.deepPriceDataFull(where=(&where)) noprint;
      var gridPrice1-gridPrice100 gridy1-gridy100;
```

```

        output out=trueCurve_&j mean=gridPrice1-gridPrice100
        gridym1-gridym100;
run;

data trueCurve2_&j;
  set trueCurve_&j;
  array prices[100] gridPrice1-gridPrice100;
  array demands[100] gridym1-gridym100;
  do i = 1 to 100;
    price=prices[i];
    demand=demands[i];
    output;
  end;
  keep price demand;
run;
%end;
%mend;

```

The following statements generate subsets of estimated price and demand data sets under various conditions:

```

%macro createdata(infile,outfile,bothfile,k);
  proc univariate data=&infile noprint;
    var gridPrice1-gridPrice100 gridy1-gridy100;
    output out=&outfile mean=gridPrice1-gridPrice100
    gridym1-gridym100
    std=gridPrices1-gridPrices100 gridys1-gridys100;
  run;

  data "&outfile%scan(_2,-1, ' ') "n;
    set &outfile;
    array prices[100] gridPrice1-gridPrice100;
    array demands[100] gridym1-gridym100;
    array stds[100] gridys1-gridys100;
    do i = 1 to 100;
      price=prices[i];
      estDemand=demands[i];
      estDemandLb = estDemand - 1.96*stds[i]/sqrt(10000);
      estDemandUb = estDemand + 1.96*stds[i]/sqrt(10000);
      output;
    end;
    keep price estDemand estDemandLb estDemandUb;
  run;

  data &bothfile;
    set trueCurve2_&k;
    set "&outfile%scan(_2,-1, ' ') "n;
  run;
%mend;

```

The following statements merge the data sets and plot the demand curves under various conditions:

```

%macro createcurve(n,where,title,cols,rows,type);
  %do k=1 %to &n;
    %let l=%eval(&k-1);
    data mygrid_log_if_&k;
      set mygrid_log_if(where=(&where));
    run;
    %let infile=mygrid_log_if_&k;
    %let outfile=estCurve_log_if_&k;
    %let bothfile=bothCurves_log_if_&k;
    %createdata(&infile.,&outfile.,&bothfile.,&k);

    data bothCurves_log_if_&k;
      set bothCurves_log_if_&k;
      &type=&k;
    run;
    %if &k=1 %then %do;
      data bothCurves_log_if_all;
        set bothCurves_log_if_&k;
      run;
    %end;
    %if &k>1 %then %do;
      data bothCurves_log_if_all;
        merge bothCurves_log_if_all bothCurves_log_if_&k;
        by &type;
      run;
    %end;
  %end;

  %let varname=;

  %if &type=time %then %do;
    data bothCurves_log_if_all;
      set bothCurves_log_if_all;
      month=put(time,NLSTRMON3.1);
    run;
    %let type=time month;
    %let varname=novarname;
  %end;

  proc sgpanel data=bothCurves_log_if_all;
    title "&title";
    panelby &type /layout=panel columns=&cols rows=&rows uniscale=column &varname;
    scatter x=price y=demand / markerattrs=(color=red size=5 symbol=trianglefilled)
    legendlabel="True Demand" name="demand";
    band x=price lower=estDemandLb upper=estDemandUb /
    legendlabel="95% Confidence Band " name="band";
    scatter x=price y=estDemand / markerattrs=(color=blue size=3 symbol=circle)
    legendlabel="Estimated Demand" name="estDemand";
    refline 1.5 /AXIS=X lineattrs=( color=green thickness=1);
    refline 1 /AXIS=Y lineattrs=( color=green thickness=1);
    colaxis label='Price';
    rowaxis label='Demand' values=(0 to 10 by 1) ;
  run;

```

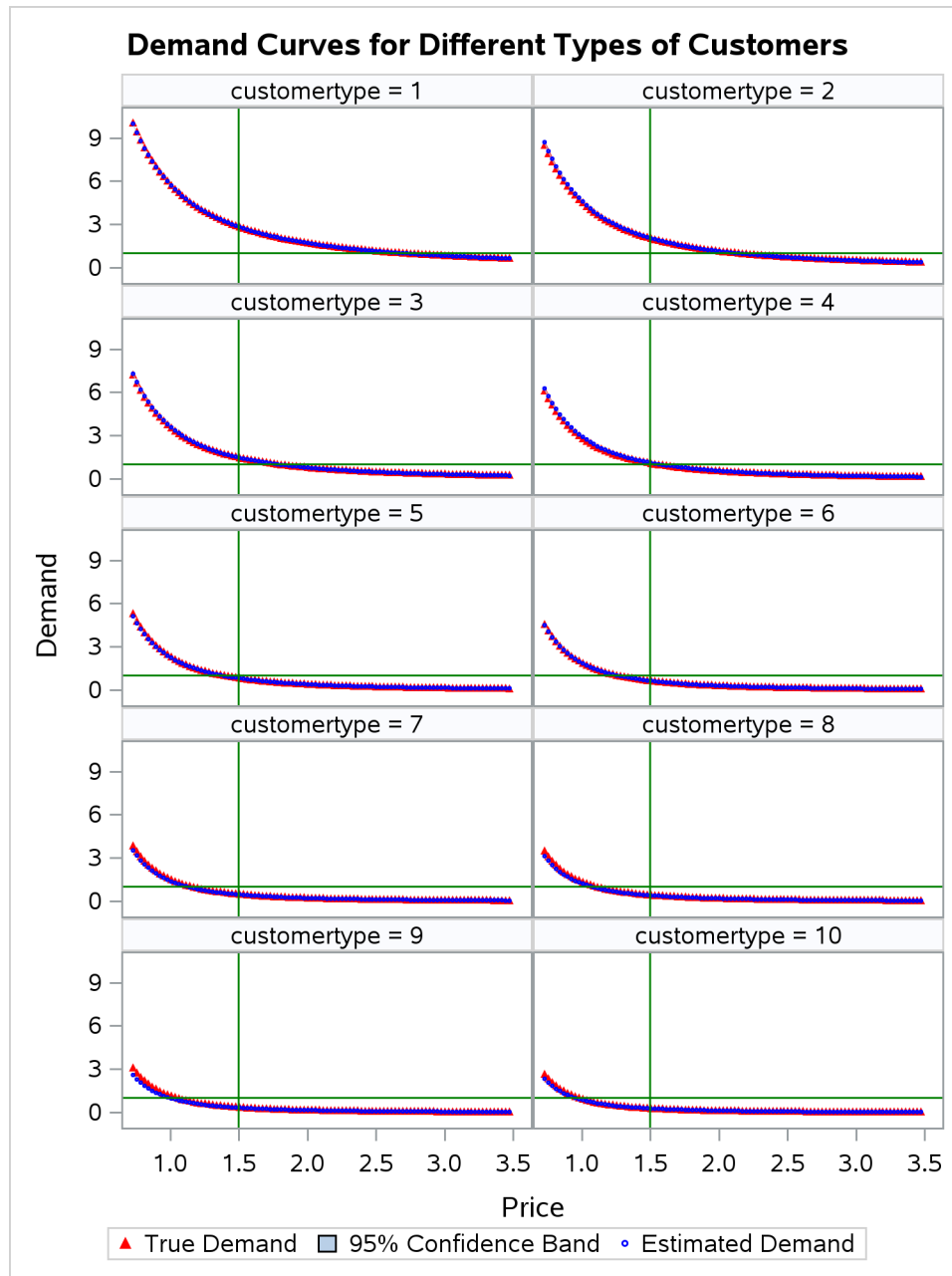
```
%mend;
```

The following statements pass parameters to the macro, select criteria for classification, add a title, and format the display:

```
%let n=10;
%let where= customertype %quote(=)%str(&j);
%createtrue(&n,&where);
%let where= customertype %quote(=)%str(&k);
%let title=Demand Curves for Different Types of Customers;
%let cols=2;
%let rows=5;
%let type=customertype;
%createcurve(&n,&where,&title,&cols,&rows,&type);
```

The results are shown in [Output 16.2.7](#)

Output 16.2.7 True and Estimated Condition Price vs. Demand Curves



Different types of customers have varying degrees of sensitivity to prices. Customers of customertype1 and customertype2 are the most price-sensitive, especially when price < 1.5. Customers of customertype4 and customertype5 are somewhat price-sensitive, whereas customers of customertype7, customertype8, customertype9, and customertype10 are relatively price-insensitive.

The following statements pass parameters to the macro, select criteria for purchasing time, add a title, and format the display:

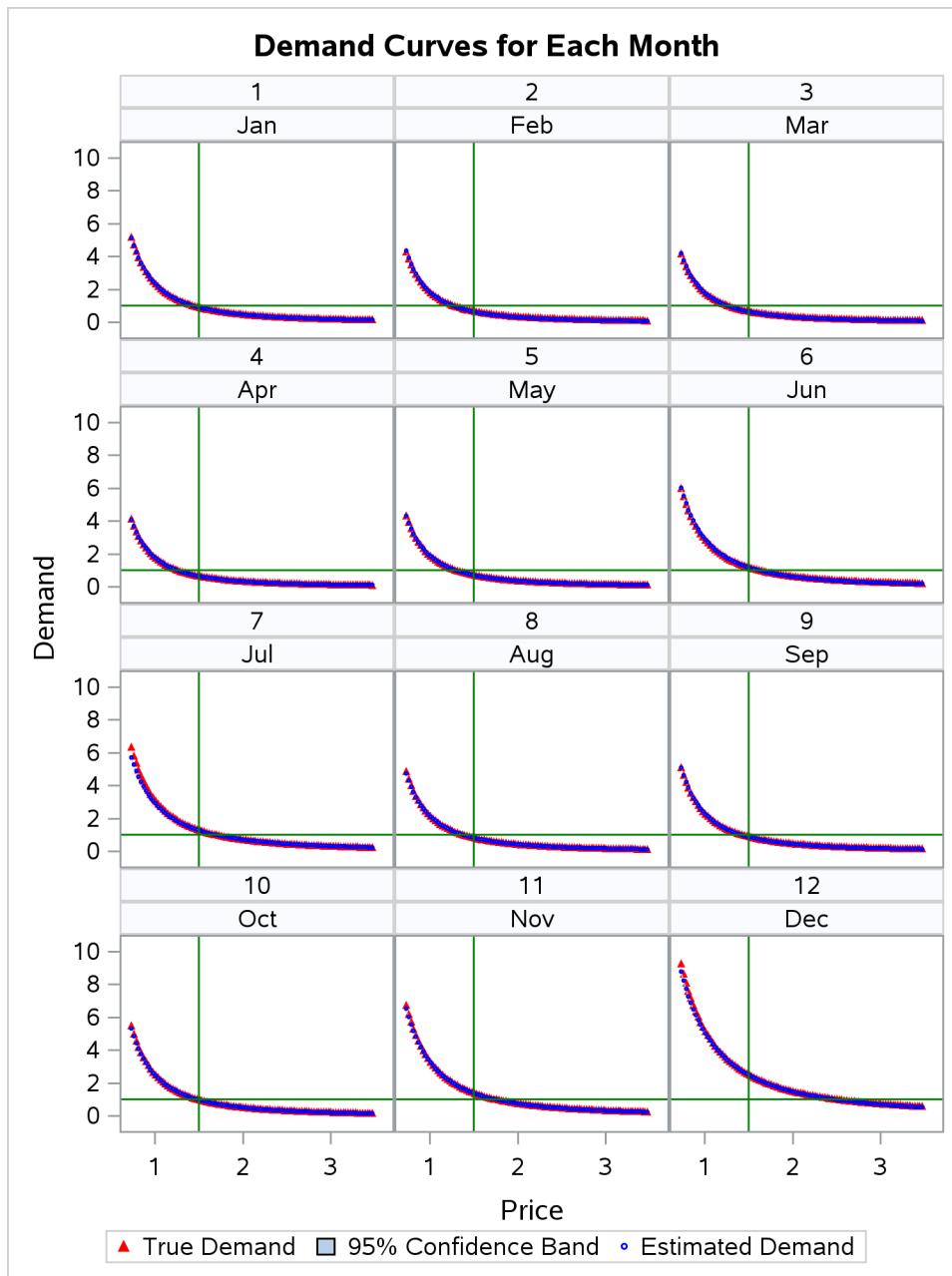
```

%let n=12;
%let where=time%quote(>=)%str(&)1 and time%quote(<=)%str(&)j;
%createtrue(&n, &where)
%let where=time%quote(>=)%str(&)1 and time%quote(<=)%str(&)k;
%let title=Demand Curves for Each Month;
%let type=time;
%let cols=3;
%let rows=4;
%createcurve(&n, &where, &title, &cols, &rows, &type);

```

The results are shown in [Output 16.2.8](#):

Output 16.2.8 True and Estimated Condition Demand Curves



The price sensitivity varies in different months. In February, March, April, and May, customers are not very sensitive to price changes, but in November and December, customers are very sensitive to price changes. The closer to the end of the year, the higher the price sensitivity. At the beginning of the year, the price sensitivity is relatively low.

In summary, only by specifying the correct functional form in PROC DEEPPRICE can you correctly estimate the demand curve. The shape of the estimated demand curves varies across customer types and over time.

References

Farrell, M. H., Liang, T., and Misra, S. (2020). “Deep Learning for Individual Heterogeneity.” <https://arxiv.org/abs/2010.14694>.

Chapter 17

DYNAMICLINEAR Procedure

Contents

Overview: DYNAMICLINEAR Procedure	1028
Using CAS Sessions and CAS Engine Librefs	1028
Overview of Dynamic Linear Models	1029
Getting Started: DYNAMICLINEAR Procedure	1030
Simultaneous Graphical Dynamic Linear Models	1030
Syntax: DYNAMICLINEAR Procedure	1037
Functional Summary	1037
PROC DYNAMICLINEAR Statement	1040
DISCOUNTFACTOR Statement	1041
FILTER Statement	1043
FORECAST Statement	1047
GPU Statement	1048
ID Statement	1049
INITIALDISTRIBUTION Statement	1049
MODEL Statement	1052
PARENTALSET Statement	1052
Details: The DYNAMICLINEAR Procedure	1053
Simultaneous Graphical Dynamic Linear Models	1053
Multiple Representations of the Adjacency Matrix	1055
Discount Factors in Dynamic Linear Models	1057
Memory Considerations in Simultaneous Graphical Dynamic Linear Models	1058
Parental Sets of Simultaneous Graphical Dynamic Linear Models	1059
Precision of GPU Computation in PROC DYNAMICLINEAR	1059
Data Table Input	1060
Data Table Output	1063
Examples: DYNAMICLINEAR Procedure	1068
Example 17.1: Tracking Volatility with Simultaneous Graphical Dynamic Linear Models	1068
Example 17.2: Comparing CPU and GPU Workflows in Simultaneous Graphical Dynamic Linear Models	1077
References	1095

Overview: DYNAMICLINEAR Procedure

Dynamic linear models (DLMs) are a cornerstone in the analysis of time series data. Their versatility has led to their widespread application in diverse fields such as economics, finance, science, and engineering. One of the primary SAS tools for working with DLMs is the DYNAMICLINEAR procedure, which focuses on simultaneous graphical dynamic linear models (SGDLMs) in particular.

PROC DYNAMICLINEAR offers a number of features, including the following:

- the ability to specify the model structure, enabling users to define the relationship between variables
- a provision for setting initial values for prior distributions, which is pivotal in Bayesian time series analysis
- detailed output options, which include filtering results. These help you refine and adjusting the model for better predictions.
- comprehensive forecasting tools, which deliver the mean, standard error, covariance matrix, median, and confidence intervals for forecasts of variables. Such in-depth statistics provide a holistic view of potential future scenarios.
- support for multistep forecasts after each observation, which enhance the granularity and flexibility of prediction

Because PROC DYNAMICLINEAR runs on SAS Cloud Analytic Services (CAS), it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Overview of Dynamic Linear Models

Dynamic linear models (DLMs) emerged in the early 1960s and are intricately linked with state space models (SSMs). Petris, Petrone, and Campagnoli (2009) stated that DLMs held an advantage over the then-popular ARMA models by providing more flexibility in handling nonstationary data, modeling structural changes, and offering increased interoperability.

At the core of DLMs is the idea of considering the outputs of a dynamic system as functions of unobservable or latent state processes that are influenced by random disturbances. By conditioning on these latent variables, DLMs can effectively capture the dependencies within data. A standout feature of DLMs is their Bayesian structure, which facilitates recursive computation. This capability ensures that conditional distributions can be seamlessly updated to incorporate fresh data. Significantly, this can be achieved without the need to retain past data, making DLMs especially well suited for sequential data analysis, online learning, and handling voluminous data sets.

General one-dimension DLMs have the following format:

$$y_t = \mathbf{F}_t' \boldsymbol{\theta}_t + v_t$$

$$\boldsymbol{\theta}_t = \mathbf{G}_t \boldsymbol{\theta}_{t-1} + \mathbf{w}_t$$

where

- y_t is the observation at time t .
- $\mathbf{F}_t$ is the known $p \times 1$ constant or regressor matrix at time t .
- $\boldsymbol{\theta}_t$ is the $p \times 1$ state variable.
- v_t is the observation noise, where $v_t \sim N(0, v_t)$.
- $\mathbf{w}_t$ is the $p \times 1$ state evolution noise, or innovation, where $\mathbf{w}_t \sim N(0, \mathbf{W}_t)$.
- v_t and $\mathbf{w}_t$ are mutually independent.
- $\mathbf{G}_t$ is the known $p \times p$ state evolution matrix at time t .
- $\boldsymbol{\theta}_t \sim N(\mathbf{m}_t, \mathbf{C}_t)$, where $\boldsymbol{\theta}_t$ is independent of v_t and $\mathbf{w}_t$ for all t .

Getting Started: DYNAMICLINEAR Procedure

This section provides a basic example of using the DYNAMICLINEAR procedure.

Simultaneous Graphical Dynamic Linear Models

This example is designed to demonstrate the process of defining a simultaneous graphical dynamic linear model (SGDLM) within PROC DYNAMICLINEAR. SGDLMs, known for their advanced approach in handling dynamic, multivariate time series data, offer a robust framework for modeling complex interactions over time. For a deeper dive into the mechanics and applications of SGDLMs, including their unique strengths and how they can be effectively used in various scenarios, see the section “[Simultaneous Graphical Dynamic Linear Models](#)” on page 1053.

The following DATA step loads the sample data:

```
data mylib.test_data;
  input t MMDDYY10. y1 y2 y3 y4 y5 BEST12.;
  format t MMDDYY10. y1 y2 y3 y4 y5 BEST12.;
datalines;
01/03/2002  1.158633277 -0.165838131 -0.220000432  0.08356095  0.097975091
01/04/2002 -0.995606401 -0.557778703  0.677071567  0.854087381  1.521078958
01/07/2002  1.494689626  1.200269513 -0.757267367 -0.468820299  1.558016222
01/08/2002  0.336831213  0.21765148  0.178311073  0.718694403 -1.277531854
01/09/2002  2.605081293  0.088138021 -0.95800973  -0.440623956  0.027367858
01/10/2002  1.365290757  0.207851726 -0.778749273  0.483102741 -0.259593807
01/11/2002  1.287939549 -0.625462152 -0.844034609  0.841906761  0.108750805
01/12/2002 -0.951405914 -0.272690273 -0.261733034 -2.068829141  0.671949044
01/13/2002  0.679413851  1.228575289 -0.840836877  1.124017627  0.185222142
01/14/2002  0.012315804  1.449826749 -0.424428315 -0.841630952 -0.431210039
;
```

Parental sets play a pivotal role in the structure and effectiveness of SGDLMs. According to Gruber and West (2016) and Gruber and West (2017), SGDLMs are adept at extracting correlations among time series, largely because of their sophisticated structural design that involves parental sets. For a more comprehensive understanding of parental sets, including their definition and importance in SGDLMs, see the sections “[PARENTALSET Statement](#)” on page 1052 and “[Parental Sets of Simultaneous Graphical Dynamic Linear Models](#)” on page 1059. The following code produces the parental sets for the provided sample data, illustrating their configuration within the model:

```
data mylib.adj_table;
  input ChildID ParentID;
datalines;
2          1
3          1
3          2
4          2
2          3
4          3
2          4
```



```

3           4
3           5
4           5
1           5
;

```

In SGDLMS, it is assumed that the observations adhere to a normal-gamma distribution, and their progression is tracked using the Kalman filter. PROC DYNAMICLINEAR offers multiple methods of defining the initial posterior distribution of these models. The following SAS code shows how to create input data tables that determine the initial means and covariances for SGDLMS. These tables are subsequently used in the INITIALDISTRIBUTION statement to set up the model's initial state.

```

data mylib.init_mean;
  input varname $1-4   varid parmname $24-40   parmid parmvalue;
datalines;
y1           1   Const           1   0.008596271
y1           1   ParSet_1       2   0.041964947
y1           1   ParSet_2       3   0.367778311
y2           2   Const           1  -0.004223895
y2           2   ParSet_1       2  -0.002395241
y2           2   ParSet_2       3   0.022385176
y3           3   Const           1   0.003904836
y3           3   ParSet_1       2   0.047429031
y3           3   ParSet_2       3  -0.489922549
y4           4   Const           1  -0.05762658
y4           4   ParSet_1       2   0.075128833
y4           4   ParSet_2       3  -0.224003793
y5           5   Const           1  -0.104497114
y5           5   ParSet_1       2  -2.129695978
y5           5   ParSet_2       3   0.047091226
y5           5   ParSet_3       4  -0.31115146
;

data mylib.init_cov;
  input ParmValue_1 ParmValue_2 ParmValue_3 ParmValue_4 RowID;
datalines;
0.008068698   -0.000016642   -0.000191582   0   1
-0.000016642   0.040067528   -0.000022887   0   2
-0.000191582   -0.000022887   0.350903386   0   3
0.003999164   -6.19E-7   -3.51E-6   0   4
-6.19E-7   0.002287045   -6.4E-8   0   5
-3.51E-6   -6.4E-8   0.021373754   0   6
0.003699612   -8.63E-6   -0.000071039   0   7
-8.63E-6   0.04528632   -0.000021115   0   8
-0.000071039   -0.000021115   0.46764184   0   9
0.049313412   -0.000182135   -0.000713341   0  10
-0.000182135   0.07172781   -0.000022758   0  11
-0.000713341   -0.000022758   0.213791508   0  12
0.270791097   -0.001387886   -0.064982122   -0.019065019  13
-0.001387886   0.10090566   -0.000259538   -0.000076145  14
-0.064982122   -0.000259538   4.712605925   -0.003565199  15
-0.019065019   -0.000076145   -0.003565199   0.29606287  16
;

```

This initial setup is crucial, because it forms the foundation on which the model builds its subsequent forecasts and analyses. For a thorough understanding of how to configure the initial distribution in SGDLMs, see the section “[INITIALDISTRIBUTION Statement](#)” on page 1049.

The INITIALDISTRIBUTION statement enables you to continue analyzing a previously saved SGDLM. This is particularly effective when you use it along with the OUTFILTERINFO= option in the [FILTER statement](#). Together, these features allow for a seamless continuation of model analysis from a saved state, ensuring the integrity and consistency of the SGDLM’s progression over time.

DLMs use discount factors to regulate the proportion of state evolution error variance that is considered during the model’s evolution. For more information, see the section “[Discount Factors in Dynamic Linear Models](#)” on page 1057. The following code shows how to use PROC DYNAMICLINEAR to fit an SGDLM to a given data set. In this example, the discount factors are defined using the [DISCOUNTFACTOR statement](#).

```
proc dynamicLinear data=mylib.test_data;
  id t;
  model y1-y5;
  parentalSet inAdjacency=mylib.adj_table;
  filter out(percentiles=(2.5 50 97.5))=mylib.out_filter back=0;
  forecast out(percentiles=(2.5 50 97.5))=mylib.out_forecast lead=3;
  initialDistribution inMean=mylib.init_mean inCovariance=mylib.init_cov
    shape=(positions=(1 2 5) values=(20 30 40));
  discountFactor deltaGamma=(positions=(3 5) values=(0.997 0.998));
run;
```

Upon successful execution of the preceding code, you can review the model’s summary tables in [Figure 17.1](#), which provides insight into the fit and performance of the SGDLM.

Figure 17.1 Model Summary
The DYNAMICLINEAR Procedure

Model Information	
Number of Variables Used	5
Simulation Size	10000
Random Number Seed	12345
Forecast Lead	3
Filter Back	0
Filter Lead	1
Number of Observations	10
Number of Observations Used	10

Input Tables			
Name	Contents	Number of Rows	Number of Columns
TEST_DATA	Data	10	6
ADJ_TABLE	Adjacency Matrices	11	2
INIT_MEAN	Means	16	5
INIT_COV	Covariances (Full)	16	5

Output CAS Tables			
CAS Library	Name	Number of Rows	Number of Columns
CASUSER(chuaxu)	OUT_FORECAST	3	28
CASUSER(chuaxu)	OUT_FILTER	9	28

An example output of the forecasting results that the model generates is shown in [Figure 17.2](#). The following PROC PRINT code prints the forecasting statistics that are generated by the OUT= option in the FORECAST statement:

```
proc print data=mylib.out_forecast;
run;
```

Figure 17.2 Example Output of Forecasting Results

Obs	t	LeadStep	Mean_y1	Variance_y1	P2_5_y1	P50_y1	P97_5_y1
1	01/14/2002	1	0.0562287843	1.6740409231	-2.551030149	0.0617329767	2.5853554959
2	01/14/2002	2	0.051792414	1.689381942	-2.631142191	0.0690782901	2.6253935528
3	01/14/2002	3	0.0540313157	1.6909249168	-2.604198531	0.062165089	2.6275629885

Obs	Mean_y2	Variance_y2	P2_5_y2	P50_y2	P97_5_y2	Mean_y3	Variance_y3
1	0.0030048896	0.2458605001	-0.98092187	0.0012353502	0.9947060545	-0.019379404	0.6742365123
2	0.0026459188	0.2477405635	-0.975475003	0.0031644677	0.983248444	-0.015777041	0.6780195983
3	0.0027979918	0.2484346856	-0.987649683	0.0006723175	1.0001381185	-0.017778213	0.6803869184

Obs	P2_5_y3	P50_y3	P97_5_y3	Mean_y4	Variance_y4	P2_5_y4	P50_y4
1	-1.63017889	-0.021860637	1.6088152766	0.0249764176	0.9116031811	-1.917011353	0.033318787
2	-1.638780493	-0.019907368	1.6204697268	0.0278951803	0.9193195028	-1.870895488	0.0331119637
3	-1.653028696	-0.019050342	1.6139603159	0.0239834421	0.9381107425	-1.907374826	0.0305862902

Obs	P97_5_y4	Mean_y5	Variance_y5	P2_5_y5	P50_y5	P97_5_y5	RowID
1	1.8836982077	0.3963792387	0.6318790441	-1.128707987	0.3941573569	2.015804329	1
2	1.8975615687	0.3953798175	0.6299070777	-1.168920807	0.3929888396	1.9860243651	2
3	1.9175205501	0.3980515611	0.6431627646	-1.153080309	0.3955947084	2.0030555469	3

Additionally, an example of the filtering results that the model generates is shown in [Figure 17.3](#). The following PROC PRINT code prints the first three observations of the forecasting statistics that are generated by the OUT= option in the FILTER statement:

```
proc print data=mylib.out_filter(obs=3);
run;
```

Figure 17.3 Example Output of Filtering Results

Obs	t	LeadStep	Mean_y1	Variance_y1	P2_5_y1	P50_y1	P97_5_y1	Mean_y2
1	01/03/2002	1	-0.054443041	855.25543349	-3.049609881	0.0159366682	2.8734103027	0.1601617164
2	01/04/2002	1	-0.265953197	415.77670547	-2.158601034	0.032960854	2.2092917348	-0.030385622
3	01/07/2002	1	-0.363511487	428.55045744	-1.919494327	0.0009872112	1.9481959368	-0.064543628

Obs	Variance_y2	P2_5_y2	P50_y2	P97_5_y2	Mean_y3	Variance_y3	P2_5_y3
1	577.91896746	-1.052842183	-0.01847281	0.9765916156	-1.644242541	50108.636773	-4.496104256
2	2.8241685107	-0.997871327	-0.020501244	0.9419312213	0.4069338608	1480.2701993	-3.633721189
3	28.362208296	-0.920294573	-0.032285013	0.8703851427	0.4065960356	647.68497031	-2.075284273

Obs	P50_y3	P97_5_y3	Mean_y4	Variance_y4	P2_5_y4	P50_y4	P97_5_y4
1	0.0251924004	4.6038303985	0.9964143126	25474.95232	-3.679928735	-0.056030822	3.630336531
2	0.0148692715	3.3896686134	-0.271851459	520.40410236	-2.839657981	-0.038934639	2.884275991
3	0.0086831118	2.164646309	0.1948211869	675.7931971	-1.908574062	0.0347232143	2.0284860639

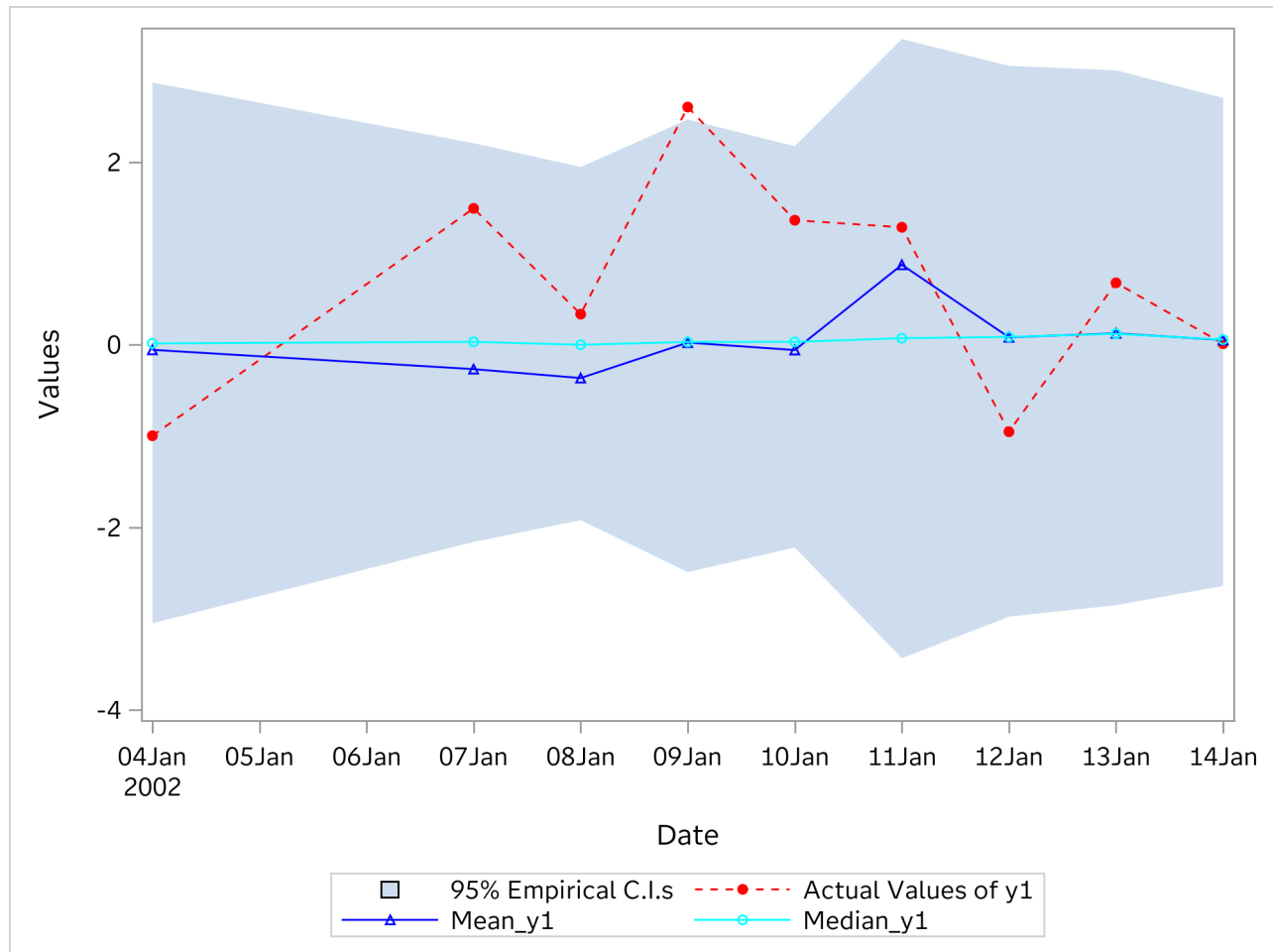
Obs	Mean_y5	Variance_y5	P2_5_y5	P50_y5	P97_5_y5	RowID
1	-0.481015551	41789.705442	-10.42614548	-0.091581671	10.867644377	1
2	-0.524440442	26815.660623	-13.4860025	0.8797552241	14.731493133	2
3	0.7692750656	1807.6712956	-3.121739776	1.1222731888	5.5148719972	3

The following SAS code is designed to create a visual representation of the empirical confidence interval for the variable y_1 . An empirical confidence interval offers an estimate of the range within which you expect to find the true value of y_1 a certain percentage of the time, according to the sample data. This code produces a plot that showcases the variability of y_1 , with the empirical confidence interval demarcated, thus providing a visual assessment of the precision of y_1 's estimated values within the model. The results are shown in Figure 17.4.

```
data y1_comb;
    format t MMDDYY10. Mean_y1 P2_5_y1 P50_y1 P97_5_y1 Best12.;
    set mylib.out_filter(keep=Mean_y1 P2_5_y1 P50_y1 P97_5_y1);
    set mylib.test_data(keep=t y1 where=(t>'03Jan2002'd));
run;

proc sort data=y1_comb;
    by t;
run;

proc sgplot data=y1_comb;
    band x=t upper=P2_5_y1 lower=P97_5_y1 / transparency=.3
        legendlabel='95% Empirical C.I.s';
    series x=t y=y1 / legendlabel='Actual Values of y1'
        markers markerattrs=(color=red size=5 symbol=circlefilled)
        lineattrs=(color=red pattern=shortdash);
    series x=t y=Mean_y1 / legendlabel='Mean_y1'
        markers markerattrs=(color=blue size=5 symbol=triangle)
        lineattrs=(color=blue pattern=solid);
    series x=t y=P50_y1 / legendlabel='Median_y1'
        markers markerattrs=(color=cyan size=5 symbol=circle)
        lineattrs=(color=cyan pattern=solid);
    label t="Date" y1="Values";
run;
```

Figure 17.4 Empirical Confidence Intervals of Variable y1

The prior example demonstrates how to specify parental sets, initial distributions, and discount factors in PROC DYNAMICLINEAR. It is important to note that these statements are optional, offering flexibility according to the user's needs and expertise. Additionally, for generating plots of confidence intervals, the data that are specified in the OUT= option of the FILTER statement are essential.

PROC DYNAMICLINEAR is designed with a high degree of flexibility to cater to a wide range of users. This includes those who are primarily interested in forecasting outcomes but might not have extensive knowledge of model configuration. For such users, a simpler and more concise code setup can be just as effective. The following code is an example of such a streamlined configuration, which is aimed at maximizing ease of use and accessibility:

```
proc dynamicLinear data=mylib.test_data;
  id t;
  model y1-y5;
  parentalSet inAdjacency=mylib.adj_table;
  forecast out=mylib.out_forecast lead=3;
run;
```

This simplified approach enables users to take advantage of the powerful forecasting capabilities of PROC DYNAMICLINEAR without the need to delve into more complex configuration details.

Syntax: DYNAMICLINEAR Procedure

The following statements are available in the DYNAMICLINEAR procedure. Items within angle brackets (<>) are optional.

```
PROC DYNAMICLINEAR DATA=libref.data-table < options > ;
  DISCOUNTFACTOR < options > ;
  FILTER < options > ;
  FORECAST < options > ;
  GPU < options > ;
  ID variable ;
  INITIALDISTRIBUTION < options > ;
  MODEL variable-list ;
  PARENTALSET < options > ;
```

Functional Summary

The statements and options available in the DYNAMICLINEAR procedure are summarized in [Table 17.1](#).

Table 17.1 Functional Summary

Description	Statement	Option
Input Data Table Options		
Specifies the input data table	PROC DYNAMICLINEAR	DATA=
Specifies the initial values for all discount factors	DISCOUNTFACTOR	INDISCOUNTFACTOR=
Specifies the initial covariance values by using a row-oriented table	INITIALDISTRIBUTION	INCOVARIANCE=
Specifies the initial variance values (diagonal) for the initial distribution's covariances	INITIALDISTRIBUTION	INCOVARIANCE(DIAGONAL)=
Specifies the initial values for the shape and rate parameters of the normal-gamma distribution	INITIALDISTRIBUTION	INGAMMA=
Specifies the initial values of means by using a column-oriented table	INITIALDISTRIBUTION	INMEAN=
Specifies the adjacency matrix of the parental sets	PARENTALSET	INADJACENCY=
Output Data Table Options		
Specifies the table to use for validating the input discount factors	DISCOUNTFACTOR	OUTVERIFICATION=
Specifies the output table for the covariances of prior/naive posterior/posterior normal-gamma distributions	FILTER	COVARIANCE=
Specifies the output table for the shape and rate parameters of prior/naive posterior/posterior normal-gamma distributions	FILTER	GAMMA=

Table 17.1 *continued*

Description	Statement	Option
Specifies the output table for Kullback-Leibler divergence information in the decoupling step of updating the distribution from naive posterior to posterior	<code>FILTER</code>	<code>KLDIVERGENCE=</code>
Specifies the output table for raw random samples of precisions during forecasting/importance sampling	<code>FILTER</code>	<code>LAMBDA=</code>
Specifies the output table for the means of prior/naive posterior/posterior normal-gamma distributions	<code>FILTER</code>	<code>MEAN=</code>
Specifies the output table for raw random samples of observation errors during forecasting	<code>FILTER</code>	<code>NU=</code>
Writes in-sample forecasting results and statistics to an output table	<code>FILTER</code>	<code>OUT=</code>
Writes the covariances of samples of in-sample forecasting observations to an output table	<code>FILTER</code>	<code>OUTCOVARIANCE=</code>
Writes samples of in-sample forecasting observations to an output table	<code>FILTER</code>	<code>OUTSIM=</code>
Specifies the output table for raw random samples of state vectors during forecasting/importance sampling	<code>FILTER</code>	<code>THETA=</code>
Writes out-of-sample forecasting results and statistics to an output table	<code>FORECAST</code>	<code>OUT=</code>
Writes the covariances of samples of out-of-sample forecasting observations to an output table	<code>FORECAST</code>	<code>OUTCOVARIANCE=</code>
Writes samples of out-of-sample forecasting observations to an output table	<code>FORECAST</code>	<code>OUTSIM=</code>
Specifies the output table for the covariances of the initial distribution	<code>INITIALDISTRIBUTION</code>	<code>COVARIANCE=</code>
Specifies the output table for the shape and rate parameters of the initial distribution	<code>INITIALDISTRIBUTION</code>	<code>GAMMA=</code>
Specifies the output table for the means of the initial distribution	<code>INITIALDISTRIBUTION</code>	<code>MEAN=</code>
Specifies the output table to use for verifying the input adjacency matrices	<code>PARENTALSET</code>	<code>OUTVERIFICATION=</code>
Common Options		
Specifies the number of simulations in filtering and forecasting	<code>PROC DYNAMICLINEAR</code>	<code>NSIMULATIONS=</code>
Specifies a nonnegative integer to use as the seed for generating random number sequences	<code>PROC DYNAMICLINEAR</code>	<code>SEED=</code>

Table 17.1 *continued*

Description	Statement	Option
ID Variable		
Identifies observations in the input data table by specifying a variable for the time series data	ID	
Variable and Parameter Declaration Options		
Specifies the model's stochastic volatility discount factors	DISCOUNTFACTOR	BETA=
Specifies the model's discount factors for regressors in the parental sets	DISCOUNTFACTOR	DELTAGAMMA=
Specifies the model's discount factors for regressors not in the parental sets	DISCOUNTFACTOR	DELTAPHI=
Specifies the starting covariance settings for the initial distribution	INITIALDISTRIBUTION	COVARIANCE=
Specifies the default initial value for all means	INITIALDISTRIBUTION	MEAN=
Specifies the starting setting for the initial distribution's rate parameter	INITIALDISTRIBUTION	RATE=
Specifies the starting setting for the initial distribution's shape parameter	INITIALDISTRIBUTION	SHAPE=
Specifies the variables to use for the model	MODEL	
Inference Options		
Specifies the number of observations before the end of the data at which the forecasts begin	FILTER	BACK=
Specifies the number of the multistep forecast values to compute during filtering	FILTER	LEAD=
Specifies which parameters of the naive posterior distribution to output	FILTER	NAIVEPOSTERIOR=
Specifies which filtering information to write to output tables	FILTER	OUTFILTERINFO=
Specifies which parameters of the posterior distribution to output	FILTER	POSTERIOR=
Specifies which raw random samples to output during forecasting	FILTER	PREDICTIONSAMPLE=
Specifies which parameters of the prior distribution to output	FILTER	PRIOR=
Specifies which raw random samples to output during the recoupling of importance sampling	FILTER	RECOUPLINGSAMPLE=
Specifies the number of the multistep forecast values to compute during forecasting	FORECAST	LEAD=
GPU Options		
Specifies that the model perform calculations by using GPU hardware	GPU	

Table 17.1 *continued*

Description	Statement	Option
Specifies the computation precision that the model uses for calculations on GPU hardware	GPU	PRECISION=

PROC DYNAMICLINEAR Statement

PROC DYNAMICLINEAR DATA=*libref.data-table* < options > ;

The PROC DYNAMICLINEAR statement invokes the DYNAMICLINEAR procedure. You must specify the following option:

DATA=*libref.data-table*

TABLE=*libref.data-table*

names the input data table for PROC DYNAMICLINEAR to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

data-table specifies the name of the input data table.

For the sample data, see the section “Simultaneous Graphical Dynamic Linear Models” on page 1030.

NOTE: No missing data are allowed in the input data table. If missing data are detected, the procedure stops and returns an error message.

You can also specify the following *options*:

NSIMULATIONS=*number*

NSIM=*number*

specifies the number of simulations during both forecasting and the subsequent posterior update phase. The *number* must be a positive integer. By default, NSIMULATIONS=10000.

SEED=*number*

specifies a nonnegative integer as the seed for generating random number sequences. When the seed value is set to 0, random number streams are generated by using the time of day from the computer’s clock. A seed value greater than 0 ensures a consistent and reproducible random number sequence. By default, SEED=0.

DISCOUNTFACTOR Statement

DISCOUNTFACTOR <options> ;

The DISCOUNTFACTOR statement specifies parameters of discount factors. For more information, see the section “Discount Factors in Dynamic Linear Models” on page 1057.

You can use one of the following methods to specify the discount factors:

1. Default value: If you do not specify a value, a system-defined default value is applied to all variables.
2. Position-value number-list pair: You can provide a list in which each pair consists of a variable's position in the *variables*, followed by the desired discount factor.
3. INDISCOUNTFACTOR= option: You can assign parameters directly through an input data table that you specify in the INDISCOUNTFACTOR= option.

When you use multiple methods to specify the discount factors, they are applied in the preceding sequence: first, the default value is applied; then, any specified position-value number-list pairs override the default; and finally, any values that are provided in the input table supersede previous specifications.

You can specify the following *options*:

BETA=(suboptions)

specifies the volatility discount factor, which is applicable to all variables (series) that you specify in the MODEL statement. The values of discount factors must be between 0 and 1, exclusive.

You can specify the following *suboptions*:

DEFAULT=number

DEF=number

specifies the default BETA= option value for all variables in the MODEL statement. By default, DEFAULT=0.95.

POSITIONS=(number-list)

POS=(number-list)

designates the positions of the variables for which values differ from the default. Every number in the *number-list* must be a positive integer.

VALUES=(number-list)

VAL=(number-list)

specifies values that correspond to the POSITIONS= option list values. Every number in the *number-list* must be between 0 and 1, exclusive.

DELTAGAMMA=(suboptions)

specifies the discount factor for regressors (the independent or exogenous variables) that are in the parental sets.

You can specify the following *suboptions*:

DEFAULT=*number*

DEF=*number*

specifies the default DELTAGAMMA= option value for all variables in the MODEL statement. By default, DEFAULT=0.995.

POSITIONS=(*number-list*)

POS=(*number-list*)

designates the positions of the variables for which values differ from the default. Every number in the *number-list* must be a positive integer.

VALUES=(*number-list*)

VAL=(*number-list*)

specifies values that correspond to the POSITIONS= option list values. Every number in the *number-list* must be between 0 and 1, exclusive.

DELTAPHI=(*suboptions*)

specifies the model's discount factors for regressors (the independent or exogenous variables) that are not in the parental sets.

You can specify the following *suboptions*:

DEFAULT=*number*

DEF=*number*

specifies the default DELTAPHI= option value for all variables in the MODEL statement. By default, DEFAULT=0.995.

POSITIONS=(*number-list*)

POS=(*number-list*)

designates the positions of the variables for which values differ from the default. Every number in the *number-list* must be a positive integer.

VALUES=(*number-list*)

VAL=(*number-list*)

specifies values that correspond to the POSITIONS= option list values. Every number in the *number-list* must be between 0 and 1, exclusive.

INDISCOUNTFACTOR=*libref.data-table*

INDF=*libref.data-table*

specifies the input data table for discount factors. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

When you use this option to specify discount factors, the input data table must adhere to specific format requirements. For more information, see the section “[INDISCOUNTFACTOR= Data Table in the DISCOUNTFACTOR Statement](#)” on page 1060.

OUTVERIFICATION=*libref.data-table*

OUTVERIFY=*libref.data-table*

specifies the table to use for validating input discount factors. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

FILTER Statement

FILTER <options> ;

The FILTER statement specifies the outputs and parameters of filtering. You can specify the following options:

BACK=*number*

specifies the number of steps from the end of the data set at which to begin generating output for the **OUT=**, **OUTSIM=**, and **OUTFILTERINFO=** options. The *number* should be less than or equal to the total number of observations minus one and must be greater than zero. When **BACK=0**, the output starts from the beginning of the data set. By default, **BACK=1**, which initiates the output one step before the end of the data set.

LEAD=*number*

specifies the quantity of multistep forecast values to be calculated during the filtering process. The *number* must be a nonnegative integer. By default, **LEAD=1**, which executes a one-step-ahead forecast.

OUT<(**PERCENT=***numeric-list*)>=*libref.data-table*

OUT<(**PCT=***numeric-list*)>=*libref.data-table*

writes the sample means of predictions, the sample variances, and the associated percentiles during the filtering process to a designated output data table. **PERCENT=***numeric-list* is an optional component that denotes the desired percentile points. Every value in the *numeric-list* should be between 0 and 100, inclusive, with values separated by either commas or spaces. Notably, the values of 0 and 100 equate to the minimum and maximum values, respectively. By default, **PERCENT=2.5** and **97.5**, which represent the 2.5th and 97.5th percentile points, respectively. For more information, see the section “[OUT= Data Table Generated by the Statements](#)” on page 1065. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

Consistency in the results of percentiles aligns with the **PCTLDEF=4** criteria, as discussed in the section “Calculating Percentiles” (Chapter 3, *Base SAS Procedures Guide: Statistical Procedures*) in the UNIVARIATE procedure.

OUTCOVARIANCE=*libref.data-table*

OUTCOV=*libref.data-table*

writes the covariances of samples of in-sample forecasting observations to an output table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

NOTE: This table consumes memory resources, particularly in scenarios that involve multistep forecasting where the LEAD= option value is greater than 1 and the number of variables in the MODEL statement is large. The requirement for memory also increases when multiple steps at the end of the data are included for output, which occurs when the BACK= option value is 0 or greater than 1.

OUTFILTERINFO=(suboptions)

specifies the scope of filtering information to be exported to output tables. It includes the distribution parameters such as the prior, the naive posterior, and the posterior. Additionally, it encompasses the Kullback-Leibler divergence of posteriors, raw samples obtained during the prediction generation, and the recoupling of importance sampling. **NOTE:** Exercise caution when using the following *suboptions*, because they are memory-intensive.

You can specify the following *suboptions*:

KLDIVERGENCE=libref.data-table

KLDIV=libref.data-table

specifies the output table for Kullback-Leibler (KL) divergence information in the decoupling step of updating the distribution from naive posterior to posterior. The KL divergence is the indication of the closeness between the revised variational-Bayes-based posterior estimation and the true posterior. The larger the value, the closer the two distributions are. The range of the KL divergence is between 0 and 1, inclusive. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

NAIVEPOSTERIOR=(naiveposterior-options)

NAIVEPOST=(naiveposterior-options)

specifies which parameters of the naive posterior distribution to output.

You can specify the following *naiveposterior-options*:

COVARIANCE=libref.data-table

COV=libref.data-table

specifies the output table for the covariances of naive posterior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

GAMMA=libref.data-table

specifies the output table for the shape and rate parameters of naive posterior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

MEAN=*libref.data-table*

specifies the output table for the means of naive posterior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

POSTERIOR=(*posterior-options*)**POST=**(*posterior-options*)

specifies which parameters of the posterior distribution to output.

You can specify the following *posterior-options*:

COVARIANCE=*libref.data-table***COV=***libref.data-table*

specifies the output table for the covariances of posterior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

GAMMA=*libref.data-table*

specifies the output table for the shape and rate parameters of posterior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

MEAN=*libref.data-table*

specifies the output table for the means of posterior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

PREDICTIONSAMPLE=(*predictionsample-options*)**PREDSMPL=**(*predictionsample-options*)

specifies which raw random samples to output during forecasting. **NOTE:** 1. These tables require significant memory resources, particularly the table that stores samples of state vectors (a table that you specify in the *THETA= predictionsample-option*). 2. The *PREDSMPL=* option is not supported when you run the procedure and enable GPUs.

You can specify the following *predictionsample-options*:

LAMBDA=*libref.data-table*

specifies the output table for raw random samples of precisions during forecasting. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

NU=libref.data-table

specifies the output table for raw random samples of observation errors during forecasting. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

THETA=libref.data-table

specifies the output table for raw random samples of state vectors during forecasting. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

PRIOR=(prior-options)

specifies which parameters of the prior distribution to output.

You can specify the following *prior-options*:

COVARIANCE=libref.data-table**COV=libref.data-table**

specifies the output table for the covariances of prior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

GAMMA=libref.data-table

specifies the output table for the shape and rate parameters of prior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

MEAN=libref.data-table

specifies the output table for the means of prior normal-gamma distributions. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

RECOUPLINGSAMPLE=(recouplingsample-options)**RECSMPL=(recouplingsample-options)**

specifies which raw random samples to output during the recoupling of importance sampling. **NOTE:** 1. These tables require significant memory resources, particularly the table that stores samples of state vectors (a table that you specify in the *THETA= prediction-sample-option*). 2. The RECSMPL= option is not supported when you run the procedure and enable GPUs.

You can specify the following *recouplingsample-options*:

LAMBDA=libref.data-table

specifies the output table for raw random samples of precisions during the recoupling of importance sampling. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

THETA=libref.data-table

specifies the output table for raw random samples of state vectors during the recoupling of importance sampling. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

OUTSIM=libref.data-table

writes samples of in-sample forecasting observations to an output table. These samples are generated according to the prior distribution and are used to compute the statistics in the [OUT=](#) output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

NOTE: This table requires substantial memory resources, particularly in scenarios that involve multi-step forecasting where the [LEAD=](#) option value is greater than 1. The requirement for memory also increases when multiple steps at the end of the data are included for output, which occurs when the [BACK=](#) option value is greater than 1 or when [BACK=0](#).

FORECAST Statement

FORECAST < options> ;

The FORECAST statement specifies the outputs and parameters of forecasting.

You can specify the following *options*:

LEAD=number

specifies the number of the multistep forecast values to compute, where *number* is a nonnegative integer. By default, [LEAD=1](#), which means that a one-step-ahead forecast is performed at the end of the available data.

OUT<(PERCENT=numeric-list)>=libref.data-table**OUT<(PCT=numeric-list)>=libref.data-table**

writes the sample means of predictions, the sample variances, and the associated percentiles during the forecasting process to a designated output data table. [PERCENT=numeric-list](#) is an optional component that denotes the desired percentile points. Every value in the *numeric-list* should be between 0 and 100, inclusive, with values separated by either commas or spaces. Notably, the values of 0 and 100 equate to the minimum and maximum values, respectively. By default, [PERCENT=2.5](#) and [97.5](#), which represent the 2.5th and 97.5th percentile points, respectively. For more information, see the section “[OUT= Data Table Generated by the Statements](#)” on page 1065. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more

information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

Consistency in the results of percentiles aligns with the PCTLDEF=4 criteria, as discussed in the section “Calculating Percentiles” (Chapter 3, *Base SAS Procedures Guide: Statistical Procedures*) in the UNIVARIATE procedure.

OUTCOVARIANCE=*libref.data-table*

OUTCOV=*libref.data-table*

writes the covariances of samples of out-of-sample forecasting observations to an output table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

NOTE: This table consumes memory resources, particularly in scenarios that involve multistep forecasting where the **LEAD=** option value is greater than 1 and the number of variables in the **MODEL** statement is large.

OUTSIM=*libref.data-table*

writes samples of out-of-sample forecasting observations to an output table. These samples are generated according to the prior distribution and are used to compute the statistics in the **OUT=** output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

NOTE: This table requires substantial memory resources, particularly in scenarios that involve multi-step forecasting where the **LEAD=** option value is greater than 1.

GPU Statement

GPU <option> ;

USEGPU <option> ;

The GPU statement specifies that the model perform calculations by using graphics processing units (GPUs).

NOTE: The model reverts to CPU processing for efficiency when the value of the **NSIMULATIONS=** option in the **DYNAMICLINEAR** statement is less than 100.

You can specify the following *option*:

PRECISION=FP32 | TF32 | FP64

specifies the computation precision that the model uses for calculations on GPU hardware. The value must be FP32, TF32, or FP64. For more information, see the section “Precision of GPU Computation in PROC DYNAMICLINEAR” on page 1059. By default, **PRECISION=FP32**.

ID Statement

ID *variable* ;

The ID statement identifies observations in the input data table by specifying a variable for the time series data. The *variable* cannot have missing or duplicated values.

INITIALDISTRIBUTION Statement

INITIALDISTRIBUTION < *options* > ;

The INITIALDISTRIBUTION statement defines the initial parameters of the posterior normal-gamma distribution, which sets the starting point for the filtering process in the model. This involves specifying the mean vector, variance-covariance matrix, and other hyperparameters that are associated with the normal-gamma distribution to establish the prior beliefs about the state vector before any data are processed.

You can use one of the following methods to specify the parameters for the initial gamma distribution:

1. Default value: If you do not specify a value, a system-defined default value is applied to all variables.
2. Position-value number-list pair: You can provide a list in which each pair consists of a variable's position in the *variables*, followed by the desired gamma distribution parameter.
3. INGAMMA= option: You can assign parameters directly through an input data table that you specify in the INGAMMA= option.

When you use multiple methods to specify the gamma distribution parameters, they are applied in the preceding sequence: first, the default value is applied; then, any specified position-value number-list pairs override the default; and finally, any values that are provided in the input table supersede previous specifications.

You can specify the following *options*:

INCOVARIANCE=*libref.data-table*

INCOV=*libref.data-table*

specifies the covariance matrices for the state variables by inputting full matrices. **NOTE:** For the covariance matrices, you need to provide only the lower-triangular part of each full matrix. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

You can find an example input table for the INCOVARIANCE= option in the section “[Simultaneous Graphical Dynamic Linear Models](#)” on page 1030 and the format requirements in the section “[INCOVARIANCE= Data Table in the INITIALDISTRIBUTION Statement](#)” on page 1061.

INCOVARIANCE(DIAGONAL)=libref.data-table**INCOVARIANCE(DIAG)=libref.data-table**

specifies the diagonal elements of the covariance matrices, following the same structural guidelines as those for the INMEAN= option (see the section “[INMEAN= Data Table in the INITIALDISTRIBUTION Statement](#)” on page 1062). *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1028.

It is crucial to ensure that the inputs for the diagonal elements in the table are nonzero values. This is because zero values on the diagonal of a covariance matrix imply no variance, which is invalid in the context of defining a distribution. Variance must be greater than zero to reflect the inherent uncertainty and variability of the state variables.

You can find example inputs in the section “[INCOVARIANCE\(DIAGONAL\)= Data Table in the INITIALDISTRIBUTION Statement](#)” on page 1061.

INGAMMA=libref.data-table

specifies initial values for the shape and rate parameters of the normal-gamma distribution. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1028.

The input data table must adhere to specific format requirements. These are provided in the section “[INGAMMA= Data Table in the INITIALDISTRIBUTION Statement](#)” on page 1062.

INMEAN=libref.data-table

specifies initial values for means by using a column-oriented table. You can find an example input table for this option in the section “[Simultaneous Graphical Dynamic Linear Models](#)” on page 1030. The inclusion of varname and parmname in the table is optional, because they primarily serve to clarify the table. There are certain requirements for specifying the INMEAN= option. The requirements are provided in the section “[INMEAN= Data Table in the INITIALDISTRIBUTION Statement](#)” on page 1062. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1028.

MEAN=number

specifies the default initial value for all means within the normal-gamma distribution framework. By default, MEAN=0.

OUTVERIFICATION=(suboptions)**OUTVERIFY=(suboptions)**

outputs the initial distribution to tables for validation.

You can specify the following *suboptions*:

COVARIANCE=*libref.data-table*

COV=*libref.data-table*

specifies the output table for the covariances of the initial distribution. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

GAMMA=*libref.data-table*

specifies the output table for the shape and rate parameters of the initial distribution. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

MEAN=*libref.data-table*

specifies the output table for the means of the initial distribution. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

RATE=(*suboptions*)

specifies the initial values for the rate parameters of gamma priors that are associated with all variables (series) in the MODEL statement. For the rate parameters, you must specify positive numbers greater than 0.

You can specify the following *suboptions*:

DEFAULT=*number*

DEF=*number*

specifies the default rate parameter for all variables. By default, DEFAULT=5.

POSITIONS=(*number-list*)

POS=(*number-list*)

designates the positions of the variables for which values differ from the default. Every number in the *number-list* must be a positive integer.

VALUES=(*number-list*)

VAL=(*number-list*)

specifies values that correspond to the POSITIONS= option list values. Every number in the *number-list* must be greater than 0.

SHAPE=(*suboptions*)

specifies the initial values for the shape parameters of gamma priors that are associated with all variables (series) in the MODEL statement. For the rate parameters, specified values must be positive numbers greater than 1. These parameters are crucial, because they influence the skewness and variability of the prior distribution, thereby affecting the posterior estimates in Bayesian analysis.

You can specify the following *suboptions*:

DEFAULT=*number*

DEF=*number*

specifies the default shape parameter for all variables. By default, DEFAULT=10.

POSITIONS=(*number-list*)

POS=(*number-list*)

designates the positions of the variables for which values differ from the default. Every number in the *number-list* must be a positive integer.

VALUES=(*number-list*)

VAL=(*number-list*)

specifies values that correspond to the POSITIONS= option list values. Every number in the *number-list* must be greater than 1.

MODEL Statement

MODEL *variable-list* ;

The MODEL statement specifies the variables to use in the model. You can specify variables in a space-separated *variable-list*, and the order does not matter. The variables must be selected from the input data table (see the **DATA=** option in the **PROC DYNAMICLINEAR** statement).

PARENTALSET Statement

PARENTALSET < *options* > ;

The PARENTALSET statement specifies the parental set of the model. For more information, see the section “Parental Sets of Simultaneous Graphical Dynamic Linear Models” on page 1059.

You can specify the following *options*:

INADJACENCY=*libref.data-table*

INADJ=*libref.data-table*

specifies the input data table for the adjacency matrix, which delineates the relationships within parental sets. This option supports multiple ways to specify the adjacency matrix. For more information, see the section “Multiple Representations of the Adjacency Matrix” on page 1055. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

OUTVERIFICATION=*libref.data-table*

OUTVERIFY=*libref.data-table*

specifies the output table to use for verifying the input adjacency matrices. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1028.

Details: The DYNAMICLINEAR Procedure

Simultaneous Graphical Dynamic Linear Models

The classical dynamic linear model (DLM) often falters when confronted with high-dimensional time series that are characterized by intricate correlations. Recognizing this limitation, Gruber and West (2016) and Gruber and West (2017) introduced an enhanced version known as a simultaneous graphical dynamic linear model (SGDLM). This advanced model is adept at efficiently managing large-scale, correlated time series.

Consider an m -dimensional time series that is denoted by $\mathbf{y}_t = (y_{1t}, y_{2t}, \dots, y_{mt})'$. The SGDLM can be expressed as

$$\begin{aligned} y_{jt} &= \mathbf{F}'_{jt} \boldsymbol{\theta}_{jt} + v_{jt} = \mathbf{x}'_{jt} \boldsymbol{\phi}_{jt} + \mathbf{y}'_{\text{sp}(j),t} \boldsymbol{\gamma}_{jt} + v_{jt} \\ \boldsymbol{\theta}_{jt} &= \mathbf{G}_{jt} \boldsymbol{\theta}_{j,t-1} + \mathbf{w}_{jt} \end{aligned}$$

In this representation, $v_{jt} \sim N(0, \lambda_{jt}^{-1})$ and $\mathbf{w}_{jt} \sim N(\mathbf{0}, \mathbf{W}_{jt})$. The expression $\mathbf{y}_{\text{sp}(j),t}$ denotes a vector that represents values of other series indexed by $\text{sp}(j) \subseteq 1 : m \setminus j$, where $\text{sp}(j)$ denotes the simultaneous parental set of series j . The vector $\mathbf{x}_{jt} = (x_{j,1,t}, x_{j,2,t}, \dots)'$ denotes the vector of external predictors for series j at time t ; it is important to note that $\mathbf{x}_{jt}$ might vary in dimensions across the m variables.

First, here are the definitions of certain matrices and vectors that are related to SGDLMs:

- Let $\boldsymbol{\Theta}_t$ represent the collection of $\boldsymbol{\theta}_{jt}$ values for every j , formally written as

$$\boldsymbol{\Theta}_t = \{\boldsymbol{\theta}_{1t}, \boldsymbol{\theta}_{2t}, \dots, \boldsymbol{\theta}_{mt}\}$$

- Define $\boldsymbol{\Lambda}_t$ as a diagonal matrix that contains λ_{jt} values:

$$\boldsymbol{\Lambda}_t = \text{diag}\{\lambda_{1t}, \lambda_{2t}, \dots, \lambda_{mt}\}$$

- $\boldsymbol{\mu}_t$ is the vector given by

$$\boldsymbol{\mu}_t = (\mu_{1t}, \mu_{2t}, \dots, \mu_{mt})'$$

where the individual components are defined as $\mu_{jt} = \mathbf{x}'_{jt} \boldsymbol{\phi}_{jt}$.

The matrix $\boldsymbol{\Gamma}_t$ captures the relationships between various time series. It is defined as

$$\boldsymbol{\Gamma}_t = \begin{bmatrix} \boldsymbol{\gamma}'_{1t} \\ \boldsymbol{\gamma}'_{2t} \\ \vdots \\ \boldsymbol{\gamma}'_{mt} \end{bmatrix} = \begin{bmatrix} 0 & \gamma_{1,2,t} & \gamma_{1,3,t} & \cdots & \gamma_{1,m,t} \\ \gamma_{2,1,t} & 0 & \gamma_{2,3,t} & \cdots & \gamma_{2,m,t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma_{m,1,t} & \gamma_{m,2,t} & \cdots & \gamma_{m,m-1,t} & 0 \end{bmatrix}$$

where $\gamma_{j,h,t} = 0$ for each $h \notin \text{sp}(j)$, $j = 1 : m$.

Use these definitions to represent the distribution of $\mathbf{y}_t$ as

$$\mathbf{y}_t \sim N(\boldsymbol{\Lambda}_t \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$$

where the matrices $\mathbf{A}_t$ and $\mathbf{\Sigma}_t$ are given by

$$\mathbf{A}_t = (\mathbf{I} - \mathbf{\Gamma}_t)^{-1}, \quad \mathbf{\Omega}_t \equiv \mathbf{\Sigma}_t^{-1} = (\mathbf{I} - \mathbf{\Gamma}_t)' \mathbf{A}_t (\mathbf{I} - \mathbf{\Gamma}_t)$$

and where $\mathbf{\Omega}_t$ is the precision matrix.

Finally, PROC DYNAMICLINEAR integrates the local-level dynamic linear model with the simultaneous graphical feature, as follows:

$$\begin{aligned} y_{jt} &= \mathbf{F}'_{jt} \boldsymbol{\theta}_{jt} + v_{jt} = 1 \cdot \phi_{jt} + \mathbf{y}'_{\text{sp}(j),t} \boldsymbol{\gamma}_{jt} + v_{jt} \\ \boldsymbol{\theta}_{jt} &= \mathbf{I} \cdot \boldsymbol{\theta}_{j,t-1} + \mathbf{w}_{jt} \end{aligned}$$

where $\mathbf{F}_{jt} = (1, \mathbf{y}_{\text{sp}(j),t})'$ for all j , and $\mathbf{G}_{jt} = \mathbf{I}$ for all j and t .

Forward Filtering in SGDLMS

To understand the forward filtering process in an SGDLMS, you need to understand the distributions at play. Here are the key distributions that this process uses. For more information about these distributions, see Gruber and West (2016) and Gruber and West (2017).

- The prior at time t is given by

$$p(\boldsymbol{\Theta}_t, \mathbf{A}_t | \mathcal{I}_t, \mathcal{D}_{t-1}) = \prod_{j=1:m} p_{jt}(\boldsymbol{\theta}_{jt}, \lambda_{jt} | \mathcal{I}_t, \mathcal{D}_{t-1})$$

where $\mathcal{I}_t$ denotes the information set at time t ; $\mathcal{D}_{t-1}$ denotes the data set up to time $t - 1$; and $p(\cdot | \cdot)$ denotes the density function of

$$(\boldsymbol{\theta}_{jt}, \lambda_{jt} | \mathcal{I}_t, \mathcal{D}_{t-1}) \sim \text{NG}(\mathbf{a}_{jt}, \mathbf{R}_{jt}, r_{jt}, c_{jt})$$

and

$$\begin{aligned} (\boldsymbol{\theta}_{jt} | \lambda_{jt}, \mathcal{I}_t, \mathcal{D}_{t-1}) &\sim \text{N}(\mathbf{a}_{jt}, \mathbf{R}_{jt} / (c_{jt} \lambda_{jt})) \\ (\lambda_{jt} | \mathcal{I}_t, \mathcal{D}_{t-1}) &\sim \text{Gamma}(r_{jt}/2, r_{jt} c_{jt} / 2) \end{aligned}$$

where $\text{N}(\mathbf{a}, \mathbf{A})$ is a multivariate normal distribution that has the mean $\mathbf{a}$ and the covariance matrix $\mathbf{A}$, and $\text{Gamma}(\alpha, \beta)$ is a gamma distribution that has the shape parameter α and the rate parameter β .

- The naive posterior approximation at time t to the true posterior $p(\boldsymbol{\Theta}_t, \mathbf{A}_t | \mathcal{D}_t)$ is given by

$$\tilde{p}(\boldsymbol{\Theta}_t, \mathbf{A}_t | \mathcal{D}_t) = \prod_{j=1:m} \tilde{p}_{jt}(\boldsymbol{\theta}_{jt}, \lambda_{jt} | \mathcal{D}_t)$$

where

$$(\boldsymbol{\theta}_{jt}, \lambda_{jt} | \mathcal{D}_t) \sim \text{NG}(\tilde{\mathbf{m}}_{jt}, \tilde{\mathbf{C}}_{jt}, \tilde{n}_{jt}, \tilde{s}_{jt})$$

- The true posterior at time t is

$$p(\boldsymbol{\Theta}_t, \mathbf{A}_t | \mathcal{D}_t) \propto |\mathbf{I} - \mathbf{\Gamma}_t| \prod_{j=1:m} \tilde{p}_{jt}(\boldsymbol{\theta}_{jt}, \lambda_{jt} | \mathcal{D}_t) \propto \prod_{j=1:m} p_{jt}(\boldsymbol{\theta}_{jt}, \lambda_{jt} | \mathcal{D}_t)$$

with the components

$$(\boldsymbol{\theta}_{jt}, \lambda_{jt} | \mathcal{D}_t) \sim \text{NG}(\mathbf{m}_{jt}, \mathbf{C}_{jt}, n_{jt}, s_{jt})$$

Multiple Representations of the Adjacency Matrix

For the adjacency matrix Γ , the matrix entries $\Gamma(i, j)$ indicate whether variable j belongs to the parental set of variable i . These entries must be binary (0 or 1), and all diagonal elements must be set to zero; that is, $\Gamma(i, j)$ takes the value of 0 or 1, and for all i , $\Gamma(i, i) = 0$. Here, i and j range from 1 to n , where n is the total number of variables in the data set. The adjacency matrix is not required to be symmetric, implying that $\Gamma(i, j)$ does not necessarily equal $\Gamma(j, i)$.

The Γ matrix might be formatted in one of two ways:

- A single or multiple full adjacency matrices can be provided. For a single matrix, the data table should include all variables in the MODEL statement along with an additional variable VarID, which indexes the variables according to their order in the statement. When multiple matrices are present, the data table must also include the variable RowID for the purpose of ordering. If there are n matrices in the data table and m observations for which $m > n$, then each of the first $n - 1$ matrices correlates with the $n - 1$ observations, and the last matrix applies to the subsequent $m - n + 1$ observations. **NOTE:** For different adjacency matrices that correspond to distinct time points, it is imperative to maintain a consistent number of candidates within each variable's parental set across all time points. For example, if the variable y1 has two candidates in its parental set at the initial time point, it must retain a parental set of two throughout all subsequent time points. This consistency must be followed for all variables.
- A compact form of the adjacency matrix is also available for use when suitable. In this approach, the data table that you specify in the INADJACENCY= option in the PARENTALSET statement should have as many observations as there are nonzero entries in the adjacency matrix. This data table has precisely two columns, which record the row and column indices of the nonzero elements. You can find an example of the compact form input in the section “[Simultaneous Graphical Dynamic Linear Models](#)” on page 1030.

Compact Representation of the Adjacency Matrix

PROC DYNAMICLINEAR facilitates a more streamlined input process by allowing a compact representation of Γ . To use this compact form, specify a data table that contains just two variables via the INADJACENCY= option. In this efficient format, the number of rows in the data table, each of which corresponds to an observation, should match the total count of nonzero elements in Γ .

When you use this compact representation, the data table must include these two columns:

- ParentID: Identifies the variable that is considered the “parent” in the relationship.
- ChildID: Denotes variables that are “children,” meaning that they are part of the parent variable's parental set.

The values in ParentID and ChildID are not arbitrary but should directly correspond to the order of variables as they are listed in the MODEL statement.

As an illustration, consider a compact representation for an adjacency matrix Γ_1 ,

$$\Gamma_1 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Here is the code for how such a table would be structured:

```
data Gamma_1;
  input ParentID ChildID;
datalines;
1 2
1 4
2 1
3 4
4 1
4 3
;
```

In instances where a variable's parental set is empty, meaning that it has no members, two methods are available to denote this in the compact form of an adjacency matrix. For illustration, consider that the parental sets for the second and third variables (ParentID = 2 and ParentID = 3, respectively) are empty. You can take the following approaches:

1. Assign ChildID = 0 to the variable that has an empty parental set, which in this example would apply to ParentID = 2.
2. Omit the entire row that corresponds to the variable that has the empty set, as would be the case for ParentID = 3.

A compact code representation of the adjacency matrix Γ_2 ,

$$\Gamma_2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

is as follows:

```
data Gamma_2;
  input ParentID ChildID;
datalines;
1 2
1 4
2 0
4 1
4 3
;
```

Full Representation of the Adjacency Matrix

The following code is an example of a full adjacency matrix. In this matrix, the presence of a 1 indicates that the column variable is in the parental set of the row variable. For example, the variables y_2 and y_3 are in the parental set of the variable y_1 , the variables y_3 and y_4 are in the parental set of the variable y_2 , and the pattern continues accordingly.

```
data mylib.adjacency;
  input VarID y1 y2 y3 y4 y5;
datalines;
1 0 1 1 0 0
2 0 0 1 1 0
3 0 1 0 1 0
4 0 1 1 0 0
5 0 0 1 1 0
;
```

In scenarios where multiple adjacency matrices are required—such as for two distinct adjacency matrices—you can code the specification as follows:

```
data mylib.multi_adjacency;
  input VarID y1 y2 y3 y4 y5 RowID;
datalines;
1 0 1 1 0 0 1
2 0 0 1 1 0 2
3 0 1 0 1 0 3
4 0 1 1 0 0 4
5 0 0 1 1 0 5
1 0 1 0 1 0 6
2 0 0 1 0 1 7
3 1 0 0 1 0 8
4 1 0 1 0 0 9
5 0 1 1 1 0 10
;
```

Discount Factors in Dynamic Linear Models

In accordance with the exposition of Petris, Petrone, and Campagnoli (2009), the discount factor δ signifies the proportion of the state evolution error variance to be accounted for during the transition from the posterior to the prior distribution. The state evolution is mathematically represented as

$$\theta_t = \mathbf{G}_t \theta_{t-1} + \mathbf{w}_t \implies \mathbf{R}_t = \mathbf{G}_t' \mathbf{C}_{t-1} \mathbf{G}_t + \mathbf{W}_t$$

where θ_t is the state at time t , $\mathbf{G}_t$ is the state transition matrix, $\mathbf{w}_t$ is the process noise, $\mathbf{R}_t$ is the covariance of the prior normal-gamma distribution at time t , and $\mathbf{C}_{t-1}$ is the covariance of the posterior at time $t - 1$. This leads to an interpretation of $\mathbf{W}_t$ as a fraction of the term $\mathbf{G}_t' \mathbf{C}_{t-1} \mathbf{G}_t$:

$$\mathbf{W}_t = \frac{1 - \delta}{\delta} \mathbf{G}_t' \mathbf{R}_{t-1} \mathbf{G}_t \implies \mathbf{R}_t = \frac{1}{\delta} \mathbf{G}_t' \mathbf{C}_{t-1} \mathbf{G}_t$$

which suggests that $\mathbf{W}_t$ adjusts the covariance of the state evolution error to reflect the selected proportion of variance.

Petris, Petrone, and Campagnoli (2009) further note that, in practical settings, the discount factor is typically selected within the range of 0.9 to 0.99. A value closer to 1 indicates a higher confidence in the model's stability over time, whereas a value closer to 0.9 suggests greater variability and less certainty in the persistence of the state's dynamics.

The stochastic volatility discount factor, which is denoted by β , plays a critical role in the evolution of the model by updating the precision λ_{t-1} of the observation error v as you transition from the posterior to the prior distribution. To illustrate, if the gamma distribution of the posterior normal-gamma distribution at time $t - 1$ is characterized by a shape parameter n_{t-1} and a rate parameter s_{t-1} , then the transition that is influenced by β yields the parameters of the prior gamma distribution at time t as $n_t = \beta \times n_{t-1}$ for the shape and $s_t = s_{t-1}$ for the rate. This reflects how β modulates the degree of belief that is carried over from one time step to the next in the sequence of observations.

Memory Considerations in Simultaneous Graphical Dynamic Linear Models

When you work with SGDLMs, it is crucial to understand and plan for memory requirements, because these requirements scale with the size of the problem. For example, consider modeling a system that contains 400 variables. The adjacency matrix $\mathbf{\Gamma}_t$ and the precision matrix $\mathbf{\Omega}_t$, which are pivotal for importance sampling and prediction generation, would each have dimensions of 400×400 , totaling 160,000 elements per matrix. Storing each element in double precision, which is standard practice and requires 8 bytes per element, means that a single matrix consumes approximately 1.28 MB of memory, calculated by $160,000 \times 8$ bytes.

In Monte Carlo simulations that encompass 10,000 iterations, storing all adjacency and precision matrices would require approximately 25.6 GB of memory, which is twice the product of 1.28 MB per matrix and 10,000 iterations, or 12.8 GB multiplied by 2. This figure does not account for the memory consumption during other steps. Adequate memory resources are paramount. If you have insufficient memory, PROC DYNAMICLINEAR cannot run.

If memory constraints present an issue, you might need to reduce the number of variables being modeled, shorten the length of time series, or decrease the number of simulations. Each of these adjustments comes with trade-offs: reducing the number of variables and shortening series length might oversimplify the model, whereas fewer simulations could compromise the statistical robustness of the forecast results. Therefore, it is essential to find an optimal point that meets the memory limitations without significantly compromising the model's integrity and results.

Generating detailed output from the filtering and forecasting process demands significant memory resources, because these data must be retained in memory prior to being recorded in tables upon completion of the program's execution. Exercise caution when enabling such output options. For more information, see the document of the [FILTER statement](#) and the [FORECAST statement](#).

Currently, an SGDLM that contains approximately 350 variables is estimated to require about 50 GB of memory for processing on CPUs. Through careful design and optimized implementation, the GPU workflow of PROC DYNAMICLINEAR uses memory more efficiently than the CPU workflow. Using single-precision floating-point arithmetic (FP32) for GPU computations further reduces memory consumption while maintaining sufficient accuracy for the SGDLM. For instance, “[Example 17.2: Comparing CPU and GPU Workflows in Simultaneous Graphical Dynamic Linear Models](#)” on page 1077 has 300 time series ($m = 300$) and 300 observations ($T = 300$), where each variable has 10 elements in its parental set. In that case, FP32

precision requires approximately 7.6 GB of GPU memory, whereas FP64 precision requires approximately 15.04 GB.

Parental Sets of Simultaneous Graphical Dynamic Linear Models

Gruber and West (2016) posit that in practical scenarios, models are likely to have small parental sets, denoted as $\text{sp}(j)$, leading to a sparse $\mathbf{\Gamma}_t$ matrix. This sparsity in $\mathbf{\Gamma}_t$ often suggests a correspondingly sparse structure in the precision matrices $\mathbf{\Omega}_t$. The connection between zero entries in $\mathbf{\Omega}_t$ and conditional independencies within the Gaussian graphical models is discussed by Carvalho and West (2007). This foundational concept informs the nomenclature of this model class as simultaneous graphical dynamic linear models. In cases where the number of variables m is even moderate, the parental sets tend to be relatively limited in scope. If k represents the maximum size of any parental set, there would be mk nonzero entries in $\mathbf{\Gamma}_t$. Therefore, a condition where $k < (m - 1)/2$ ensures that $\mathbf{\Omega}_t$ avoids overparameterization.

The SGDLM, as implemented in PROC DYNAMICLINEAR, is specifically designed for adaptability and flexibility. Unlike fixed models, it allows the parental sets to vary over time, a feature that is crucial for evolving in response to data. This means that candidates of parental sets can differ across various time points and even for different variables. Such flexibility enables the model to effectively adapt to data, capturing the dynamics of the system under study.

Gruber and West (2017) provide an extensive discussion on the adaptability of SGDLMs. They find that smaller parental sets are generally preferable, provided that they are supported by data and predictive performance, thereby favoring increased sparsity. In larger-scale problems, a set of, for example, 20 parents for one series over a short time period might be practically indistinguishable from other candidate sets in which some parents are replaced by strongly collinear alternatives. The objective is to identify compact parental sets and adaptively revise them over time. This approach is essential for capturing and characterizing the structure and dynamics of resulting volatility matrices in terms of precision and variance. For example, in the study by Gruber and West (2017), a portfolio optimization problem that involves 401 assets is analyzed, using 20 parents for each series.

Precision of GPU Computation in PROC DYNAMICLINEAR

By default, the GPU workflow in PROC DYNAMICLINEAR uses single-precision floating-point arithmetic. This can lead to differences in the results compared to the CPU workflow, which uses double-precision arithmetic. The precision of GPU computation can affect the accuracy of the predictions and the convergence of the model, especially in cases where the data has a large range or the model is complex. It is important to understand these differences and consider them when you interpret the results from PROC DYNAMICLINEAR.

The GPU workflow supports three different precisions: TF32, FP32, and FP64.

- TF32 uses 19 bits for the significand and 8 bits for the exponent, allowing it to represent a wide range of values while maintaining a reasonable level of precision.
- FP32, or single-precision floating-point format, uses 23 bits for the significand and 8 bits for the exponent, providing higher precision than TF32 but with increased computational requirements.

- FP64, or double-precision floating-point format, uses 52 bits for the significand and 11 bits for the exponent, offering even higher precision but with significantly increased computational requirements.

The default precision is FP32, which provides a good balance between performance and accuracy for many applications. However, you can use FP64 for higher precision at the cost of slower computations. The choice of precision can depend on the specific requirements of the analysis and the characteristics of the data that you are analyzing. In general, it is recommended that you start with the default FP32 precision and evaluate the results before considering other precision options.

TF32 is not a C++ data type; it is purely a compute mode for NVIDIA Tensor Cores on Ampere+ GPUs. Within cuBLAS GEMM (general matrix multiplication) operations, the Tensor Core internally truncates the mantissa from 23 to 10 bits, performs multiplication in that reduced precision, and accumulates the result in FP32. The output is then written back as a regular single-precision float. Because GPU architectures are optimized around GEMM operations, TF32 can deliver substantial throughput gains (measured in tera floating-point operations per second, or TFLOPS) over FP32 and FP64, as illustrated by the following A100 Tensor Core peak throughput figures:

Precision	GEMM TFLOPS
FP64	19.5
FP32	19.5
TF32	156
FP16	312

Because PROC DYNAMICLINEAR is currently more GEMV (general matrix-vector multiplication)-intensive than GEMM-intensive, TF32 provides less benefit than it would for a workload that is dominated by GEMM operations. However, as the implementation of PROC DYNAMICLINEAR continues to evolve, future versions might use more GEMM operations, which could make TF32 a more attractive option for users who want to maximize performance on compatible GPUs.

Data Table Input

PROC DYNAMICLINEAR takes multiple input data tables. All input data tables are listed in [Table 17.1](#). The following sections provide an example of each table.

INDISCOUNTFACTOR= Data Table in the DISCOUNTFACTOR Statement

The input data table should be row-oriented, adhering to the following format specifications:

- The identifiers FactorID for discount factors are outlined in the following table:

Factor Name	Factor ID
beta	1
deltaphi	2
deltagamma	3

- You must include the IDs of discount factors, labeled as FactorID, in the data table.

- In the input of the TABLE= option, the number 0 can serve as a placeholder for any variables for which you choose not to assign specific values. This placeholder is ignored by the program, preserving the previously specified values from the DEFAULT= option or from the lists in the POSITIONS= and VALUES= options.

The following code demonstrates how to set the discount factors for a subset of variables—namely, y1, y2, y3, and y5—within your model for the BETA= and DELTAGAMMA= options:

```
data mylib.init_discfactors;
    input factorid y1 y2 y3 y5;
datalines;
1    0.91  0.92  0.93  0.94
3    0.85  0.86  0.87  0.89
;
```

INCOVARIANCE= Data Table in the INITIALDISTRIBUTION Statement

When you specify the INCOVARIANCE= option, use the following format requirements:

- An index column labeled RowID must be included in the INCOVARIANCE= data table. This column is essential for sorting the entries in the correct order, ensuring the proper structure of the covariance matrix.
- The table should be free of redundant columns. Because all columns in the table are processed, extraneous columns could introduce errors or distort the intended covariance specifications.
- The table must not have any missing values. In the context of a covariance matrix, missing entries are not permissible, because they could imply undefined relationships between variables. If gaps are present, they must be filled with zeros. However, care should be taken, because zero entries in a covariance matrix suggest no correlation, which might not always be the intended implication. Therefore, filling with zeros should be done with consideration of the matrix's interpretive context.

INCOVARIANCE(DIAGONAL)= Data Table in the INITIALDISTRIBUTION Statement

Here is example input for the diagonal covariance input. Including varname and parmname in the table is optional, because they primarily serve to make the table easier to understand.

```
data mylib.init_diag;
    input varname $1-4  varid parmname $24-40 parmvalue;
datalines;
y1          1    Const          1      0.008596271
y1          1    ParSet_1       2      0.041964947
y1          1    ParSet_2       3      0.367778311
y2          2    Const          1      0.004223895
y2          2    ParSet_1       2      0.002395241
y2          2    ParSet_2       3      0.022385176
y3          3    Const          1      0.003904836
y3          3    ParSet_1       2      0.047429031
y3          3    ParSet_2       3      0.489922549
y4          4    Const          1      0.05762658
y4          4    ParSet_1       2      0.075128833
```

```

y4          4    ParSet_2          3    0.22400379
y5          5    Const             1    0.104497114
y5          5    ParSet_1          2    2.129695978
y5          5    ParSet_2          3    0.047091226
;

```

INGAMMA= Data Table in the INITIALDISTRIBUTION Statement

The input data table should use a row-oriented format. The specifications for the gamma distribution factors are as follows:

- Assign FactorID to the gamma distribution parameters as outlined in the following table:

Factor Name	Factor ID
rate	2
shape	1

- You must include the FactorID column in the data table to reference these parameters.
- In the input of the GAMMA= option, the number 0 can be used as a placeholder for any variables for which values are not being specified. This placeholder is ignored by the program, and the program uses existing values that you specify in the DEFAULT= option or in the lists in the POSITIONS= and VALUES= options.

Here is an example of the input gamma. The following code specifies the shape parameter for y1 as 2, y2 as 5, y5 as 8, and the rate parameter for y3 as 4, respectively:

```

data mylib.init_gamma;
  input factorid y1 y2 y3 y5;
datalines;
1  2  5  0  8
2  0  0  4  0
;

```

INMEAN= Data Table in the INITIALDISTRIBUTION Statement

To correctly format the input data table of the INMEAN= option, use the following requirements:

- The table must include three specific columns: VarID, ParmID, and ParmValue.
- VarID values should correspond to the order of the variables as they appear in the MODEL statement.
- ParmID values represent distinct parameters that are associated with the means of state variables within the initial normal-gamma distribution framework:
 - A ParmID value of 1 designates the constant level (intercept) for the state variable's mean.
 - Subsequent integers (2, 3, and so on) in ParmID relate to the sequential elements of the variable's parental set, in the order in which they are defined.
- ParmValue values represent the actual values of the means for the state variables.

Data Table Output

PROC DYNAMICLINEAR can create multiple output data tables. All output data tables are listed in Table 17.1. The information in these tables is described in the following sections.

DISCOUNTFACTOR Statement

OUTVERIFICATION= Data Table Generated by the Statement

The OUTVERIFICATION= output data table contains the following variables:

RowID	index of the rows.
Variable_ i	value of the discount factors for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement.
VarName	name of the parameter. The values are beta, deltaphi, and deltagamma.
VarID	FactorID value of the parameter.

FILTER Statement

COVARIANCE= Data Table Generated by the PRIOR=/NAIVEPOSTERIOR=/POSTERIOR= Suboption of the OUTFILTERINFO= Option

The COVARIANCE= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
ParmValue_ k	parameter value for the covariances of the normal-gamma distributions at the time given by the sum of the ID Variable value and the LeadStep value. The parameters correspond to those in the local-level simultaneous graphical dynamic linear model in the section “ Simultaneous Graphical Dynamic Linear Models ” on page 1053. The dimension p of the variable is the sum of the number of the variables in the i th variable parental set and 1, and $k = 1, \dots, p$.
RowID	index of the rows.
VarName	name of the variable that corresponds to the variable in the MODEL statement.
VarID	order of the variables as they appear in the MODEL statement.
ParmName	name of the parameter.
ParmID	order of the parameters as they appear in the local-level SGDLM.

GAMMA= Data Table Generated by the PRIOR=/NAIVEPOSTERIOR=/POSTERIOR= Suboption of the OUTFILTERINFO= Option

The GAMMA= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.

Variable_ i	value of the shape and rate parameters for the normal-gamma distributions at a given time step for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement. These parameters correspond to the time given by the sum of the ID Variable value and the LeadStep value.
RowID	index of the rows.
VarName	name of the parameter. The possible values are shape and rate.
VarID	FactorID value of the parameter. The possible values are 1 and 2.

KLDIVERGENCE= Data Table Generated by the OUTFILTERINFO= Option

The KLDIVERGENCE= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
Effective Sample Size	value of the effective sample size at a given time step. The value is calculated on the basis of the KL divergence during the importance sampling of posterior updating. For more information about the calculation, see Gruber and West (2016).
KL Divergence	value of KL divergence at the time given by the sum of the ID Variable value and the LeadStep value.

LAMBDA= Data Table Generated by the PREDICTIONSAMPLE=/RECOUPLINGSAMPLE= Suboption of the OUTFILTERINFO= Option

The LAMBDA= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
Variable_ i	samples of the precision parameter for observation errors at a given time step for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement. The given time step is the sum of the ID Variable value and the LeadStep value.
RowID	index of the rows.

MEAN= Data Table Generated by the PRIOR=/NAIVEPOSTERIOR=/POSTERIOR= Suboption of the OUTFILTERINFO= Option

The MEAN= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
ParmValue	parameter value of the means for the normal-gamma distributions at the time given by the sum of the ID Variable value and the LeadStep value.
RowID	index of the rows.
VarName	name of the variable that corresponds to the variable in the MODEL statement.
VarID	order of the variables as they appear in the MODEL statement.

ParmName	name of the parameter.
ParmID	order of the parameters as they appear in the local-level simultaneous graphical dynamic linear model in the section “ Simultaneous Graphical Dynamic Linear Models ” on page 1053.

NU= Data Table Generated by the RECOUPLINGSAMPLE= Suboption of the OUTFILTERINFO= Option

The NU= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
Variable_ i	samples of the observation errors at a given time step for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement. The given time step is the sum of the ID Variable value and the LeadStep value.
RowID	index of the rows.

THETA= Data Table Generated by the PREDICTIONSAMPLE=/RECOUPLINGSAMPLE= Suboption of the OUTFILTERINFO= Option

The THETA= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
ParmValue_ k	samples of the state vectors at the time given by the sum of the ID Variable value and the LeadStep value. The dimension p of the variable is the sum of the number of the variables in the i th variable parental set and 1, and $k = 1, \dots, p$.
RowID	index of the rows.
VarName	name of the variable that corresponds to the variable in the MODEL statement.
VarID	order of the variables as they appear in the MODEL statement.

FILTER and FORECAST Statements

OUT= Data Table Generated by the Statements

NOTE: The variable names display a maximum of four decimal places. Any percentile values lower than 0.0001 are replaced by 0.0001 for representation.

In the output table, you might encounter instances where variance values of all variables are listed as 0. This situation typically arises when the NSIMULATIONS= option in the PROC DYNAMICLINEAR statement is set too low; the result is that each CPU thread handles only one job or none at all. Such a scenario can lead to an insufficient number of simulations for accurately estimating variance. To solve this problem, consider increasing the NSIMULATIONS= option value, ensuring that each thread processes a sufficient number of jobs to produce more reliable variance estimates.

The OUT= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
Mean_ i	sample mean at a given time step for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement. These sample means are derived from the prior normal-gamma distributions that correspond to the time given by the sum of the ID Variable value and the LeadStep value.
Variance_ i	sample variance at a given time step for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement. Like the sample mean, this variable is calculated using the prior normal-gamma distributions that correspond to the time given by the sum of the ID Variable value and the LeadStep value.
P_ ij	j th ($j = 1, \dots, J$) percentile points at a given time step for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement. The percentile points are specified by the PERCENT= <i>number-list</i> suboption of the OUT= option, and J is the number of elements in the <i>number-list</i> .

OUTCOVARIANCE= Data Table Generated by the Statements

The OUTCOVARIANCE= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
Variable_ i	the covariance value at a given time step between the i th and j th variables, $i, j = 1, \dots, m$, where m is the number of variables in the MODEL statement. The sample covariance is calculated using the samples in the OUTSIM= output data table.
RowID	index of the rows.
VarName	name of the variable that corresponds to the variable in the MODEL statement.
VarID	order of the variables as they appear in the MODEL statement.

OUTSIM= Data Table Generated by the Statements

The OUTSIM= output data table contains the following variables:

ID Variable	value of the variable in the ID statement.
LeadStep	value of h in the h -step-ahead forecast.
Variable_ i	samples of forecasting observations at a given time step for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement. These sample means are derived from the prior normal-gamma distributions that correspond to the time given by the sum of the ID Variable value and the LeadStep value.
RowID	index of the rows.

INITIALDISTRIBUTION Statement***COVARIANCE= Data Table Generated by the OUTVERIFICATION= Option***

The COVARIANCE= output data table contains the following variables:

RowID	index of the rows.
ParmValue_ k	parameter value for the covariances of the initial normal-gamma distributions. The parameters correspond to those in the local-level simultaneous graphical dynamic linear model in the section “ Simultaneous Graphical Dynamic Linear Models ” on page 1053. The dimension p of the variable is the sum of the number of the variables in the i th variable parental set and 1, and $k = 1, \dots, p$.
VarName	name of the variable that corresponds to the variable in the MODEL statement.
VarID	order of the variables as they appear in the MODEL statement.
ParmName	name of the parameter.
ParmID	order of the parameters as they appear in the local-level SGDLM.

GAMMA= Data Table Generated by the OUTVERIFICATION= Option

The GAMMA= output data table contains the following variables:

RowID	index of the rows.
Variable_ i	value of the shape and rate parameters for the initial normal-gamma distributions for the i th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement.
VarName	name of the parameter.
VarID	FactorID value of the parameter.

MEAN= Data Table Generated by the OUTVERIFICATION= Option

The MEAN= output data table contains the following variables:

RowID	index of the rows.
ParmValue	parameter value of the means for the initial normal-gamma distributions.
VarName	name of the variable that corresponds to the variable in the MODEL statement.
VarID	order of the variables as they appear in the MODEL statement.
ParmName	name of the parameter.
ParmID	order of the parameters as they appear in the local-level simultaneous graphical dynamic linear model in the section “ Simultaneous Graphical Dynamic Linear Models ” on page 1053.

PARENTALSET Statement**OUTVERIFICATION= Data Table Generated by the Statement**

The OUTVERIFICATION= output data table contains the following variables:

RowID	index of the rows.
Variable_ <i>i</i>	value of the adjacency matrices for the <i>i</i> th variable, $i = 1, \dots, m$, where m is the number of variables in the MODEL statement.
VarName	name of the variable that corresponds to the variable in the MODEL statement.
VarID	order of the variables as they appear in the MODEL statement.

Examples: DYNAMICLINEAR Procedure**Example 17.1: Tracking Volatility with Simultaneous Graphical Dynamic Linear Models**

This example shows how the DYNAMICLINEAR procedure enables you to track volatility by using the simultaneous graphical dynamic linear model (SGDLM) and how the parental set of the SGDLM can help you predict volatility more accurately.

Using the same notation as in the section “[Simultaneous Graphical Dynamic Linear Models](#)” on page 1053, consider generating m -dimensional time series $\mathbf{y}_t = (y_{1t}, y_{2t}, \dots, y_{mt})'$ that follow the SGDLM by using the following steps:

1. Define the adjacency matrix representation of the parental sets in the SGDLM and generate the matrix $\mathbf{\Gamma}_t$ that corresponds to the adjacency matrix for each time step, where $\mathbf{y}_{it} \sim N(0, \sigma_\gamma \mathbf{I})$ for $i = 1, \dots, m$ in the $\mathbf{\Gamma}_t$, and σ_γ is the factor that controls the volatility of the state vectors that correspond to the parental sets.
2. Generate the precision matrix $\mathbf{\Lambda}_t$ for each time step, where λ_{it} follows the gamma distribution with a shape parameter value of 1 and a rate parameter value of 1 for all $i = 1, \dots, m$ in $\mathbf{\Lambda}_t$.
3. Generate the vector $\boldsymbol{\mu}_t$ for each time step, where $\boldsymbol{\mu}_t \sim N(0, \sigma_\phi \mathbf{I})$, and where σ_ϕ is a factor to control the volatility of the state vectors that correspond to the external predictors.
4. Calculate the matrices $\mathbf{A}_t$ and $\boldsymbol{\Sigma}_t$ according to the equations $\mathbf{A}_t = (\mathbf{I} - \mathbf{\Gamma}_t)^{-1}$ and $\boldsymbol{\Sigma}_t^{-1} = (\mathbf{I} - \mathbf{\Gamma}_t)' \mathbf{\Lambda}_t (\mathbf{I} - \mathbf{\Gamma}_t)$.
5. Calculate the Cholesky decomposition of $\boldsymbol{\Sigma}_t$ to generate the lower-triangular matrix $\mathbf{L}_t$.
6. Simulate a vector $\tilde{\mathbf{y}}_t$, where $\tilde{\mathbf{y}}_t \sim N(0, \mathbf{I})$. Then generate $\mathbf{y}_t$, where $\mathbf{y}_t = \mathbf{A}_t \boldsymbol{\mu}_t + \mathbf{L}_t \tilde{\mathbf{y}}_t$.

This example assumes that each variable has five elements in its parental set. The following statements simulate 300 observations ($T = 300$) for 11 time series ($m = 11$) with the volatility factors $\sigma_\phi = 0.0001$ and $\sigma_\gamma = 0.01$:

```

*macro variables;
%let seed = 12345;
%let m = 11;
%let T = 300;
%let nparent = 5;
%let sigma_phi = 0.0001;
%let sigma_gamma = 0.01;

proc iml;
  call randseed(&seed.);
  *define the parental sets;
  nparent_j = j(&m,1,0);
  do h=1 to &m.;
    nparent_j[h] = &nparent.;
  end;

  parents=j(&m.,&m.,0);
  all_ind = 1:&m.;
  do h=1 to &m.;
    p_gam=nparent_j[h];
    if p_gam >0 then do;
      all_ind_j = all_ind[loc(all_ind ^= h)];
      parent_ju = sample(all_ind_j, p_gam, "NoReplace");
      parent_ju_c = parent_ju`;
      call sort(parent_ju_c);
      parent_j = parent_ju_c`;
      *print p_gam parent_j;
      parents[h,parent_j]=1;
    end;
  end;

  parentsc= {};
  all_ind = 1:&m.;
  do h=1 to &m.;
    parents_h = loc(parents[h,]=1);
    npar_h = ncol(parents_h);
    if npar_h >0 then do;
      do k=1 to npar_h;
        pjk = h||parents_h[k];
        parentsc = parentsc//pjk;
      end;
    end;
  end;

  *generate data;
  y=j(&T,&m.+1);
  do t=1 to &T;
    gamma_t = j(&m, &m, 0);
    all_ind = 1:&m.;
    do h=1 to &m.;
      p_gam=nparent_j[h];
      if p_gam >0 then do;
        parent_j = all_ind[loc(parents[h,] = 1)];

```

```

        gamj = j(p_gam,1);
        call randgen(gamj,"normal");
        gamma_t[h,parent_j`]=&sigma_gamma.*gamj`;
    end;
end;

*generate mu_t;
mu_ts = j(&m, 1);
call randgen(mu_ts, "Normal");
mu_t = &sigma_phi.*mu_ts;

*generate Lambdat;
lambda_t = j(&m, 1);
call randgen(lambda_t, "Gamma",1);
lambda_t = inv(diag(lambda_t));

*compute A_t, Sigma_t;
A_t = inv(I(&m)-gamma_t);
mean_t = A_t*mu_t;
Sigma_t = A_t*inv(lambda_t)*A_t`;
/* the Cholesky root of the full covariance matrix */
L_t = root(Sigma_t);
z_t = j(&m,1);
call randgen(z_t, "Normal"); /* z is MVN(0,I(&m)) */
y_t = mean_t + L_t`*z_t;
y[t,1:&m.]=y_t`;
y[t,&m.+1]=t;
end;

yr = y[loc(y[,&m.+1] <= &T-1),];

yvarNames = "y1":"y&m.";
varNames =yvarNames||"t";
create data_correlated from y [colname=varNames];
append from y;
close data_correlated;

create data_correlatedr from yr [colname=varNames];
append from yr;
close data_correlatedr;

create parents from parents [colname=yvarNames];
append from parents;
close parents;

varNamesc = {'ParentID', 'ChildID'};
create adj_table from parentsc [colname=varNamesc];
append from parentsc;
close adj_table;
run;

proc sort data=data_correlatedr;
    by t;
run;

```


The following statements load the generated data set `data_correlatedr` and the defined adjacency matrix data set `adj_table` into data tables in your session that is associated with the `mylib` libref. The DATA step assumes that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
data mylib.test_data;
    set data_correlatedr;
run;

data mylib.adj_table;
    set adj_table;
run;
```

Initialization settings are very important in SGDLMs. PROC DYNAMICLINEAR uses the identity matrix as the default initialization covariance. However, the identity matrix does not work well in this example. Following Gruber and West (2016), the covariance matrix $\mathbf{R}_{i0} = \text{diag}\{0.0001, 0.01, 0.01, 0.01, 0.01, 0.01\}$ is chosen as the initialization covariance matrix for all variables in the SGDLM. The following DATA step inputs the covariance matrices into the data set `initial_cov_with_adj` and then loads it into a data table in your session that is associated with the `mylib` libref:

```
data initial_cov_with_adj;
    input VarID ParmID ParmValue;
datalines;
1 1 0.0001
1 2 0.01
1 3 0.01

... more lines ...

11 3 0.01
11 4 0.01
11 5 0.01
11 6 0.01
;

data mylib.initial_cov_with_adj;
    set initial_cov_with_adj;
run;
```

The following statements use PROC DYNAMICLINEAR to define the preceding SGDLM and then generate predictions for the simulated data set:

```
proc dynamicLinear data=mylib.test_data nSimulations=10000 seed=12345;
    id t;
    model y1-y11;
    parentalSet inAdjacency=mylib.adj_table;
    filter out(percentiles=(0.5 2.5 50 97.5 99.5))=mylib.out_filter_with_adj
        outFilterInfo=(klDivergence=mylib.kldiv)
        back=0;
    forecast out(percentiles=(0.5 2.5 50 97.5 99.5))=mylib.out_forecast_with_adj;
    initialDistribution inCovariance(diagonal)=mylib.initial_cov_with_adj
        shape=(default=5) rate=(default=1.5);
    discountFactor beta=(default=0.98) deltaPhi=(default=0.98)
```

```
deltaGamma=(default=0.99);
run;
```

The summary tables of the model are shown in [Output 17.1.1](#).

Output 17.1.1 Model Summary
The DYNAMICLINEAR Procedure

Model Information			
Number of Variables Used		11	
Simulation Size		10000	
Random Number Seed		12345	
Forecast Lead		1	
Filter Back		0	
Filter Lead		1	
Number of Observations		299	
Number of Observations Used		299	
Input Tables			
Name	Contents	Number of Rows	Number of Columns
TEST_DATA	Data	299	12
ADJ_TABLE	Adjacency Matrices	55	2
INITIAL_COV_WITH_ADJ	Covariances (Diagonal)	66	3
Output CAS Tables			
CAS Library	Name	Number of Rows	Number of Columns
CASUSER(chuaxu)	OUT_FORECAST_WITH_ADJ	1	80
CASUSER(chuaxu)	OUT_FILTER_WITH_ADJ	298	80
CASUSER(chuaxu)	KLDIV	298	4

SGDLMs use the Kullback-Leibler (KL) divergence to measure the difference between the estimated and true posterior distributions. The KL divergence is calculated during the importance sampling of posterior updating. PROC DYNAMICLINEAR provides the KLDIVERGENCE= suboption of the OUTFILTERINFO= option in the [FILTER statement](#) to output the values of the KL divergence and the effective sample size at each step. The following statements create a plot of the two values at each step. The plot is shown in [Output 17.1.2](#).

```
data kldiv;
    set mylib.kldiv;
run;

proc sort data=kldiv;
    by t;
run;

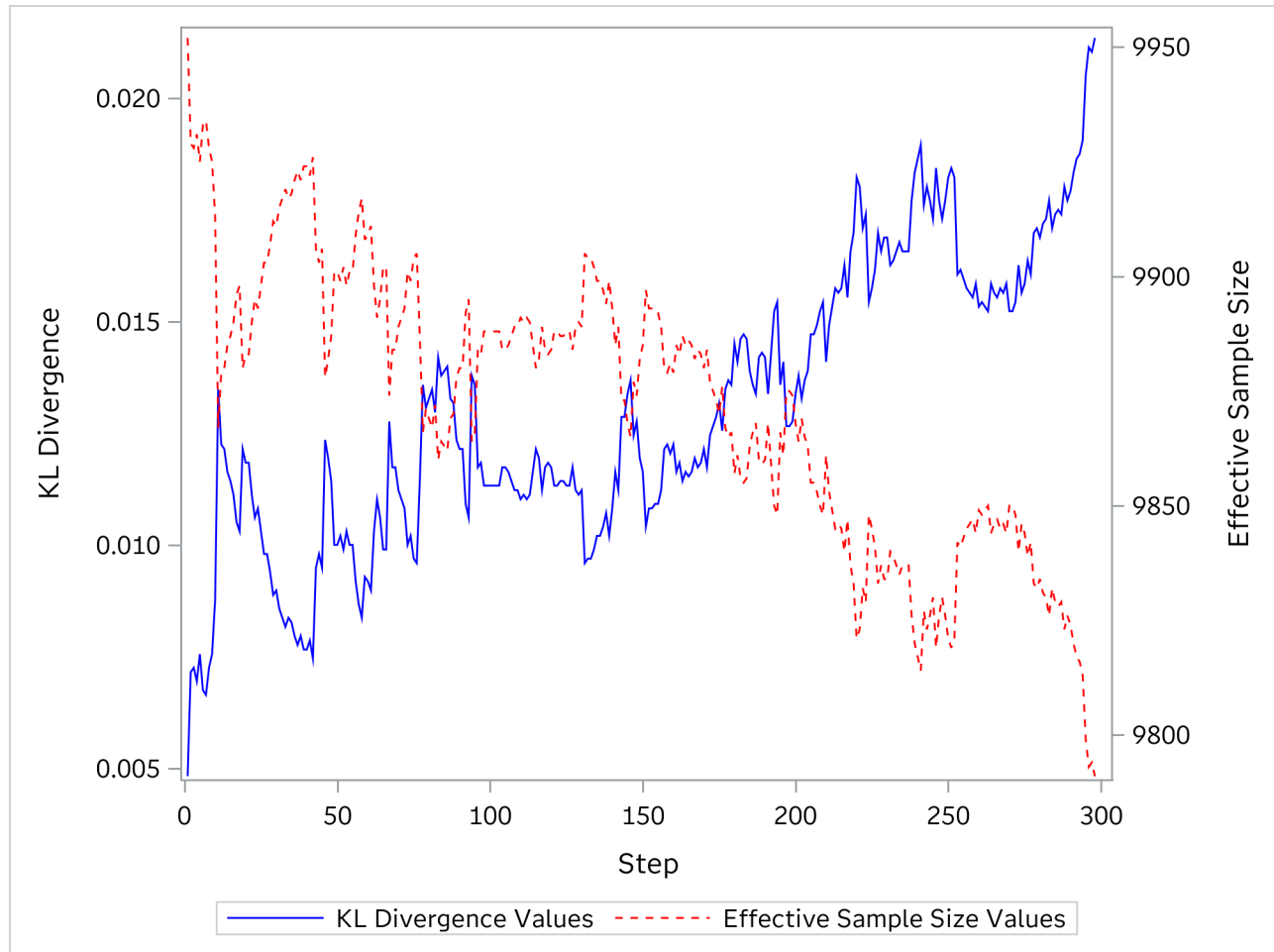
proc sgplot data=kldiv;
    series x=t y=KLDivergence / legendlabel='KL Divergence Values'
```

```

lineattrs=(color=blue pattern=solid);
series x=t y=EffectiveSampleSize / legendlabel='Effective Sample Size Values'
lineattrs=(color=red pattern=shortdash) Y2Axis;
label t="Step" KLDivergence="KL Divergence"
EffectiveSampleSize="Effective Sample Size";
run;

```

Output 17.1.2 Values of the KL Divergence and the Effective Sample Size



Output 17.1.2 shows that PROC DYNAMICLINEAR creates a good estimation of the posterior distribution in this example: first, the effective sample size converges to the number of simulations (10,000); then the KL divergence converges to 0. For more information about the quality of the posterior estimation in SGDLMs, see Gruber and West (2016).

The following statements run the same SGDLM *without the parental sets* to compare the results to the previous ones. The purpose of this comparison is to demonstrate the effect of the parental sets. The statements first define the initial covariance matrices and then run PROC DYNAMICLINEAR. All other settings are the same as those in the preceding statements.

```

data mylib.initial_cov_wo_adj;
    input VarID ParmID ParmValue;
datalines;
1 1 0.0001
2 1 0.0001
3 1 0.0001
4 1 0.0001
5 1 0.0001
6 1 0.0001
7 1 0.0001
8 1 0.0001
9 1 0.0001
10 1 0.0001
11 1 0.0001
;

proc dynamicLinear data=mylib.test_data;
    id t;
    model y1-y11;
    filter out(percentiles=(0.5 2.5 50 97.5 99.5))=mylib.out_filter_wo_adj
    back=0;
    forecast out(percentiles=(0.5 2.5 50 97.5 99.5))=mylib.out_forecast_wo_adj;
    initialDistribution inCovariance(diagonal)=mylib.initial_cov_wo_adj
        shape=(default=5) rate=(default=1.5);
    discountFactor beta=(default=0.98) deltaPhi=(default=0.98)
        deltaGamma=(default=0.99);
run;

```

The following statements create a plot of the 60-step moving average values, which are the mean predictions with and without the parental sets of the variable y_1 at each step, starting from $t = 60$. The moving average values of the variable y_1 are calculated by the EXPAND procedure and recorded in the data table `test_data_ma`. Then the data are merged and recorded in the data table `y1_mean_comb`. The plot is shown in [Output 17.1.3](#).

```

proc expand data=data_correlatedr out=test_data_ma method=none;
    id t;
    convert y1 = y1_ma / transout=(movave 60);
run;

data y1_mean_comb;
    format t y1_ma y1_adj y1_wo_adj Best12.;
    set test_data_ma(keep=t y1_ma where=(t>=60));
    set mylib.out_filter_with_adj(keep=t Mean_y1 where=(t>=59)
        rename=(Mean_y1=y1_adj));
    set mylib.out_filter_wo_adj(keep=t Mean_y1 where=(t>=59)
        rename=(Mean_y1=y1_wo_adj));
    t = t+1;
run;

proc sort data=y1_mean_comb;
    by t;
run;

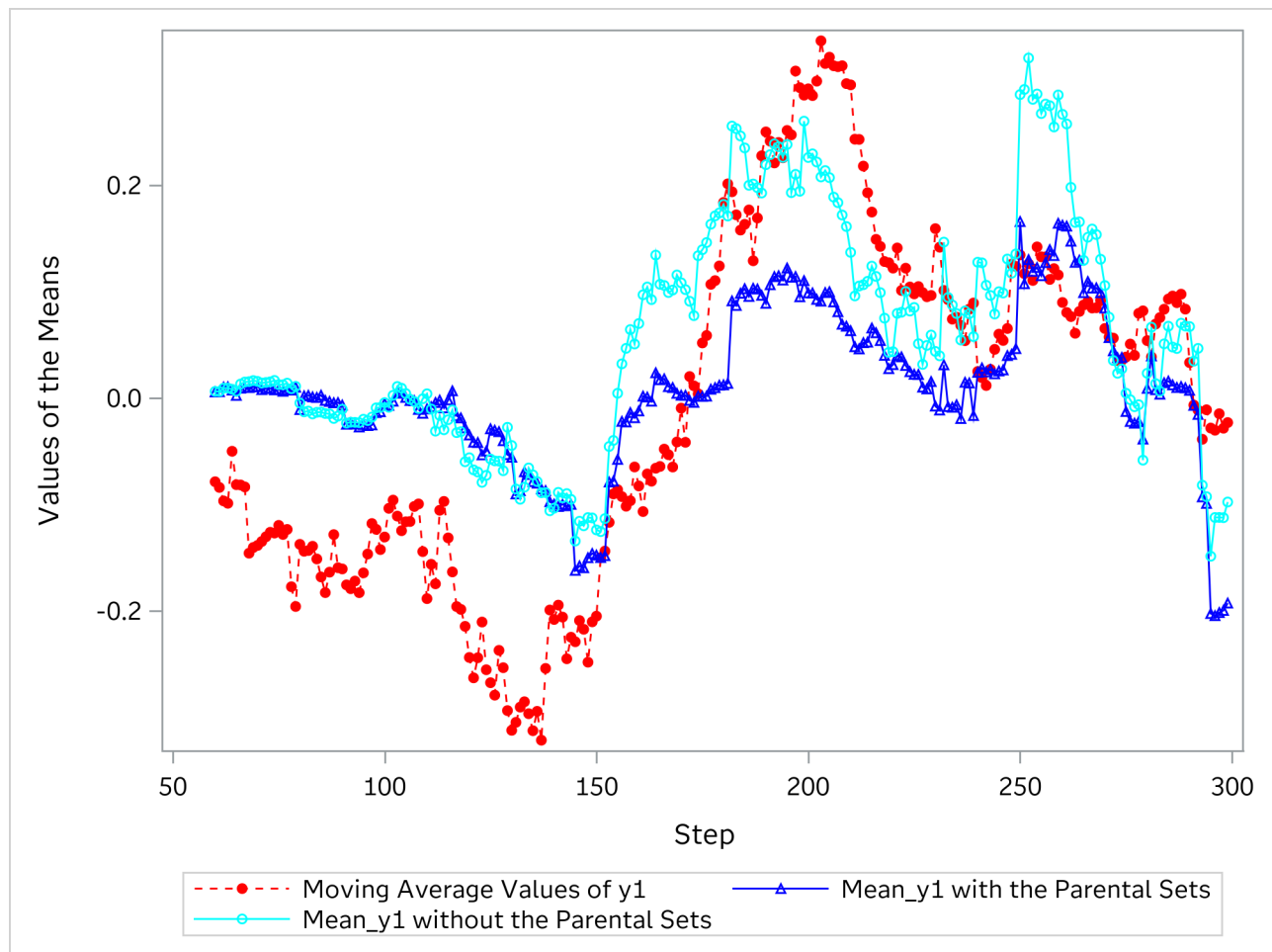
```

```

proc sgplot data=y1_mean_comb;
  series x=t y=y1_ma / legendlabel='Moving Average Values of y1'
    markers markerattrs=(color=red size=5 symbol=circlefilled)
    lineattrs=(color=red pattern=shortdash);
  series x=t y=y1_adj / legendlabel='Mean_y1 with the Parental Sets'
    markers markerattrs=(color=blue size=5 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  series x=t y=y1_wo_adj / legendlabel='Mean_y1 without the Parental Sets'
    markers markerattrs=(color=cyan size=5 symbol=circle)
    lineattrs=(color=cyan pattern=solid);
  label t="Step" y1_ma="Values of the Means";
run;

```

Output 17.1.3 Comparison of the Mean Predictions



Output 17.1.3 shows that in this example, the SGDLM with the parental sets and the SGDLM without the parental sets provide similar mean predictions.

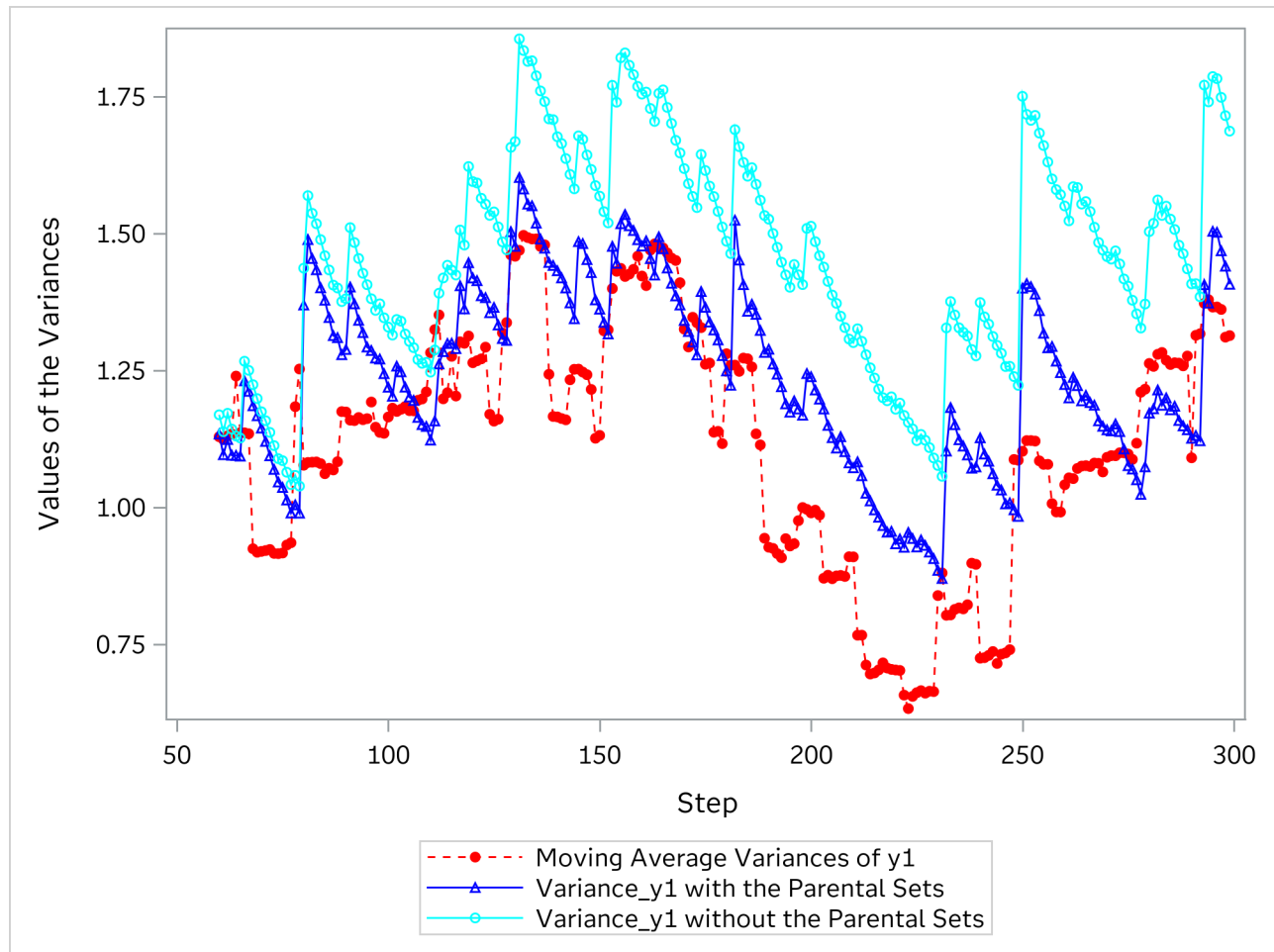
The following statements create a plot of the 60-step moving average variances, which are the variance predictions with and without the parental sets of the variable y_1 at each step, starting from $t = 60$. The moving average variances of the variable y_1 are calculated by the EXPAND procedure and recorded in the data table `test_data_ma_var`. Then the data are merged and recorded in the data table `y1_var_comb`. The plot is shown in [Output 17.1.4](#).

```
/*calculate 60-step moving average variances of the variable y1*/
proc expand data=data_correlatedr out=test_data_ma_var method=none;
  id t;
  convert y1 = y1_ma_var / transout=(MOVVAR 60);
run;

data y1_var_comb;
  format t y1_ma_var y1_adj y1_wo_adj Best12.;
  set test_data_ma_var(keep=t y1_ma_var where=(t>=60));
  set mylib.out_filter_with_adj(keep=t Variance_y1 where=(t>=59)
    rename=(Variance_y1=y1_adj));
  set mylib.out_filter_wo_adj(keep=t Variance_y1 where=(t>=59)
    rename=(Variance_y1=y1_wo_adj));
  t = t+1;
run;

proc sort data=y1_var_comb;
  by t;
run;

proc sgplot data=y1_var_comb;
  series x=t y=y1_ma_var / legendlabel='Moving Average Variances of y1'
    markers markerattrs=(color=red size=5 symbol=circlefilled)
    lineattrs=(color=red pattern=shortdash);
  series x=t y=y1_adj / legendlabel='Variance_y1 with the Parental Sets'
    markers markerattrs=(color=blue size=5 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  series x=t y=y1_wo_adj / legendlabel='Variance_y1 without the Parental Sets'
    markers markerattrs=(color=cyan size=5 symbol=circle)
    lineattrs=(color=cyan pattern=solid);
  label t="Step" y1_ma_var="Values of the Variances";
run;
```

Output 17.1.4 Comparison of the Variance Predictions

Output 17.1.4 shows that the SGDLM with the parental sets provides better volatility tracking than the SGDLM without the parental sets. The results are consistent with the conclusions of Gruber and West (2016).

Example 17.2: Comparing CPU and GPU Workflows in Simultaneous Graphical Dynamic Linear Models

This example demonstrates how to use the GPU statement in PROC DYNAMICLINEAR to accelerate the filtering and forecasting of a large-scale simultaneous graphical dynamic linear model (SGDLM). It compares the Kullback-Leibler (KL) divergence and effective sample size of the CPU and graphics processing unit (GPU) workflows, as well as the mean, median, and variance predictions for one of the variables of an SGDLM that has 300 time series ($m = 300$) and 300 observations ($T = 300$), where each variable has 10 elements in its parental set. The example is similar to “[Example 17.1: Tracking Volatility with Simultaneous Graphical Dynamic Linear Models](#)” on page 1068. **NOTE:** Because of the difference in implementation, the GPU workflow can generate results that differ from the CPU workflow results, even when using the same random seed.

Using a data generation process similar to the one in [Example 17.1](#), the following statements simulate 300 observations for 300 time series. As in [Example 17.1](#), the covariance matrix $\mathbf{R}_{i0} = \text{diag}\{0.0001, 0.01, 0.01, \dots, 0.01, 0.01, 0.01\}$ is selected as the initialization covariance matrix for all variables in the SGDLN:

```
*macro variables;
%let seed = 12345;
%let m = 300;
%let T = 300;
%let nparent = 10;
%let sigma_phi = 0.0001;
%let sigma_gamma = 0.01;
%let nSim = 10000;
%let lead = 1;

proc iml;
  call randseed(&seed.);
  *define the parental sets;
  nparent_j = j(&m, 1, 0);
  do h = 1 to &m.;
    nparent_j[h] = &nparent.;
  end;

  parents = j(&m., &m., 0);
  all_ind = 1:&m.;
  do h = 1 to &m.;
    p_gam = nparent_j[h];
    if p_gam > 0 then do;
      all_ind_j = all_ind[loc(all_ind ^= h)];
      parent_ju = sample(all_ind_j, p_gam, "NoReplace");
      parent_ju_c = parent_ju`;
      call sort(parent_ju_c);
      parent_j = parent_ju_c`;
      parents[h, parent_j] = 1;
    end;
  end;

  parentsc = {};
  all_ind = 1:&m.;
  do h = 1 to &m.;
    parents_h = loc(parents[h,] = 1);
    npar_h = ncol(parents_h);
    if npar_h > 0 then do;
      do k = 1 to npar_h;
        pjk = h || parents_h[k];
        parentsc = parentsc // pjk;
      end;
    end;
  end;

  *generate data;
  y = j(&T, &m. + 1);
  do t = 1 to &T;
```



```

gamma_t = j(&m, &m, 0);
all_ind = 1:&m.;
do h = 1 to &m.;
    p_gam = nparent_j[h];
    if p_gam > 0 then do;
        parent_j = all_ind[loc(parents[h,] = 1)];
        gamj = j(p_gam, 1);
        call randgen(gamj, "normal");
        gamma_t[h, parent_j`] = &sigma_gamma. * gamj`;
    end;
end;

*generate mut;
mu_ts = j(&m, 1);
call randgen(mu_ts, "Normal");
mu_t = &sigma_phi. * mu_ts;

*generate Lambdat;
lambda_t = j(&m, 1);
call randgen(lambda_t, "Gamma", 1);
lambda_t = inv(diag(lambda_t));

*compute A_t, Sigma_t;
A_t = inv(I(&m) - gamma_t);
mean_t = A_t * mu_t;
Sigma_t = A_t * inv(lambda_t) * A_t`;
/* the Cholesky root of the full covariance matrix */
L_t = root(Sigma_t);
z_t = j(&m, 1);
call randgen(z_t, "Normal"); /* z is MVN(0,I(&m)) */
y_t = mean_t + L_t` * z_t;
y[t, 1:&m.] = y_t`;
y[t, &m. + 1] = t;
end;

yr = y[loc(y[, &m. + 1] <= &T - 1), ];

yvarNames = "y1":"y&m.";
varNames = yvarNames || "t";
create data_correlated from y [colname=varNames];
append from y;
close data_correlated;

create data_correlatedr from yr [colname=varNames];
append from yr;
close data_correlatedr;

create parents from parents [colname=yvarNames];
append from parents;
close parents;

varNamesc = {'ParentID', 'ChildID'};
create adj_table from parentsc [colname=varNamesc];
append from parentsc;

```

```

close adj_table;

*generate initial covariance data programmatically;
nParmPerVar = 1 + &nparent.;
nRows = &m. * nParmPerVar;
init_cov = j(nRows, 3, 0);
row = 0;
do v = 1 to &m.;
  do p = 1 to nParmPerVar;
    row = row + 1;
    init_cov[row, 1] = v;
    init_cov[row, 2] = p;
    if p = 1 then
      init_cov[row, 3] = 0.0001;
    else
      init_cov[row, 3] = 0.01;
    end;
  end;
end;

covNames = {'VarID', 'ParmID', 'ParmValue'};
create initial_cov_with_adj from init_cov [colname=covNames];
append from init_cov;
close initial_cov_with_adj;
run;

proc sort data=data_correlatedr;
  by t;
run;

```

The following statements load the generated data set `data_correlatedr`, the defined adjacency matrix data set `adj_table`, and the initial covariance data set `initial_cov_with_adj` into data tables in your session that is associated with the `mylib` libref. The DATA step assumes that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```

data mylib.test_data;
  set data_correlatedr;
run;

data mylib.adj_table;
  set adj_table;
run;

data mylib.initial_cov_with_adj;
  set initial_cov_with_adj;
run;

```

The following statements run the CPU workflow of PROC DYNAMICLINEAR on the simulated data set:

```

proc dynamicLinear data=mylib.test_data nSimulations=10000 seed=12345;
  id t;
  model y1 - y300;
  parentalSet inAdjacency=mylib.adj_table;
  filter out(percentiles=(2.5, 25, 50, 75, 97.5))=mylib.out_filter_cpu
    outFilterInfo=(klDivergence=mylib.kldiv_cpu)

```

```

    back=0;
    forecast out(percentiles=(2.5, 25, 50, 75, 97.5))=mylib.out_forecast_cpu;
    initialDistribution inCovariance(diagonal)=mylib.initial_cov_with_adj
        shape=(default=5) rate=(default=1.5);
    discountFactor beta=(default=0.98) deltaPhi=(default=0.98)
        deltaGamma=(default=0.99);
run;

```

The following statements run the GPU workflow. The only difference from the CPU workflow is the addition of the **GPU statement**, which directs PROC DYNAMICLINEAR to perform the computations on GPU hardware. By default, the GPU calculations use single-precision floating-point arithmetic, which typically increases the speed significantly while maintaining sufficient accuracy for the SGDLM:

```

proc dynamicLinear data=mylib.test_data nSimulations=10000 seed=12345;
    id t;
    model y1 - y300;
    parentalSet inAdjacency=mylib.adj_table;
    filter out(percentiles=(2.5, 25, 50, 75, 97.5))=mylib.out_filter_gpu
        outFilterInfo=(klDivergence=mylib.kldiv_gpu)
        lead=1 back=0;
    forecast out(percentiles=(2.5, 25, 50, 75, 97.5))=mylib.out_forecast_gpu
        lead=1;
    initialDistribution inCovariance(diagonal)=mylib.initial_cov_with_adj
        shape=(default=5) rate=(default=1.5);
    discountFactor beta=(default=0.98) deltaPhi=(default=0.98)
        deltaGamma=(default=0.99);
    gpu;
run;

```

Comparing KL Divergence

The following statements compare the KL divergence and effective sample size of the CPU and GPU workflows. The plot is shown in [Output 17.2.1](#).

```

data kldiv_cpu;
    set mylib.kldiv_cpu;
run;
data kldiv_gpu;
    set mylib.kldiv_gpu;
run;

proc sort data=kldiv_cpu; by t; run;
proc sort data=kldiv_gpu; by t; run;

data kldiv_comb;
    merge kldiv_cpu(rename=(KLDivergence=KLDiv_CPU
                          EffectiveSampleSize=ESS_CPU))
          kldiv_gpu(rename=(KLDivergence=KLDiv_GPU
                          EffectiveSampleSize=ESS_GPU));
    by t;
run;

proc sgplot data=kldiv_comb;
    series x=t y=KLDiv_CPU / legendlabel='KL Divergence (CPU) '

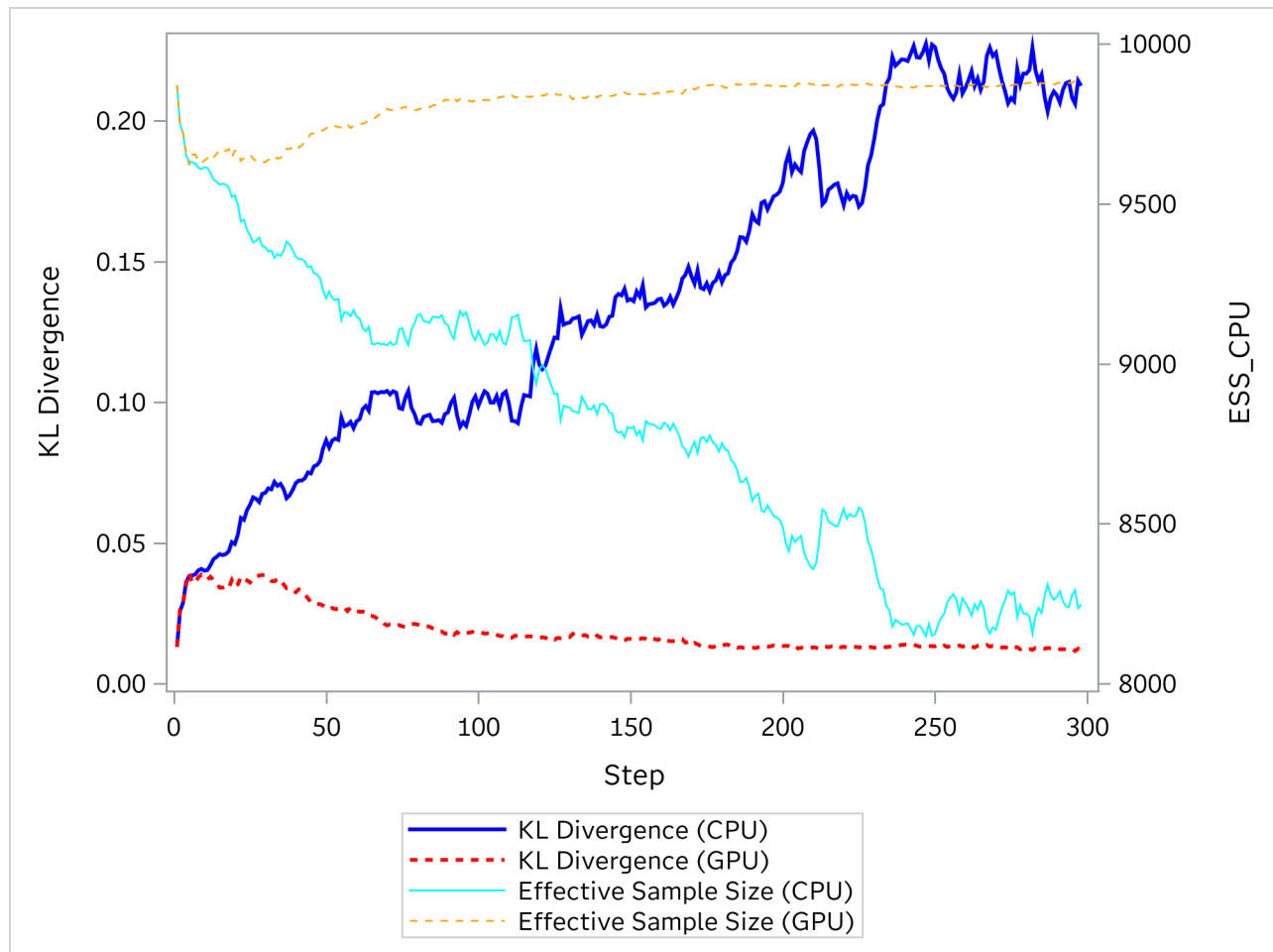
```

```

lineattrs=(color=blue pattern=solid thickness=2);
series x=t y=KLDiv_GPU / legendlabel='KL Divergence (GPU) '
lineattrs=(color=red pattern=shortdash thickness=2);
series x=t y=ESS_CPU / legendlabel='Effective Sample Size (CPU) '
lineattrs=(color=cyan pattern=solid) Y2Axis;
series x=t y=ESS_GPU / legendlabel='Effective Sample Size (GPU) '
lineattrs=(color=orange pattern=shortdash) Y2Axis;
yaxis label="KL Divergence";
xaxis label="Step";
run;

```

Output 17.2.1 KL Divergence and Effective Sample Size: CPU vs. GPU



Output 17.2.1 shows that the GPU workflow produces better results than the CPU workflow in terms of KL divergence and effective sample size for the problem that has the specified parameters. The KL divergence for the GPU workflow is consistently lower than that of the CPU workflow, indicating that the GPU-accelerated computations are providing a closer approximation to the true posterior distribution. Additionally, the effective sample size of the GPU workflow is higher than that of the CPU workflow, suggesting that the GPU computations yield more efficient sampling and better convergence properties than the CPU computations.

Comparing Mean and Median Predictions

The following statements compare the 60-step moving average of the actual values of `y1` to the mean and median predictions from the CPU and GPU workflows. The plots are shown in [Output 17.2.2](#) and [Output 17.2.3](#).

```
%let maWindow = 60;
%let varName = y1;

data out_filter_cpu;
    set mylib.out_filter_cpu;
run;

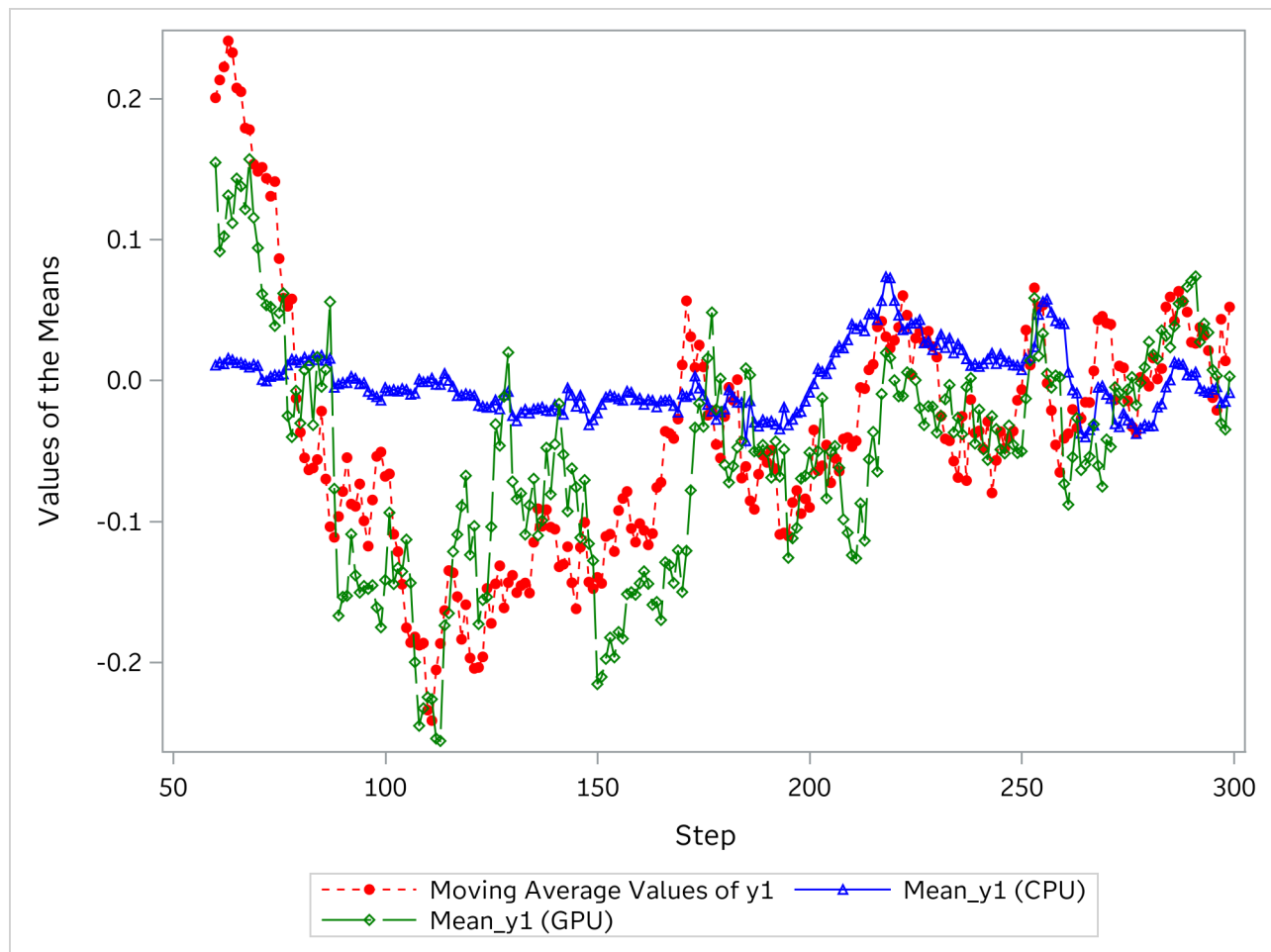
data out_filter_gpu;
    set mylib.out_filter_gpu;
run;

proc expand data=data_correlatedr out=test_data_ma method=none;
    id t;
    convert &varName. = var_ma / transout=(movave &maWindow.);
run;

data mean_comb;
    format t var_ma var_cpu var_gpu Best12.;
    set test_data_ma(keep=t var_ma where=(t >= &maWindow.));
    set out_filter_cpu(keep=t Mean_&varName. where=(t >= %eval(&maWindow. - 1))
        rename=(Mean_&varName.=var_cpu));
    set out_filter_gpu(keep=t Mean_&varName. where=(t >= %eval(&maWindow. - 1))
        rename=(Mean_&varName.=var_gpu));
    t = t + 1;
run;

proc sort data=mean_comb;
    by t;
run;

proc sgplot data=mean_comb;
    series x=t y=var_ma / legendlabel="Moving Average Values of &varName."
        markers markerattrs=(color=red size=5 symbol=circlefilled)
        lineattrs=(color=red pattern=shortdash);
    series x=t y=var_cpu / legendlabel="Mean_&varName. (CPU)"
        markers markerattrs=(color=blue size=5 symbol=triangle)
        lineattrs=(color=blue pattern=solid);
    series x=t y=var_gpu / legendlabel="Mean_&varName. (GPU)"
        markers markerattrs=(color=green size=5 symbol=diamond)
        lineattrs=(color=green pattern=longdash);
    label t="Step" var_ma="Values of the Means";
run;
```

Output 17.2.2 Comparison of the Mean Predictions: CPU vs. GPU

```
proc expand data=data_correlatedr out=test_data_med method=none;
  id t;
  convert &varName. = var_med / transout=(MOV MED &maWindow.);
run;

data median_comb;
  format t var_med var_cpu var_gpu Best12.;
  set test_data_med(keep=t var_med where=(t >= &maWindow.));
  set out_filter_cpu(keep=t P50_&varName. where=(t >= %eval(&maWindow. - 1))
    rename=(P50_&varName.=var_cpu));
  set out_filter_gpu(keep=t P50_&varName. where=(t >= %eval(&maWindow. - 1))
    rename=(P50_&varName.=var_gpu));
  t = t + 1;
run;

proc sort data=median_comb;
  by t;
run;

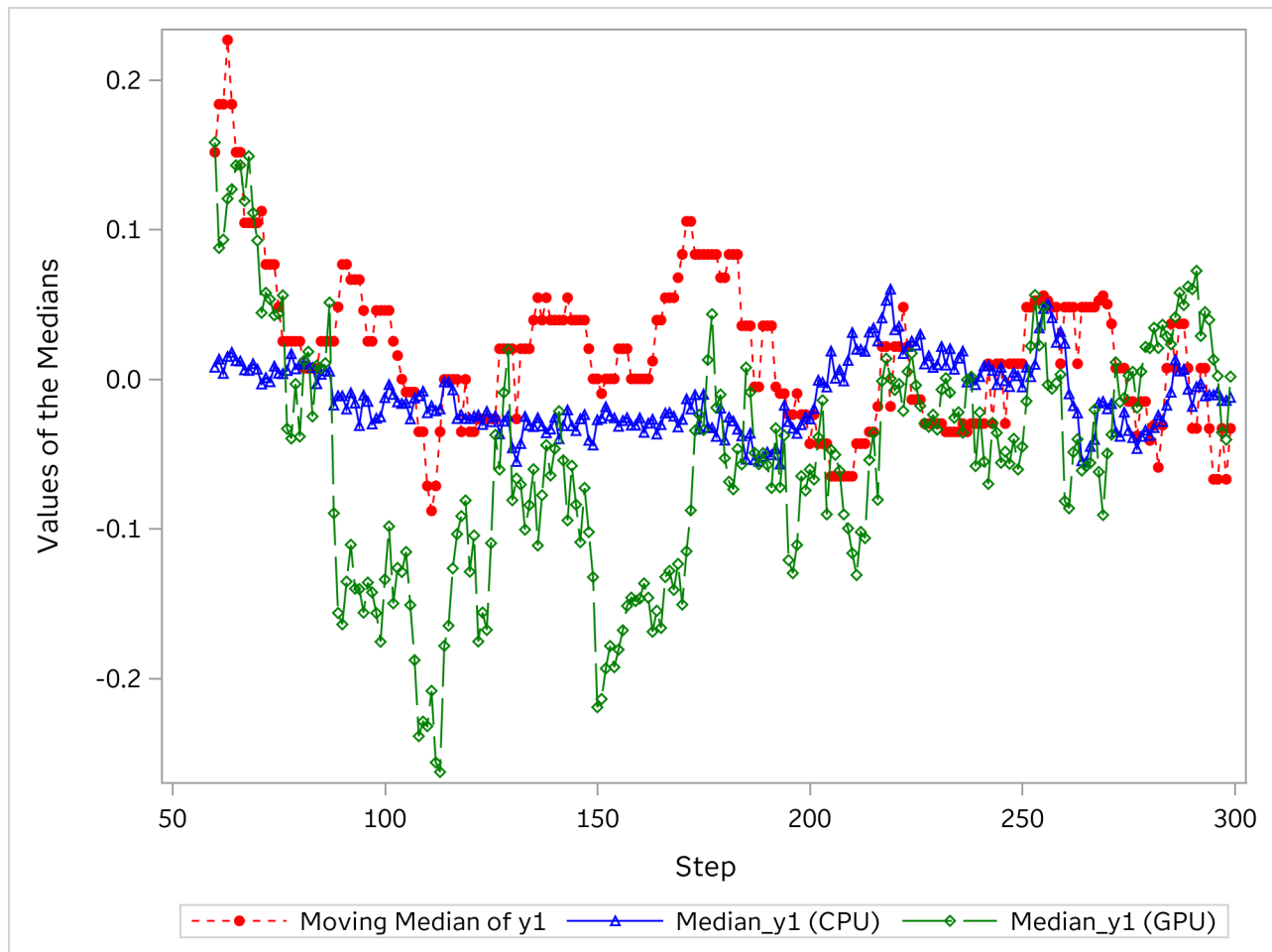
proc sgplot data=median_comb;
  series x=t y=var_med / legendlabel="Moving Median of &varName."
```

```

markers markerattrs=(color=red size=5 symbol=circlefilled)
lineattrs=(color=red pattern=shortdash);
series x=t y=var_cpu / legendlabel="Median_&varName. (CPU)"
markers markerattrs=(color=blue size=5 symbol=triangle)
lineattrs=(color=blue pattern=solid);
series x=t y=var_gpu / legendlabel="Median_&varName. (GPU)"
markers markerattrs=(color=green size=5 symbol=diamond)
lineattrs=(color=green pattern=longdash);
label t="Step" var_med="Values of the Medians";
run;

```

Output 17.2.3 Comparison of the Median Predictions (y1): CPU vs. GPU



Output 17.2.2 and Output 17.2.3 show that the CPU and GPU workflows produce mean and median predictions that share similar patterns for the variable y1. Both workflows track the moving average and moving median trends of the actual data. However, the CPU workflow has a longer burn-in period, which is reflected by the larger deviations from the moving average and moving median values in the early steps than those of the GPU workflow. The GPU workflow converges more quickly to the underlying trends, demonstrating that the GPU-accelerated computations provide more efficient sampling and better convergence properties than the CPU computations.

Comparing Variance Predictions

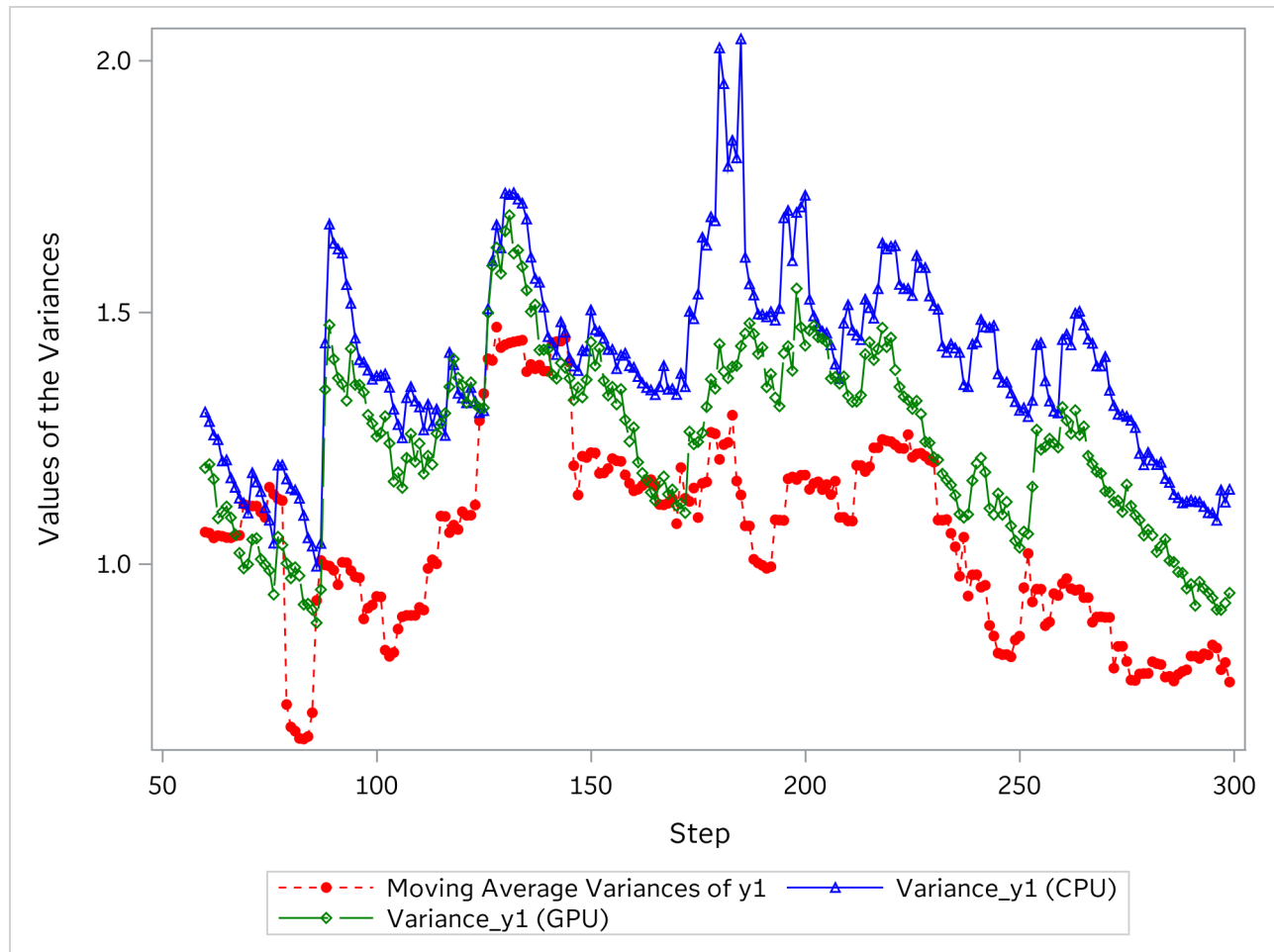
The following statements compare the 60-step moving average variance of the actual values of *y1* to the variance predictions from the CPU and GPU workflows. The plot is shown in [Output 17.2.4](#).

```
proc expand data=data_correlatedr out=test_data_ma_var method=none;
  id t;
  convert &varName. = var_ma_var / transout=(MOVVAR &maWindow.);
run;

data var_comb;
  format t var_ma_var var_cpu var_gpu Best12.;
  set test_data_ma_var(keep=t var_ma_var where=(t >= &maWindow.));
  set out_filter_cpu(keep=t Variance_&varName. where=(t >= %eval(&maWindow. - 1))
    rename=(Variance_&varName.=var_cpu));
  set out_filter_gpu(keep=t Variance_&varName. where=(t >= %eval(&maWindow. - 1))
    rename=(Variance_&varName.=var_gpu));
  t = t + 1;
run;

proc sort data=var_comb;
  by t;
run;

proc sgplot data=var_comb;
  series x=t y=var_ma_var / legendlabel="Moving Average Variances of &varName."
    markers markerattrs=(color=red size=5 symbol=circlefilled)
    lineattrs=(color=red pattern=shortdash);
  series x=t y=var_cpu / legendlabel="Variance_&varName. (CPU)"
    markers markerattrs=(color=blue size=5 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  series x=t y=var_gpu / legendlabel="Variance_&varName. (GPU)"
    markers markerattrs=(color=green size=5 symbol=diamond)
    lineattrs=(color=green pattern=longdash);
  label t="Step" var_ma_var="Values of the Variances";
run;
```

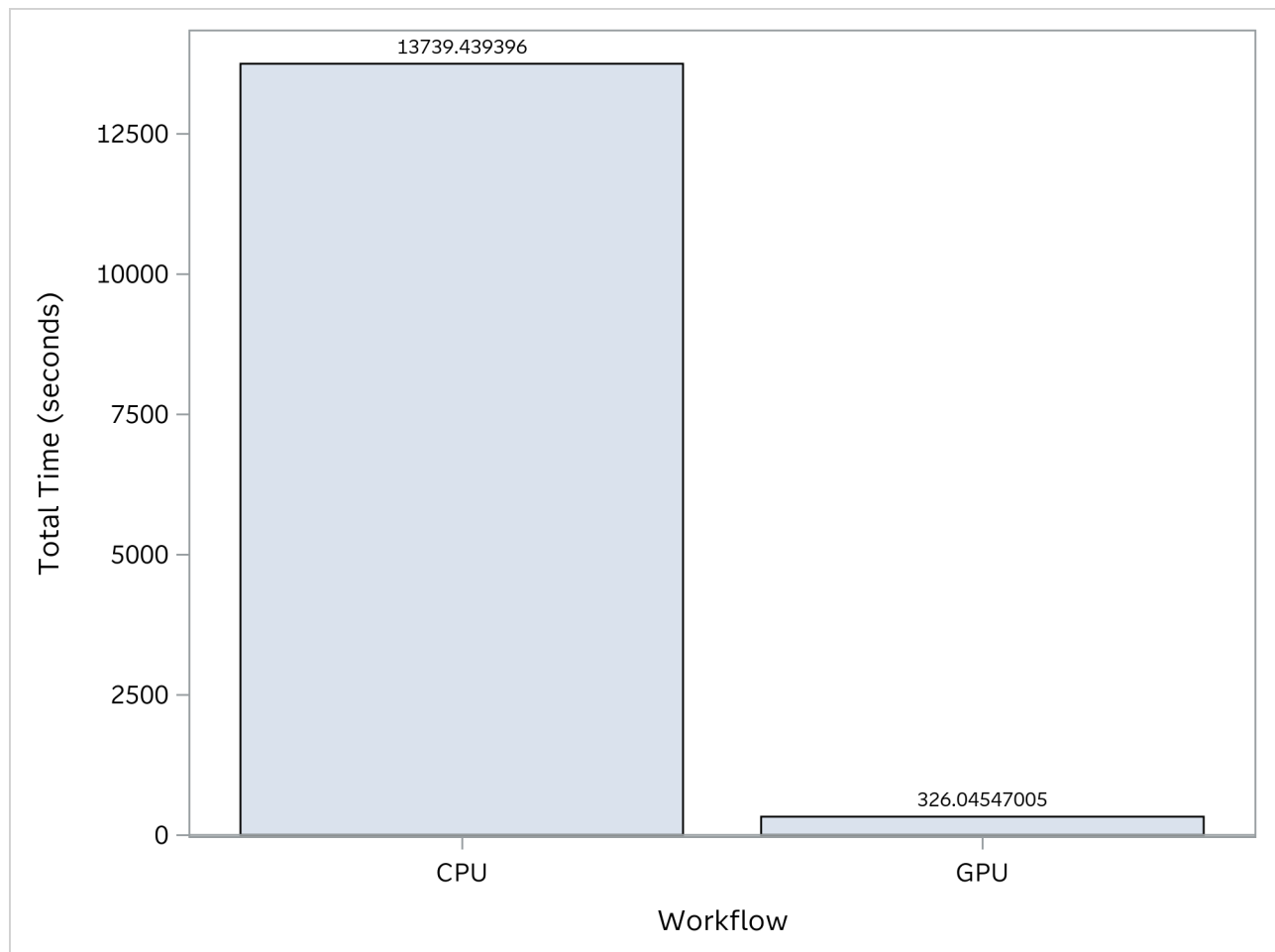

Output 17.2.4 Comparison of the Variance Predictions (y1): CPU vs. GPU

Output 17.2.4 shows that the GPU workflow provides variance predictions that closely agree with those of the CPU workflow. Both workflows track the underlying moving variance of y1. However, the results of the CPU workflow are slightly more volatile than those of the GPU workflow, especially in the early steps. This is consistent with the observations in Output 17.2.2 and Output 17.2.3 that the CPU workflow has a longer burn-in period than the GPU workflow.

Comparing Performance

In addition to providing greater accuracy, the GPU workflow also provides significantly better performance than the CPU workflow. In this example, the GPU workflow reduces the computation time from about four hours to about five minutes, demonstrating the substantial efficiency gains (more than 40 times faster) that can be achieved by GPU acceleration for large-scale SGDLMs. The example ran on a machine with an NVIDIA L40 48 GB VRAM GPU and a machine with an Intel Xeon Platinum 8358 CPU at 2.60 GHz.

Output 17.2.5 Total Run Time: CPU vs. GPU (Speedup: 42.1x)



If you take a closer look at the step-by-step run time comparison of the CPU and GPU workflows, you can see that the GPU workflow provides a significant increase in speed for each step of the computations, especially the prediction generation step, which is typically the most computationally intensive step of the SGDLM algorithm. The GPU workflow runs more than 60 times faster for the posterior updating step than the CPU workflow, substantially contributing to the overall performance improvement.

Output 17.2.6 Step-by-Step Run Time Comparison: CPU vs. GPU

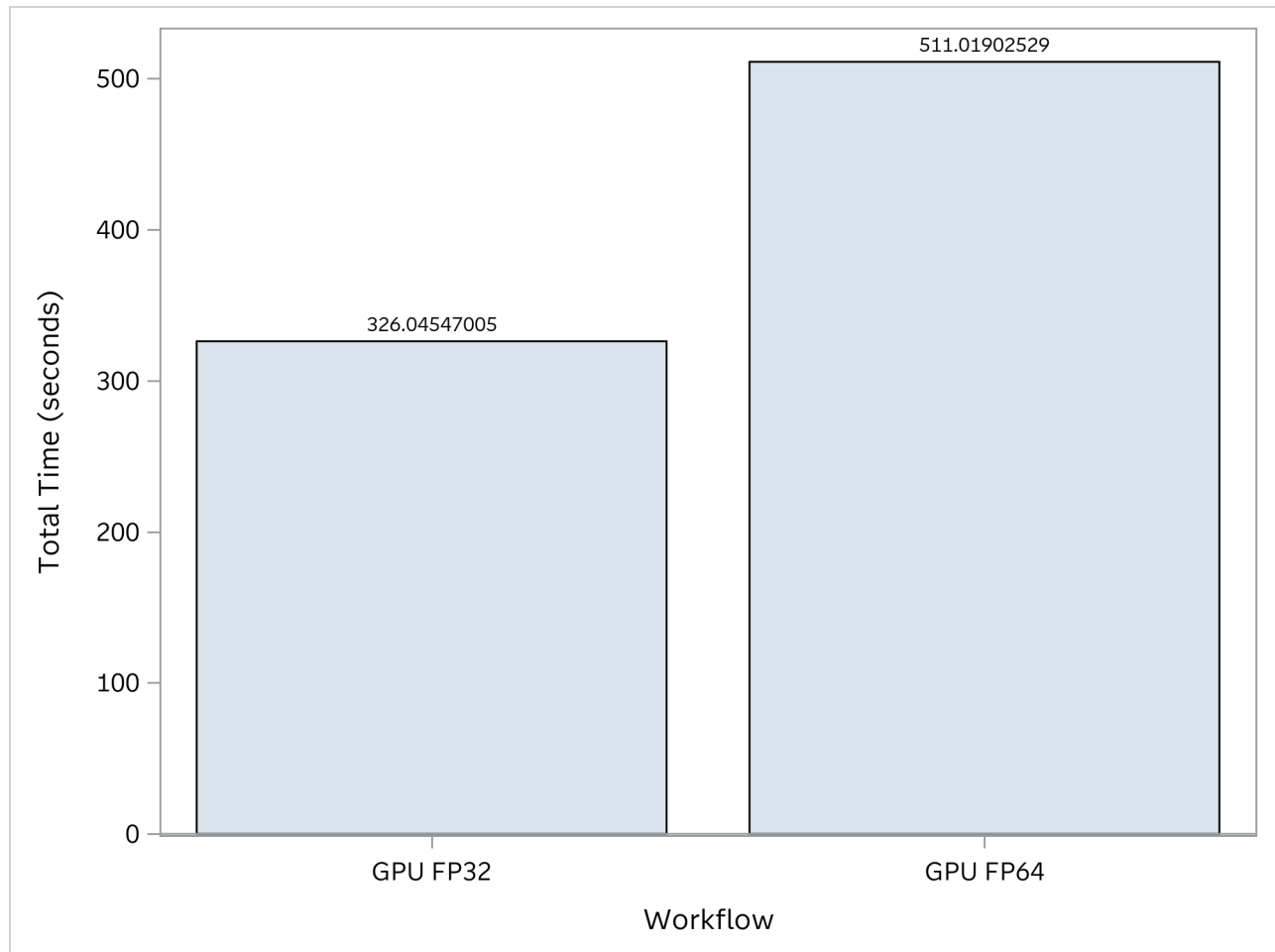
Step	CPU		GPU
	Avg (s)	Avg (s)	Speedup
computePrior	0.0044	0.00012	38.1
computeForecast	41.2756	0.64029	64.5
computePosterior	0.0017	0.00052	3.2
computeVBPosterior	4.2127	0.17667	23.8
Total	46.1055	1.09045	42.3

Comparing Precisions: FP64 vs. FP32

You can run the following statements to execute the GPU workflow with FP64 precision. The only difference from the previous GPU workflow is the addition of the PRECISION=FP64 option in the GPU statement, which directs PROC DYNAMICLINEAR to use double-precision floating-point arithmetic for the computations on GPUs.

```
proc dynamicLinear data=mylib.test_data nSimulations=10000 seed=12345;
  id t;
  model y1 - y300;
  parentalSet inAdjacency=mylib.adj_table;
  filter out(percentiles=(2.5, 25, 50, 75, 97.5))=mylib.out_filter_gpu2
    outFilterInfo=(klDivergence=mylib.kldiv_gpu2)
    lead=1 back=0;
  forecast out(percentiles=(2.5, 25, 50, 75, 97.5))=mylib.out_forecast_gpu2
    lead=1;
  initialDistribution inCovariance(diagonal)=mylib.initial_cov_with_adj
    shape=(default=5) rate=(default=1.5);
  discountFactor beta=(default=0.98) deltaPhi=(default=0.98)
    deltaGamma=(default=0.99);
  gpu precision=FP64;
run;
```

Along with the memory savings, FP32 precision can deliver significantly greater speed than FP64 precision on the NVIDIA L40S with 48 GB VRAM.

Output 17.2.7 Total Run Time: GPU FP32 vs. GPU FP64 (Speedup: 1.6x)

Meanwhile, FP32 precision provides accuracy comparable to FP64 precision for the SGDLM in this example. The following statements produce plots that compare the KL divergence and effective sample size (Output 17.2.8), means (Output 17.2.9), and variances (Output 17.2.10) of the GPU workflow predictions for each precision setting:

```
data kldiv_gpu1;
  set mylib.kldiv_gpu;
run;

proc sort data=kldiv_gpu1; by t; run;
proc sort data=kldiv_gpu2; by t; run;

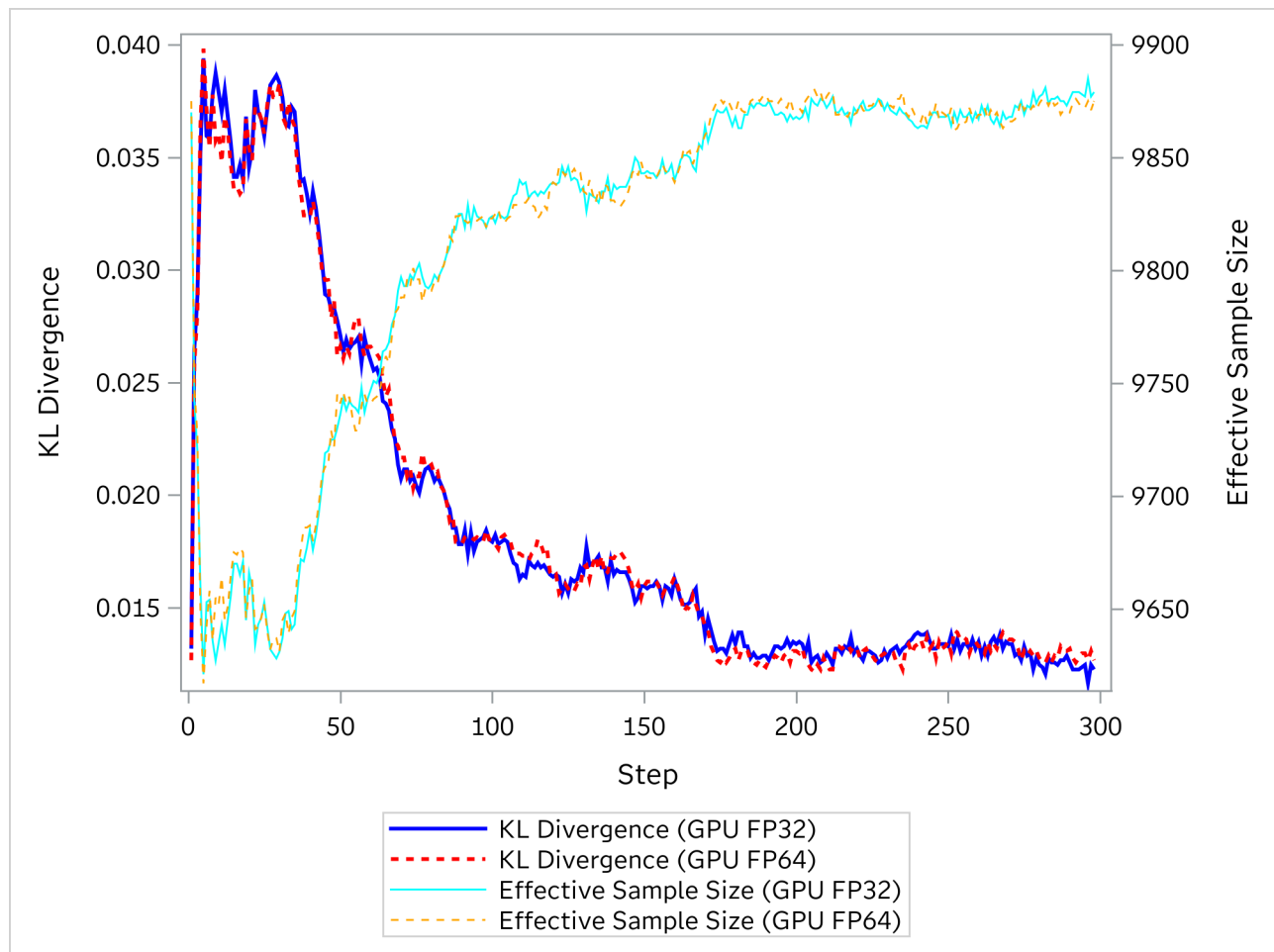
data kldiv_comb;
  merge kldiv_gpu1(rename=(KLDivergence=KLDiv_GPU1
                          EffectiveSampleSize=ESS_GPU1))
        kldiv_gpu2(rename=(KLDivergence=KLDiv_GPU2
                          EffectiveSampleSize=ESS_GPU2));
  by t;
run;
```

```

proc sgplot data=kldiv_comb;
  series x=t y=KLDiv_GPU1 / legendlabel="KL Divergence (GPU FP32)"
    lineattrs=(color=blue pattern=solid thickness=2);
  series x=t y=KLDiv_GPU2 / legendlabel="KL Divergence (GPU FP64)"
    lineattrs=(color=red pattern=shortdash thickness=2);
  series x=t y=ESS_GPU1 / legendlabel="Effective Sample Size (GPU FP32)"
    lineattrs=(color=cyan pattern=solid) Y2Axis;
  series x=t y=ESS_GPU2 / legendlabel="Effective Sample Size (GPU FP64)"
    lineattrs=(color=orange pattern=shortdash) Y2Axis;
  yaxis label="KL Divergence";
  y2axis label="Effective Sample Size";
  xaxis label="Step";
run;

```

Output 17.2.8 KL Divergence and Effective Sample Size: GPU FP32 vs. GPU FP64



```

data out_filter_gpu1;
  set mylib.out_filter_gpu;
run;

data out_filter_gpu2;
  set mylib.out_filter_gpu_fp64;
run;

```

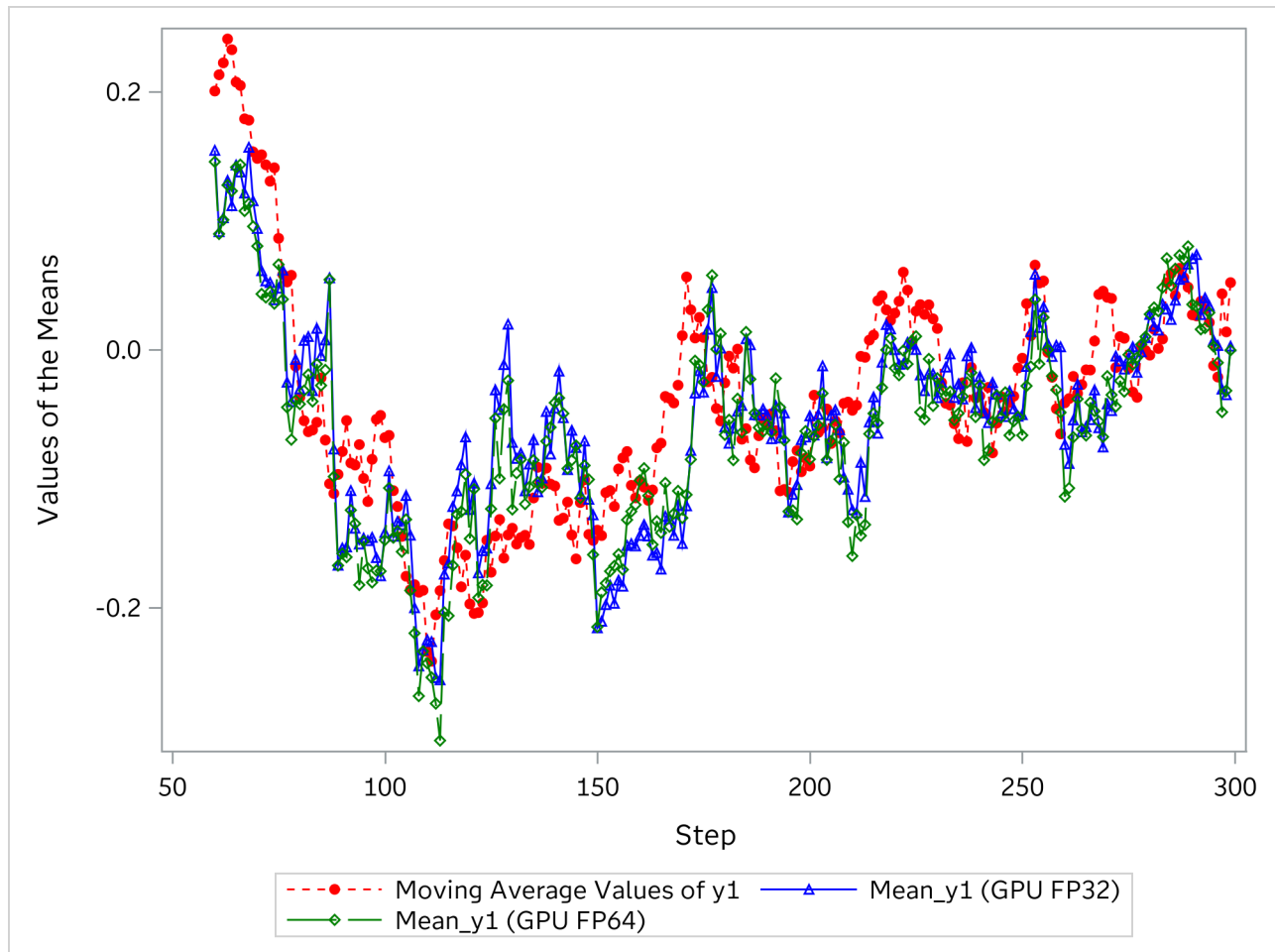
```

data mean_comb;
  format t var_gpu1 var_gpu2 Best12.;
  set test_data_ma(keep=t var_ma where=(t >= &maWindow.));
  set out_filter_gpu1(keep=t Mean_&varName. where=(t >= %eval(&maWindow. - 1))
    rename=(Mean_&varName.=var_gpu1));
  set out_filter_gpu2(keep=t Mean_&varName. where=(t >= %eval(&maWindow. - 1))
    rename=(Mean_&varName.=var_gpu2));
  t = t + 1;
run;

proc sort data=mean_comb;
  by t;
run;

proc sgplot data=mean_comb;
  series x=t y=var_ma / legendlabel="Moving Average Values of &varName."
    markers markerattrs=(color=red size=5 symbol=circlefilled)
    lineattrs=(color=red pattern=shortdash);
  series x=t y=var_gpu1 / legendlabel="Mean_&varName. (GPU FP32)"
    markers markerattrs=(color=blue size=5 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  series x=t y=var_gpu2 / legendlabel="Mean_&varName. (GPU FP64)"
    markers markerattrs=(color=green size=5 symbol=diamond)
    lineattrs=(color=green pattern=longdash);
  label t="Step" var_ma="Values of the Means";
run;

```

Output 17.2.9 Comparison of Mean Predictions: GPU FP32 vs. GPU FP64

```

data var_comb;
  format t var_ma_var var_gpu1 var_gpu2 Best12.;
  set test_data_ma_var(keep=t var_ma_var where=(t >= &maWindow.));
  set out_filter_gpu1(keep=t Variance_&varName. where=(t >= %eval(&maWindow. - 1))
    rename=(Variance_&varName.=var_gpu1));
  set out_filter_gpu2(keep=t Variance_&varName. where=(t >= %eval(&maWindow. - 1))
    rename=(Variance_&varName.=var_gpu2));
  t = t + 1;
run;

proc sort data=var_comb;
  by t;
run;

proc sgplot data=var_comb;
  series x=t y=var_ma_var / legendlabel="Moving Average Variances of &varName."
    markers markerattrs=(color=red size=5 symbol=circlefilled)
    lineattrs=(color=red pattern=shortdash);
  series x=t y=var_gpu1 / legendlabel="Variance_&varName. (GPU FP32)"
    markers markerattrs=(color=blue size=5 symbol=triangle)
    lineattrs=(color=blue pattern=solid);

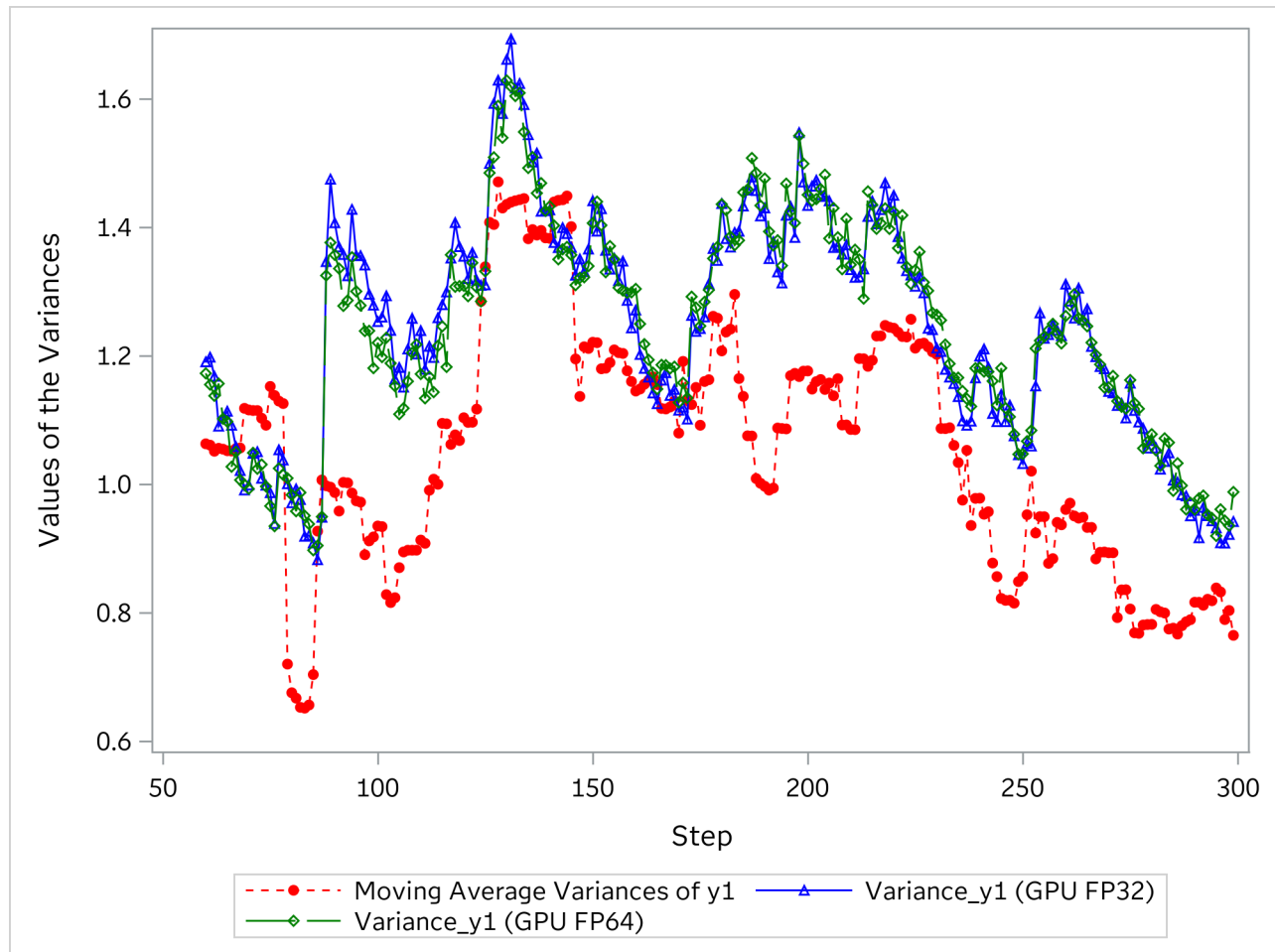
```

```

series x=t y=var_gpu2 / legendlabel="Variance_&varName. (GPU FP64) "
      markers markerattrs=(color=green size=5 symbol=diamond)
      lineattrs=(color=green pattern=longdash);
label t="Step" var_ma_var="Values of the Variances";
run;

```

Output 17.2.10 Comparison of Variance Predictions: GPU FP32 vs. GPU FP64



Summary

Overall, this example demonstrates that the GPU workflow in PROC DYNAMICLINEAR can provide accuracy comparable to that of the CPU workflow while significantly reducing computation time for large SGDLMs. The GPU-accelerated computations yield better KL divergence and effective sample size than the CPU computations, as well as more efficient sampling and better convergence properties. The performance improvements from GPU acceleration make it a powerful option for analyzing large-scale problems with SGDLMs.

References

- Carvalho, C. M., and West, M. (2007). “Dynamic Matrix-Variate Graphical Models.” *Bayesian Analysis* 2:69–97.
- Gruber, L. F., and West, M. (2016). “GPU-Accelerated Bayesian Learning and Forecasting in Simultaneous Graphical Dynamic Linear Models.” *Bayesian Analysis* 11:125–149.
- Gruber, L. F., and West, M. (2017). “Bayesian Online Variable Selection and Scalable Multivariate Volatility Forecasting in Simultaneous Graphical Dynamic Linear Models.” *Econometrics and Statistics* 3:3–22.
- Petris, G., Petrone, S., and Campagnoli, P. (2009). *Dynamic Linear Models with R*. New York: Springer-Verlag.

Chapter 18

ECM Procedure

Contents

Overview: ECM Procedure	1098
Using CAS Sessions and CAS Engine Librefs	1100
Getting Started: ECM Procedure	1101
Syntax: ECM Procedure	1105
Functional Summary	1105
PROC ECM Statement	1107
DISPLAY Statement	1112
DISPLAYOUT Statement	1113
MARGINAL Statement	1114
OUTPUT Statement	1115
OUTSUM Statement	1116
VAR Statement	1118
Details: ECM Procedure	1119
Overview of the Algorithm	1119
Percentile Computation Methods	1121
Input Specification	1122
Output Data Tables	1123
Displayed Output	1124
Examples: ECM Procedure	1127
Example 18.1: Developing an Economic Capital Model	1127
References	1143

Overview: ECM Procedure

The ECM procedure develops an economic capital model. A financial enterprise that incurs losses that are inherent to the nature of its business needs to estimate the extent of those losses across multiple business lines. Developing a model for the enterprise-wide losses enables such an enterprise not only to estimate the minimum capital it must set aside to cover the worst-case losses and meet regulatory requirements but also to assess the economic viability and improve the risk management abilities of the enterprise.

Although there is no one precise definition of economic capital, it is generally agreed that economic capital depends on the worst-case losses you expect to cover across your enterprise's business lines or risk categories. The statistical approach that is described and illustrated in this chapter can be a component of an internal model that is allowed by a regulatory framework. Contrast the approach in this chapter to the standard model, which makes some fixed assumptions and is relatively easier to compute. However, the capital requirements that stem from using the standard model often overestimate the risks and result in higher capital requirements. So, it might be prudent to invest time and resources in developing an internal model to arrive at more accurate estimates of risk, which often result in lower capital requirements and enable you to free up some capital for developing your business and increasing shareholder value. The regulatory agency should be willing to accept your internal model as long as it is statistically sound and has reasonable assumptions.

For the purposes of the ECM procedure, a model for the total loss an enterprise can incur as a result of losses across its multiple lines of business is called an economic capital model (ECM). The process of estimating an ECM is called economic capital modeling. A particularly powerful method of developing an economic capital model consists of the following steps:

1. Collect the loss event data from all business lines that incur financial losses. The loss event data consist of frequency (count) of losses that a business line incurs in a particular time period and the severity (magnitude) of each loss.
2. Estimate separate loss distribution models for the frequency and severity for each business line. Usually, the frequency model is a parametric model that consists of a discrete probability distribution, and the severity model is a parametric model that consists of a continuous probability distribution. Each model can contain regression parameters that measure the impact of external factors on the location or shape of the probability distribution.
3. Create a compound distribution model (CDM) of the aggregate loss that a business line can incur in the particular time period. This step requires combining the frequency and severity models. Because of the possibly complex nature of the frequency and severity distributions, the CDM often cannot be encoded in a concise parametric form. Hence, it is usually estimated by simulating a large empirical sample of the aggregate loss.

At this step in the process, the worst-case losses for an individual business line can be estimated by computing the value-at-risk (VaR) or tail value-at-risk (TVaR) from the large empirical sample of the CDM.

4. Estimate a loss dependency structure of the losses across all business lines. This dependency structure essentially estimates how a loss in one business line is correlated with the losses in other business lines. A typical method of estimating the dependency is to fit a copula model to the aggregate losses in each business line, where losses are matched by the time period of interest.

5. Use the estimated dependency structure to simulate a large sample of probabilities of loss in all business lines. Each observation in the sample essentially records the probability of seeing a loss in each of the business lines in the time period of interest. In other words, each observation records one possible scenario where all business lines simultaneously incur losses, the extent of which varies according to the simulated probabilities. The simulated probabilities account for the dependency among the business lines. Simulating a large number of such scenarios provides a comprehensive picture of enterprise-wide losses. However, these simulation data are on the probability scale, which creates the need for the next step of the process.
6. For each observation in the copula simulation sample, invert the probability estimate of each business line by using that line's aggregate loss sample, which the CDM simulation creates in the third step. This produces an estimate of the loss for each business line. Aggregating the losses across all business lines produces an estimate the total loss for that observation. Repeating this process for all the observations in the copula simulation table results in a large sample of the total loss, which essentially encodes the probability distribution model of the enterprise-wide losses.
7. Compute VaR and TVaR estimates for the total loss by using the large sample that the preceding step generates. These VaR and TVaR estimates help you decide the economic capital needs of your enterprise to cover worst-case losses and meet regulatory requirements.

SAS Econometrics offers various procedures to help you implement each modeling and simulation step of this process. The CNTSELECT and SEVSELECT procedures help you estimate a wide range of frequency and severity models, respectively. The CCDM procedure helps you estimate the compound distribution model (CDM) by simulating a large, distributed sample of the aggregate loss for each business line. The CCOPULA procedure helps you fit and simulate various types of copula models.

The ECM procedure helps you implement the last two critical steps of the economic capital modeling process. It uses the large, distributed CDM samples to estimate the empirical distribution function (EDF) of each business line's aggregate loss, and uses those EDF estimates to efficiently invert the probabilities in the large, distributed copula simulation sample.

The losses in each business line often depend on the economic and social environment in which the business operates. It is important to estimate the frequency and severity models that account for such external factors. The dependence of frequency and severity on external factors implies that the CDM and the economic capital model (ECM) also depend on those factors. Hence, it is not sufficient to estimate the ECM for just one set of values of the external factors, where each set of values is called an external scenario. You should estimate the ECM for multiple external scenarios, each representing a different possible state of the world. Because the CDM and ECM modeling steps need to be repeated for each external scenario, it is important that those steps run as efficiently as possible. Not only should each step use all available computational resources, but it should also be able to efficiently consume the data that the previous step generates. In particular, when the data are large and distributed across a cluster of computing and storage nodes, the modeling steps should consume the data in their distributed format instead of bringing them to one central node purely for modeling purposes.

The SAS Econometrics procedures are designed to help you achieve computational efficiency, minimal data movement, and modeling convenience. The procedures other than PROC ECM are described in their respective chapters. This chapter describes the syntax and features of PROC ECM.

PROC ECM requires SAS Cloud Analytic Services (CAS) in order to run. Because PROC ECM runs on CAS, it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: ECM Procedure

This section outlines the use of the ECM procedure to estimate the distribution of the total loss across two dependent lines of business. The example is intended as a gentle introduction to the key functionality of PROC ECM.

The losses that a particular line of business (LoB) incurs often have a compound probability distribution that cannot be expressed in a closed form, so you need to approximate the distribution by simulating a large, empirical sample from that distribution. A common way to do that is to fit probability distribution models for the frequency and severity of losses (which can usually be expressed in parametric forms) and use those models to simulate a large sample of the aggregate loss. You would typically use the CNTSELECT procedure to estimate the frequency model, the SEVSELECT procedure to estimate the severity model, and the CCDM procedure to combine those frequency and severity models to simulate an aggregate loss sample from the compound distribution. However, for the purpose of this example, let the following two DATA steps perform the aggregate loss simulations by assuming that the losses in one LoB follow a negative binomial frequency distribution and a gamma severity distribution, whereas the losses in the second LoB follow a Poisson frequency distribution and a lognormal severity distribution. This example assumes that the CAS engine libref is named mylib, as defined in the section “Using CAS Sessions and CAS Engine Librefs” on page 1100, but you can substitute any appropriately named CAS engine libref.

```
/* Simulate a sample from the compound distribution of
   negative binomial frequency and gamma severity. */
data mylib.loss1sample(keep=loss1) / sessref=mysess;
  call streaminit(12);
  theta = 3.0; p = 0.5; * negative binomial parameters;
  gTheta = 1000; gAlpha = 2.5; * gamma parameters;
  do n = 1 to 10000;
    count = rand('NEGB',p,theta);
    loss1 = 0;
    do c=1 to count;
      loss1 = loss1 + rand('Gamma', gAlpha, gTheta);
    end;
    output;
  end;
run;

/* Simulate a sample from the compound distribution of
   Poisson frequency and lognormal severity. */
data mylib.loss2sample(keep=loss2) / sessref=mysess;
  call streaminit(135);
  lambda = 2; * Poisson parameters;
  mu = log(2500); sigma = 0.85; * lognormal parameters;
  do n = 1 to 10000;
    count = rand('POISSON',lambda);
    loss2 = 0;
    do c=1 to count;
      loss2 = loss2 + exp(mu) * rand('LOGNORMAL')**sigma;
    end;
    output;
  end;
run;
```

The preceding two DATA steps produce two large sample tables, `mylib.Loss1Sample` and `mylib.Loss2Sample`, for the two LoBs. Each data table represents an empirical sample for the aggregate loss distribution of the respective LoB. The use of the `SESSREF=` option ensures that each DATA step runs on the CAS server in parallel by using multiple threads of execution on all the worker nodes in order to create large samples quickly. When each of these DATA steps is run in a CAS session that has three worker nodes, such that each worker node uses 32 threads of execution, the sample size of each LoB is 960,000, because the loop inside the DATA step runs 10,000 times independently in each thread on each worker node.

The losses in the two LoBs are usually correlated, and you can estimate their dependency structure by fitting a copula model. In the context of copula modeling, loss in each LoB constitutes a marginal random variable, referred to simply as a marginal. For this example, assume that the dependency is captured by a Gaussian copula with a correlation matrix that is recorded in the data table `mylib.CorrTab`:

```
data mylib.corrTab;
    loss1 = 1.0; loss2 = 0.35; output;
    loss1 = 0.35; loss2 = 1.0; output;
run;
```

To simulate the total loss across both LoBs that takes the dependency structure into account, you need to first simulate the probabilities of a loss value that each LoB incurs in the same time period. The following PROC CCOPULA step simulates 500,000 such loss probability combinations and stores them in the data table `mylib.LossProb`:

```
proc ccopula;
    var loss1 loss2;
    define cop normal (corr=mylib.corrTab);
    simulate cop / ndraws=500000 seed=234
        outuniform=mylib.lossprob;
run;
```

Now, you need to perform the following steps to estimate the distribution of the total loss:

1. For each observation in the copula simulation table, you need to convert the loss probability of each LoB to its loss estimate by computing the quantile function for that LoB's probability distribution. When the cumulative distribution function (CDF) or the quantile function of a marginal distribution is not available in a closed form, you need to estimate the quantile by the corresponding percentile from an empirical sample of that marginal's distribution. For this example, you need to use the empirical samples for the two LoBs that are available in the data tables `mylib.Loss1Sample` and `mylib.Loss2Sample`. After computing each LoB's loss estimate as a percentile, you need to add those loss estimates to estimate the total loss for the current observation of the copula simulation table.
2. You need to perform the preceding step for all observations of the copula simulation table. This results in a sample of the total loss, which represents an empirical estimate of the probability distribution of the total loss. You need to analyze this sample to compute the estimates of various quantities of interest, especially the risk measures such as the value-at-risk (VaR) and tail value-at-risk (TVaR).

The ECM procedure is created precisely to help you implement the preceding steps by submitting a few simple statements and executing them efficiently by using all the computational resources of a CAS server. The following PROC ECM step implements the preceding steps by using the data table `mylib.LossProb` (which contains the copula simulations) and the data tables `mylib.Loss1Sample` and `mylib.Loss2Sample` (which contain the empirical samples of the two LoBs):


```
proc ecm data=mylib.lossprob seed=57;
  marginal loss1: data=mylib.loss1sample;
  marginal loss2: data=mylib.loss2sample;
  outsum pctlpts=default tvarpts=90 95 97.5 to 99.5 by 1;
  output out=mylib.tlossSample var=tloss;
run;
```

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 54 and “Loading a SAS Data Set onto a CAS Server” on page 55 in Chapter 4, “Shared Concepts.”

The DATA= data table must contain the loss probabilities. Use of the OUTUNIFORM= option in the preceding PROC CCOPULA step ensures that the data table mylib.LossProb contains copula simulations on a uniform probability scale.

The SEED= option in the PROC ECM statement ensures that results are similar across multiple runs of the procedure for identical inputs and identical session configuration. This is required because, by default, the algorithm that PROC ECM uses samples the body region of the distribution to reduce the data movement among the worker nodes and to speed up the computations.

The two MARGINAL statements specify the two marginal variables, one for each LoB. The name of the marginal variable, which you must specify first in each MARGINAL statement followed by a colon (:), must match the name of the corresponding variable in the copula simulation table. The DATA= option in each MARGINAL statement specifies the data table that contains the empirical sample of that marginal.

The OUTSUM statement specifies the statistics that you want to estimate for the distribution of the total loss. The PCTLPTS=DEFAULT option requests that the percentiles be computed for the default percentile levels, which are {1, 5, 10, 25, 50, 75, 90, 95, 99, 99.5}. Each percentile is an estimate of the value-at-risk (VaR). The TVARPTS= option requests that the estimates of the tail value-at-risk (TVaR) be computed for 90th, 95th, 97.5th, 98.5th, and 99.5th percentile levels.

The OUTPUT statement requests that the total loss sample be written in the tloss variable in the output data table mylib.TlossSample. You can use this output table especially to compute a risk measure that is different from the two popular risk measures, VaR and TVaR, which PROC ECM computes for you.

Figure 18.1 shows some of the results that the preceding ECM step prepares. The “Summary Statistics” table displays various summary statistics. The “Percentiles” table displays the percentiles. The P th percentile is essentially an estimate of the VaR at level $\alpha = P/100$, because for a random loss variable L , the VaR at level α is defined as the smallest number l such that the probability that the loss exceeds l is no larger than $1 - \alpha$. Formally,

$$\begin{aligned}\text{VaR}_\alpha(L) &= \inf\{l \in \mathbb{R} : \Pr[L > l] \leq 1 - \alpha\} \\ &= \inf\{l \in \mathbb{R} : F_L(l) \geq \alpha\} \\ &= F_L^{-1}(\alpha)\end{aligned}$$

where F_L and F_L^{-1} denote the CDF and the quantile function of L , respectively. A percentile is essentially an empirical estimate of the quantile function, and hence an estimate of the VaR.

Figure 18.1 Summary Statistics and Percentile Estimates for the Total Loss

The ECM Procedure	
Summary Statistics	
Sample Size	500000
Minimum	0
Maximum	118808.7
Mean	14721.3
Standard Deviation	11311.6
Variance	127952686
Skewness	1.34109
Kurtosis	2.74318
Median	12368.4
Interquartile Range	14008.1
Percentiles	
Percentile	Value
1	0
5	1027.4
10	2592.0
25	6404.3
50	12368.4
75	20412.4
90	29723.3
95	36378.1
97.5	42843.2
98.5	47680.6
99	51559.0
99.5	58023.0
Percentile Method = 5	

The “Tail Value-at-Risk Estimates” table in [Figure 18.2](#) displays the VaR and TVaR estimates for the percentile levels that are specified in the TVARPTS= option. The $\Pr > \text{VaR}$ column indicates the probability that a loss exceeds the VaR value. For a level α , the TVaR_α is defined as the expected value of the loss given that the loss exceeds VaR_α . Formally, $\text{TVaR}_\alpha = E[L | L > \text{VaR}_\alpha]$. Because of this definition, TVaR is also sometimes called a conditional tail expectation (CTE). Computationally, it is the average of the loss values that exceed the P th percentile. As the “Tail Value-at-Risk Estimates” table shows, the TVaR value is always greater than the VaR value, which is expected because it is the mean of the losses that exceed the VaR value. In fact, TVaR is a better risk measure than the VaR, because TVaR accounts for the shape of the tail of the distribution. To capture the tail more accurately, you need to simulate not only large samples of the loss probabilities (copula simulation) but also large samples of each of the marginals. PROC ECM is specifically designed to handle very large copula simulation and marginal samples. It uses all the computational resources of a CAS server, and it uses a novel parallel and distributed algorithm to approximate the empirical distribution function (EDF) for computing the percentiles of the marginals. Together, these uses enable PROC ECM to perform the necessary computations significantly faster.

Figure 18.2 Tail Value-at-Risk (TVaR) Estimates for the Total Loss

Tail Value-at-Risk Estimates			
Percentile	Pr > VaR	Value at Risk (VaR)	Tail Value at Risk
90	0.1	29723.3	39197.5
95	0.05	36378.1	45702.5
97.5	0.025	42843.2	52157.3
98.5	0.015	47680.6	56881.2
99.5	0.005	58023.0	66718.0

Syntax: ECM Procedure

The following statements are available in the ECM procedure:

```

PROC ECM options ;
  MARGINAL marginal-variable: DATA=libref.data-table < marginal-options > ;
  VAR marginal-variables ;
  OUTPUT OUT=libref.data-table < output-options > ;
  OUTSUM < outsum-options > < statistic-keyword < =variable-name > ... statistic-keyword < =variable-name > > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;

```

Functional Summary

Table 18.1 summarizes the statements and options available in the ECM procedure.

Table 18.1 PROC ECM Functional Summary

Description	Statement	Option
Statements		
Specifies the information about a marginal variable	MARGINAL	
Specifies the subset of marginal variables to analyze	VAR	
Specifies where and how to write the full total loss sample	OUTPUT	
Specifies where and how to write the summary statistics of the total loss sample	OUTSUM	
Specifies the ODS tables to display (an alternative to the PRINT= option)	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	

Table 18.1 *continued*

Description	Statement	Option
Data Table Options		
Specifies the input data table that contains the copula simulation	PROC ECM	DATA=
Specifies the input data table that contains a marginal sample	MARGINAL	DATA=
Specifies the output data table for the total loss sample	OUTPUT	OUT=
Specifies the output data table for the summary statistics	OUTSUM	OUT=
Marginal Input Options		
Specifies the draw (sample) identifier for all marginals	PROC ECM	DRAWID=
Specifies the draw (sample) identifier for a specific marginal variable	MARGINAL	DRAWID=
Specifies the variable that contains the marginal variable sample	MARGINAL	SAMPLEVAR=
Empirical Distribution Function (EDF)		
Estimation Options		
Specifies the EDF accuracy for the body region of the distribution	PROC ECM	EDFACCURACY=
Computes exact counts for the final set of bins	PROC ECM	EXACTFINALCOUNT
Specifies the maximum number of iterations for the EDF estimation algorithm	PROC ECM	MAXITER=
Suppresses the shuffling of marginal data	PROC ECM	NOSHUFFLE
Specifies a sampling fraction for the body region of the distribution	PROC ECM	SAMPLEFRACTION=
Specifies a seed for the internal pseudorandom number generator	PROC ECM	SEED=
Specifies the EDF accuracy for the tail region of the distribution	PROC ECM	TAILEDFACTURACY=
Specifies the EDF value where the tail region of the distribution begins	PROC ECM	TAILSTARTEDF=
Specifies the tolerance for the EDF estimation algorithm	PROC ECM	TOLERANCE=
Output Preparation Options		
Writes the EDF and percentile estimates of marginals to the output data table	OUTPUT	MARGINALSOUT
Specifies the decimal precision to use in forming percentile variable names	OUTSUM	PCTLNDEC=

Table 18.1 *continued*

Description	Statement	Option
Specifies the prefix for the percentile variable names	OUTSUM	PCTLPRE=
Specifies the percentiles to compute and report	OUTSUM	PCTLPTS=
Specifies the method for computing the percentiles	PROC ECM	PCTLDEF=
Specifies the method for computing the percentiles of a specific marginal variable	MARGINAL	PCTLDEF=
Specifies the variable name for the total loss	OUTPUT	TOTALLOSSVAR=
Specifies the prefix for the TVaR variable names	OUTSUM	TVARPRE=
Specifies percentile values for which to compute and report the TVaR	OUTSUM	TVARPTS=
Specifies the denominator for computing second- and higher-order moments	PROC ECM	VARDEF=
Displayed Output Options		
Suppresses all displayed output	PROC ECM	NOPRINT
Specifies which displayed output to produce	PROC ECM	PRINT=

PROC ECM Statement

PROC ECM *options* ;

The PROC ECM statement invokes the procedure. You can specify the following *options*, which are listed in alphabetical order.

DATA=*libref.data-table*

COPULASIM=*libref.data-table*

names the input data table for PROC ECM to use. The default is the most recently created data table. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 1100.

data-table specifies the name of the input data table.

The DATA= data table specifies a copula sample on a uniform scale that is generated by using the OUTUNIFORM= option in the SIMULATE statement in the CCOPULA procedure.

Each observation in this data table is expected to contain the probability of the loss for each of the marginal variables. That probability is determined by accounting for the dependence structure among the marginal variables.

DRAWID=*sample-identifier*

SAMPLEID=*sample-identifier*

identifies the sample to use when the marginal data tables contain multiple samples of the marginal variable, where *sample-identifier* is an integer. A marginal data table can contain multiple samples when it is a result of the perturbation analysis that the CCDM procedure conducts. Each sample is identified by the value of the `_DRAWID_` variable. If you specify a *sample-identifier* value of *k*, then PROC ECM uses the observations in the marginal data tables whose `_DRAWID_ = k`.

The *sample-identifier* applies to all the marginal data tables that you specify in one or more MARGINAL statements. You can override the *sample-identifier* for a specific marginal table by specifying the DRAWID= option in the MARGINAL statement that you specify for the respective marginal variable.

If you omit this option or the DRAWID= option in the MARGINAL statement, then PROC ECM uses a default value of 0 for the marginal tables that contain the `_DRAWID_` variable.

EDFACCURACY=*number*

ACC=*number*

specifies the accuracy to be achieved when estimating the empirical distribution function (EDF) in the body region of the distribution, where the body region is defined as the region that contains values whose EDF estimate is less than the value that you specify in the TAILSTARTEDF= option. The *number* you specify represents the desired proportion of observations that each bin in the body region should contain. The *number* must be between 0 and 1.

PROC ECM uses *number* for estimating the EDF of each marginal variable. If you specify percentiles in the OUTSUM statement, PROC ECM also uses *number* when estimating the EDF of the total loss.

By default, EDFACCURACY=1.0E-5.

EXACTFINALCOUNT

computes the empirical distribution function (EDF) by making an additional pass over the data to count the final set of bins when the data are sampled and shuffled by the binning algorithm. This results in better EDF estimates, but at the cost of an additional data pass.

If you omit this option, then the final bin counts are approximated by inflating the counts by the sampling fraction. When the sampling fraction is less than 1, this count approximation happens only for the bins in the body region of the distribution.

If you specify a value of 1 for the **SAMPLEFRACTION=** option or if you specify the **NOSHUFFLE** option, then this option is not relevant and PROC ECM ignores it.

MAXITER=*number*

specifies the maximum number of iterations for the equal-proportions binning algorithm.

By default, MAXITER=50.

NOPRINT

suppresses all displayed output. If you specify this option, then PROC ECM ignores any value that you specify for the PRINT= option.

NOSHUFFLE

suppresses the use of shuffling in the EDF estimation process.

If you omit this option, then PROC ECM shuffles each marginal variable's data to different worker nodes so that it can perform in parallel the binning for different ranges of the marginal data. However, this additional parallelism comes at the cost of moving data across workers. When the majority of the data are in the body region of the distribution, you can reduce the amount of data that are being moved by using the **SAMPLEFRACTION=** option. However, if a marginal variable's data are very large or the **SAMPLEFRACTION=** option value is closer to 1, then the amount of data movement can make the algorithm slow.

If you specify this option, the binning algorithm does not shuffle the marginal data. Instead, it estimates the boundaries for a global set of bins in each of the regions (body and tail) of the distribution. This estimation increases the number of parameters that the algorithm needs to optimize while reducing the parallelism in the binning algorithm. Also, the algorithm needs to share the bin boundaries with all the worker nodes after every iteration; so if the values of the **EDFACCURACY=** and **TAILEDFACTURACY=** options are very small (resulting in a large number of bins), you might not save much in data movement cost. The cost of shuffling the data, which the algorithm incurs only once, might be lower than the cost of sharing the large number of bin boundaries with all worker nodes after each iteration.

If you specify this option, the **SAMPLEFRACTION=** and **SEED=** options are irrelevant and PROC ECM ignores them.

PCTLDEF=1 | 2 | 3 | 4 | 5

specifies the method of computing the percentiles. This option applies to the computation of percentiles of the total loss sample. It also applies to each marginal variable for which you do not specify a percentile method in the corresponding **MARGINAL** statement.

You can specify the following values:

- 1** uses the weighted average.
- 2** uses the value closest to the sample size times the percentile.
- 3** uses the empirical distribution function.
- 4** uses the weighted average (identical to **PCTLDEF=1**).
- 5** uses the empirical distribution function with averaging.

For more information about each method, see the section “[Percentile Computation Methods](#)” on page 1121. By default, **PCTLDEF=5**.

PRINT <(global-display-option)> =display-option

PRINT <(global-display-option)> =(display-option ... display-option)

specifies the desired displayed output. If you specify more than one *display-option*, then separate them with spaces and enclose them in parentheses. For more information about the displayed output, see the section “[Displayed Output](#)” on page 1124.

You can specify the following *global-display-option*:

ONLY

displays only the output that is requested by the *display-options*.

You can specify the following *display-options*:

ALL

displays all the output.

DATASUMMARY**DSUM**

displays a summary of the sample that is analyzed for each variable, marginal variable, or total loss variable for which PROC ECM prepares an empirical distribution function (EDF) estimate.

EDFINIT**EINIT**

displays the summary of the first initialization stage of the EDF estimation process.

EDFOPTDETAILS**EOPT**

displays the details for the data ranges for which the EDF estimation process runs an iterative binning algorithm.

EDFSUMMARY**ESUM**

displays the summary of the data ranges that the EDF estimation process analyzes.

NONE

displays no output. If you specify this option, then it overrides all other display options. The default displayed output is also suppressed.

PERCENTILES**PCTL**

displays the percentiles of the total loss sample. This includes all the predefined percentiles and percentiles that you request in the OUTSUM statement.

SUMMARYSTATISTICS**SUMSTAT**

displays the summary statistics of the total loss sample.

TVAR

displays the tail value-at-risk (TVaR) estimates of the total loss sample.

Not specifying the PRINT= option or the ONLY *global-display-option* is equivalent to specifying the PRINT=(SUMMARYSTATISTICS) option if you do not specify any percentiles in the OUTSUM statement, or is equivalent to specifying the PRINT=(PERCENTILES SUMMARYSTATISTICS) option if you specify percentiles in the OUTSUM statement.

SAMPLEFRACTION=*number***BODYSAMPLEFRACTION=***number*

specifies the fraction of observations to sample from the body region of the distribution during the EDF estimation process, where *number* must be between 0 and 1.

Sampling helps reduce the amount of data that need to be communicated across the worker nodes of the CAS server. It can reduce the accuracy of the EDF and percentile estimates, but if your marginal samples have large numbers of observations, then the loss in accuracy is not as significant as the reduction in communication cost that the sampling provides.

By default, SAMPLEFRACTION=0.5.

SEED=*number*

specifies an integer to use as the seed in generating the pseudorandom numbers that are used for sampling the body region of the distribution.

If you omit this option or if you specify a *number* that is negative or 0, then PROC ECM uses as the seed a number that depends on the time of day from the computer's clock.

TAILEDFAccURACY=*number***TAILACC=***number*

specifies the accuracy to be achieved when estimating the empirical distribution function (EDF) in the tail region of the distribution, where *number* (which must be between 0 and 1) is the desired fraction of observations that each bin in the tail region should contain. The tail region is defined as the region that contains values whose EDF estimate is larger than or equal to the value that you specify in the TAILSTARTEDF= option.

PROC ECM uses *number* when it estimates the EDF of each marginal variable. If you specify percentiles in the OUTSUM statement, PROC ECM also uses *number* when it estimates the EDF of the total loss.

By default, TAILEDFAccURACY=1.0E-6.

TAILSTARTEDF=*number***TAILSTART=***number*

specifies the empirical distribution function (EDF) value that marks the beginning of the tail region of the distribution, where *number* must be between 0 and 1. The first stage of the EDF estimation algorithm finds an approximate value y_t whose EDF is close to *number*. The part of the sample whose values are less than y_t defines the body region of the distribution, and the remaining values define the tail region of the distribution.

By default, TAILSTARTEDF=0.8.

TOLERANCE= ϵ **TOL=** ϵ

specifies the tolerance value that determines when the equal-proportion binning algorithm has converged, where ϵ must be a number between 0 and 1. The algorithm converges when the difference in the fractions of observations in the largest and smallest bins is less than or equal to $\delta\epsilon$, where δ is the accuracy of the body or tail region that is being analyzed.

By default, TOLERANCE=0.5.

VARDEF=divisor

specifies the divisor to use in calculating the variance, standard deviation, kurtosis, and skewness of the total loss sample. You can specify one of the following values for the *divisor*, where N is the sample size:

DF

sets the divisor for variance to $N - 1$. This also changes the definitions of skewness and kurtosis.

N

sets the divisor to N .

By default, VARDEF=DF.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[Displayed Output](#)” on page 1124. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the output table name.

key specifies *key* as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[Displayed Output](#)” on page 1124. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

MARGINAL Statement

MARGINAL *marginal-variable*: **DATA=***libref.data-table* < *marginal-options* > ;

MGL *marginal-variable*: **DATA=***libref.data-table* < *marginal-options* > ;

The MARGINAL statement enables you to specify information about one *marginal-variable*. Specify one MARGINAL statement for each marginal variable that you want to analyze.

You must specify the name of the *marginal-variable* followed by a colon (':') as the first item in the MARGINAL statement. This *marginal-variable* must appear in the copula simulation data table that you specify in the **DATA=** option in the PROC ECM statement.

Next, you must specify the data table by using the following option:

DATA=*libref.data-table*

specifies the data table that contains an empirical sample of the *marginal-variable*. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1100. The *libref* must be identical to the *libref* that you specify in the **DATA=** option in the PROC ECM statement.

Typically, you specify a data table that was generated by the CCDM procedure in order to simulate a compound distribution sample from the underlying frequency and severity models that the CNTSELECT and SEVSELECT procedures create.

You can specify the following *marginal-options* to specify the variable and observations that contain the marginal variable sample:

SAMPLEVAR=*sample-variable*

SVAR=*sample-variable*

specifies the variable that contains the marginal sample in the marginal data table.

If the name of the *sample-variable* is the same as the *marginal-variable*, then you do not need to specify this option. If PROC ECM cannot find a variable named *marginal-variable* in the marginal table and if you do not specify this option, then the sample variable name is assumed to be `_AGGSEV_`, which must be present in the marginal data table.

DRAWID=*sample-identifier*

SAMPLEID=*sample-identifier*

identifies the sample to use when the marginal data table contains multiple samples of the marginal variable, where *sample-identifier* is an integer.

The data table can contain multiple samples when it is a result of the perturbation analysis that the CCDM procedure conducts. Each sample is identified by the value of the `_DRAWID_` variable. If you specify a *sample-identifier* value of *k*, then PROC ECM uses the observations from the marginal data table whose `_DRAWID_=k`.

If you do not specify either this option or the DRAWID= option in the PROC ECM statement, then PROC ECM uses a default value of 0 when the marginal table contains the `_DRAWID_` variable.

The value that you specify for this option overrides the value that you specify for the DRAWID= option in the PROC ECM statement.

PCTLDEF=1 | 2 | 3 | 4 | 5

specifies the method of computing the percentiles. The value that you specify for this option overrides the value that you specify for the PCTLDEF= option in the PROC ECM statement.

You can specify the following values:

- 1** uses the weighted average.
- 2** uses the value closest to the sample size times the percentile.
- 3** uses the empirical distribution function.
- 4** uses the weighted average (identical to PCTLDEF=1).
- 5** uses the empirical distribution function with averaging.

For more information about each method, see the section “[Percentile Computation Methods](#)” on page 1121.

By default, PCTLDEF=5.

OUTPUT Statement

OUTPUT **OUT=***libref.data-table* < *output-options* > ;

The OUTPUT statement specifies the data table in which PROC ECM outputs the estimated total loss sample.

If you specify more than one OUTPUT statement, only the first one is used.

You must specify the output data table by using the following option:

OUT=*libref.data-table*

OUTSAMPLE=*libref.data-table*

specifies the name of the output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1100. The *libref* must be identical to the *libref* that you specify in the **DATA=** option.

You can specify the following *output-options* to control the contents of the OUT= data table:

MARGINALSOUT

WRITEMARGINALS

writes the marginal probability and estimated percentile information to the OUT= data table. The marginal probability is copied from the copula simulation data table.

TOTALLOSSVAR=*variable-name*

VAR=*variable-name*

specifies the name of the variable to contain the total loss value in the OUT= data table. If you omit this option, then the variable is named `_TOTALLOSS_` by default.

For more information about the variables in the OUTSAMPLE= data table, see the section “[OUTSAMPLE= Data Table](#)” on page 1123.

OUTSUM Statement

OUTSUM < *outsum-options* > < *statistic-keyword* < =*variable-name* > ... *statistic-keyword* < =*variable-name* > > ;

The OUTSUM statement specifies the data table in which PROC ECM writes the summary statistics of the total loss sample.

If you specify more than one OUTSUM statement, only the first one is used.

You can specify the following *outsum-options*:

OUT=*libref.data-table*

OUTSUM=*libref.data-table*

specifies the output data table that contains the summary statistics of the total loss sample. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1100. The *libref* must be identical to the *libref* that you specify in the DATA= option.

You can control the statistics that appear in this data table by specifying the PCTLPTS= option, the TVARPTS= option, and different *statistic-keyword* values.

PCTLNDEC=*integer-value*

specifies the maximum number of decimal places to use when creating the names of the variables for the percentile values in the PCTLPTS= or TVARPTS= options. For example, for a percentile value of 99.9995, if you specify PCTLNDEC=4, then the suffix of the variable is 99_9995. Specifying a small number might result in duplicate variable names. If that happens, PROC ECM writes an error message and exits.

By default, PROC ECM computes the required number of decimal places by using the precision of the percentiles that you specify in the PCTLPTS= option or the TVARPTS= option.

PCTLPRE=*prefix*

specifies the prefix to use for creating the names of the variables that contain the percentile estimates. The percentile estimate for a value *p* is written to a variable that has the name of the form *prefixt*, where *t* is the text representation of *p* that is formed by replacing the decimal point with an underscore (_). For example, if PCTLPRE=P, then the estimate of the 97.5th percentile is written to a variable whose name is P97_5. By default, PCTLPRE=P.

PCTLPTS=*percentile-option*

specifies the percentiles of the total loss that you want PROC ECM to compute. The *percentile-option* can have one of the following two forms:

DEFAULT**DEF**

computes only the default list of percentile values, which is {1, 5, 10, 25, 50, 75, 90, 95, 99, 99.5}.

percentile-list

adds the percentiles that you specify in the *percentile-list* to the default list of percentiles, where *percentile-list* is a comma-separated list of percentile values, each of which must belong to the (0,100) open interval. You can also use a list notation of the following form: *< number1 > to < number2 > by < increment >*. For example, the following two options are equivalent:

```
pctlpts=10, 20, 99.6, 99.7, 99.8, 99.9
pctlpts=10, 20, 99.6 to 99.9 by 0.1
```

TVARPRE=*prefix*

specifies the prefix to use for creating the names of the variables that contain the tail value-at-risk (TVaR) estimates. The TVaR estimate for a percentile value *p* is written to a variable that has the name of the form *prefixt*, where *t* is the text representation of *p* that is formed by replacing the decimal point with an underscore (_). For example, if TVARPRE=TVaR, then the tail value-at-risk estimate for the 97.5th percentile is written to a variable whose name is TVaR97_5. By default, TVARPRE=TVaR.

TVARPTS=*tvar-percentile-list*

specifies the percentile values for which you want PROC ECM to compute the tail value-at-risk (TVaR) of the total loss, where *tvar-percentile-list* is a comma-separated list of percentile values, each of which must belong to the (0,100) open interval. You can also use a list notation of the form “*< number1 > to < number2 > by < increment >*”. For example, the following two options are equivalent:

```
tvarpts=90, 95, 96.25, 97.5, 98.75, 99.5
tvarpts=90, 95 to 99 by 1.25, 99.5
```

For each value in the *tvar-percentile-list*, PROC ECM also reports the percentile estimates, which are the estimates of the value-at-risk (VaR) for the corresponding TVaR estimate.

If you specify a data table in the OUTSUM= option, PROC ECM adds the TVaR estimates to that table. You can control the name of each TVaR estimate's variable by specifying the TVARPRE= option.

You can also specify one or more predefined statistics of the total loss sample to be written to the OUTSUM= table in the following form:

*statistic-keyword***< =variable-name >**

writes the statistic that corresponds to the *statistic-keyword* to a variable named *variable-name*. For the predefined statistics that correspond to percentiles, the *variable-name* takes precedence over the default name that PROC ECM forms by using the PCTLPRE= option. All variable names in the OUTSUM= data table have a limit of 32 characters.

If you do not specify the *variable-name*, then PROC ECM writes the statistic to a variable named *statistic-keyword*.

You can specify the following *statistic-keywords*, which result in the specified statistic being written to the OUTSUM= table:

KURTOSIS KURT	writes the kurtosis of the total loss sample.
MEAN	writes the mean of the total loss sample.
P01	writes the first percentile of the total loss sample.
P05	writes the fifth percentile of the total loss sample.
P10	writes the 10th percentile of the total loss sample.
P25 Q1	writes the first quartile (the 25th percentile) of the total loss sample.
P50 MEDIAN Q2	writes the median (the 50th percentile) of the total loss sample.
P75 Q3	writes the third quartile (the 75th percentile) of the total loss sample.
P90	writes the 90th percentile of the total loss sample.
P95	writes the 95th percentile of the total loss sample.
P99	writes the 99th percentile of the total loss sample.
P99_5 P995	writes the 99.5th percentile of the total loss sample.
QRANGE	writes the interquartile range (Q3–Q1) of the total loss sample.
SKEWNESS SKEW	writes the skewness of the total loss sample.
STDDEV STD	writes the standard deviation of the total loss sample.
VARIANCE	writes the variance of the total loss sample.

VAR Statement

VAR *marginal-variables* ;

The VAR statement specifies a subset of marginal variables to analyze.

You must specify a MARGINAL statement for each of the *marginal-variables* that you specify, and each variable in the list of *marginal-variables* must appear in the copula simulation data table.

Details: ECM Procedure

Overview of the Algorithm

The ECM procedure accepts as input two types of data tables. The first is a copula simulation table that contains the probability of a loss that is incurred in the time period of interest for each line of business (LoB) that you include in the copula model. These probabilities account for the dependency among the losses in different LoBs. You typically generate this table by using the CCOPULA procedure. The second type of table is an aggregate loss sample table for each LoB, which essentially encodes an empirical estimate of the probability distribution of the losses for that LoB. You typically generate this table by using the CCDM procedure. In the context of copula modeling, loss in each LoB constitutes a marginal random variable, which is simply called a marginal for brevity. You can specify multiple marginals in a PROC ECM step as long as you have used each specified marginal in the copula model and you provide a loss sample for each marginal.

The main mandate of PROC ECM is to convert the copula simulation table to a sample of the total loss that you expect to incur across all marginals. Let the copula simulation table contain N observations, and let each i th observation contain the probability p_{ik} for the k th marginal out of K marginals. Let $F_k(l)$ and $F_k^{-1}(p)$ denote the cumulative distribution function (CDF) and the quantile function, respectively, for the k th marginal. With this notation, conceptually, PROC ECM needs to execute the following steps:

1. For each i th observation in the copula simulation table:
 - a) Compute L_{ik} (the loss estimate for each k th marginal) as $L_{ik} = F_k^{-1}(p_{ik})$.
 - b) Compute L_i (the estimate of the total loss) by adding the loss estimates of all marginals as $L_i = \sum_{k=1}^K L_{ik}$.
2. Analyze the set of all $\{L_i\}$ values (which together constitute the final sample of the total loss) to compute summary statistics, value-at-risk (VaR) estimates, and tail value-at-risk (TVaR) estimates of the total loss distribution.

The key part in these steps is the computation of the quantile function for each marginal in step 1a. Because the quantile function is not available in a closed-form equation, PROC ECM has to compute it by using the empirical sample of the loss distribution of each marginal. An empirical estimate of a quantile is the percentile. So for each probability value p_{ik} , PROC ECM needs to compute the $100p_{ik}$ th percentile from the sample of the k th marginal. This seems simple at first, but it gets complicated when the marginal's sample is large and distributed across multiple nodes of the CAS server. Moreover, the copula simulation table is also large and distributed across multiple nodes, which means that a large number of percentile computations need to be done on multiple nodes for each marginal. One straightforward solution is to assemble the entire sample of each marginal in the memory of one of the worker nodes, sort that sample, and compute the percentile. However, if a percentile of a marginal is computed and available only on one worker node, then the copula simulation table also must reside in its entirety on that same worker node. So essentially, tables of all the marginals and the copula simulation table need to be moved in their entirety to a single worker node. Not only is this time-consuming because of all the data movement among nodes, but it is also a very inefficient use of the available computational power and memory resources. Moreover, depending on the sizes of all tables, this simplistic solution might not even be feasible if the main memory of any one worker node is not large enough to hold all the tables.

To overcome these challenges, PROC ECM implements a novel algorithm to compute each marginal's percentile. The key component of the algorithm is the parallel and distributed estimation of the empirical distribution function (EDF). PROC ECM computes the EDF estimates in the form of a set of bin boundaries, such that the fraction of observations in each bin is approximately equal. By keeping approximately the same fraction in each bin, the algorithm essentially offers you control over the precision of the EDF estimates. The smaller the fraction of observations in each bin, the greater the precision and the higher the accuracy of the EDF estimates. The bin boundaries together represent a compressed version of the original empirical sample. As long as the desired accuracy is not too high to cause each bin to have very few observations, the algorithm can achieve a good degree of data compression and thereby save on the memory that it needs in order to store the bin boundaries. PROC ECM runs an iterative algorithm to adjust the bin boundaries to make the distribution of observation counts in the bins as close to a uniform distribution as possible. After an acceptable set of bin boundaries is found, PROC ECM computes the EDF estimate at the beginning of each bin. To compute a percentile, it uses these EDF estimates to do a quick search for the specified cumulative probability. The boundaries of the bins whose EDF estimates are closest to the specified cumulative probability are used to estimate the percentile. The section “[Percentile Computation Methods](#)” on page 1121 describes the different percentile computation methods.

The EDF estimation algorithm is driven by the following main tuning parameters that you can control:

- **Start of the tail region:** This is expressed as the cumulative probability, p_t , at the start of the tail. The first stage of the algorithm performs an initial EDF-axis binning to estimate l_t , which is the $100p_t$ th percentile. PROC ECM uses that percentile to demarcate the body and tail regions of the distribution. All loss values that are larger than l_t are assigned to the tail region, and the rest of the values are assigned to the body region.

You can provide a guidance for the value of p_t by specifying the `TAILSTARTEDF=` option. PROC ECM uses your specified value to decide the number of bins that the first stage uses. In fact, it also uses the initial set of bins that the first stage finds to further demarcate multiple subregions within the body and tail regions, and it assigns those subregions to different worker nodes to process in parallel. The number of bins in the first stage depends not only on p_t but also on the number of worker nodes that are available to PROC ECM and the sampling fraction for the body region, because PROC ECM attempts to distribute the required binning computations equitably among all the available worker nodes. The actual EDF value at the start of the tail might be different than p_t .

- **Desired EDF accuracy:** This is expressed as the desired width of each bin along the EDF axis, which in turn is the fraction of observations that each bin should have. The second stage of the algorithm uses this value to estimate the number of bins it needs to create. The smaller the bin width, the larger the accuracy, the larger the number of bins, and possibly the slower the estimation process. If your specified bin width is so small—that is, your desired EDF accuracy is so high—that the target number of observations in each bin is smaller than five, then PROC ECM does not use the iterative binning algorithm in the second stage. Instead, it uses the sorted set of all the loss values in the region as bin boundaries.

You can specify different accuracy values for the body and tail regions of the distribution by using the `EDFACCURACY=` and `TAILEDFACTURACY=` options, respectively. If you are interested primarily in the tail region of the distribution, which is typically the case for financial losses, you can specify a relatively lower accuracy (larger bin width) for the body region to help speed up the algorithm.

- **Acceptable tolerance in the EDF accuracy:** This is expressed as a fractional multiplier, ϵ ($0 < \epsilon < 1$), of the accuracy. The algorithm attempts to make the difference in the fractions of observations in the

largest and smallest bins less than or equal to $\delta\epsilon$, where δ is the EDF accuracy of the body or tail region that is being analyzed. You can specify ϵ by using the **TOLERANCE=** option.

As the first test of convergence of the binning algorithm, PROC ECM uses Pearson's chi-squared test to test the null hypothesis that the distribution of counts in all the bins is same as the uniform distribution. However, in practice, even after the chi-squared test fails to reject the null hypothesis—that is, the p -value is larger than the usual thresholds of 0.05 or 0.1—the bin boundaries can often be adjusted further to reduce the difference in the fractions of the largest and smallest bins. PROC ECM enables you to control for this difference so that you can further improve the accuracy of the EDF and consequently the percentile estimates.

- **Sampling fraction for the body region:** The bin boundaries that are computed in the first stage of the algorithm demarcate multiple subregions of the body and tail regions. PROC ECM shuffles all values in the sample to these subregions so that they can be processed in parallel on all available worker nodes. When the sample is large and distributed, shuffling can be expensive. So sampling the body region (which typically contains a large fraction of the total observations) helps reduce the amount of data that need to be processed and communicated among different nodes. Of course, sampling reduces the accuracy in the body region. But for large samples, the loss of accuracy is not as significant as the gains that are produced by reducing the amount of data to process. You can control the sampling fraction by specifying the **SAMPLEFRACTION=** option. Because the sampling process is driven by an internal pseudorandom number generator, you can specify the **SEED=** option to obtain similar results across multiple runs of PROC ECM for identical inputs and identical grid configuration. **NOTE:** The CAS server does not guarantee row-order consistency for the same data table that is created or loaded in different CAS sessions, so the results might vary slightly even if you specify the same seed.

The algorithm never samples the tail region in order to acknowledge the fact that most interesting financial losses fall in the tail region.

In addition to the preceding set of main tuning parameters, you can also use the **EXACTFINALCOUNT** and **NOSHUFFLE** options to control the EDF estimation process.

Percentile Computation Methods

The ECM procedure computes a percentile by using the bins that it estimates for each random variable's sample. It uses the counts in each bin to estimate the value of the empirical distribution function (EDF) at the beginning of each bin. Let $\hat{F}_i$ denote the EDF at the beginning of the i th bin. The algorithm estimates B bins from a sample of size N . Assume $\hat{F}_i = 1$ for $i > B$. For the t th percentile, let $p = \frac{t}{100}$. PROC ECM first finds the i th bin that satisfies $\hat{F}_i < p \leq \hat{F}_{i+1}$. Then it computes a ratio g as

$$g = \frac{p - \hat{F}_i}{\hat{F}_{i+1} - \hat{F}_i}$$

Let b_i denote the value at which the i th bin begins. Assume $b_i = x_{\max}$ for $i > B$, where $x_{\max}$ is the maximum value in the sample. Let j denote the integer part of Np . Then the **PCTLDEF=** option defines the t th percentile, y , as described in Table 18.2.

Table 18.2 Percentile Definitions

PCTLDEF=	Description	Formula
1	Weighted average	$y = (1 - g)b_i + gb_{i+1}$
2	Value closest to Np	$y = b_i$ if $g < \frac{1}{2}$ $y = b_i$ if $g = \frac{1}{2}$ and j is even $y = b_{i+1}$ if $g = \frac{1}{2}$ and j is odd $y = b_{i+1}$ if $g > \frac{1}{2}$
3	Empirical distribution function	$y = b_i$ if $g = 0$ $y = b_{i+1}$ if $g > 0$
4	Weighted average (identical to PCTLDEF=1)	$y = (1 - g)b_i + gb_{i+1}$
5	Empirical distribution function with averaging	$y = \frac{1}{2}(b_i + b_{i+1})$ if $g = 0$ $y = b_{i+1}$ if $g > 0$

Note that each PCTLDEF= method is similar to the corresponding method that is used by the UNIVARIATE procedure in Base SAS. For more information, see *Base SAS Procedures Guide: Statistical Procedures*. The main differences are in the definition of g and in the use of the bin boundaries instead of the ordered sample values.

Input Specification

PROC ECM accepts as input the copula simulation data table (which you specify in the DATA= option in the PROC ECM statement) and marginal data tables (which you specify in the DATA= option in the MARGINAL statements). This section describes the information that these tables are expected to contain.

Copula Simulation Data Table

The copula simulation data table is expected to contain the output of the copula simulation at a uniform scale. It must contain all the marginal variables that you want to analyze. Each observation contains the probability of a loss value for each of the marginal variables; this probability is expected to be generated by accounting for the dependence structure between the marginal variables.

This is a required data table; you must specify it in the DATA= option in the PROC ECM statement.

The copula table is usually generated by specifying the OUTUNIFORM= option in the SIMULATE statement in the CCOPULA procedure for a dependency structure that is either defined by you (in the DEFINE statement in PROC CCOPULA) or estimated by PROC CCOPULA (according to the FIT statement in PROC CCOPULA).

Marginal Data Tables

The marginal data tables that you specify in the MARGINAL statements are expected to contain an empirical sample of the corresponding marginal variable. This is a required data table; you must specify it for each marginal that you want to analyze.

The empirical sample must appear in a variable that has one of the following names:

- the name of the marginal variable
- the name that you specify in the **SAMPLEVAR=** option in the MARGINAL statement
- **_AGGSEV_**

If a marginal table contains a variable named **_DRAWID_**, then the table is assumed to contain multiple empirical samples of the marginal variable. The value of the **DRAWID=** option in the MARGINAL statement or the value of the **DRAWID=** option in the PROC ECM statement determines which sample to use for the analysis.

Output Data Tables

PROC ECM writes the output data tables that you specify in the **OUT=** option of the **OUTPUT** and **OUTSUM** statements. The contents of these output data tables are described in the following sections.

OUTSAMPLE= Data Table

The OUTSAMPLE= data table records the sample of the total loss. It contains as many observations as the number of valid observations in the copula simulation data table. The OUTSAMPLE= data table contains the following variables:

Total-loss-variable

indicates the value of the total loss. The name of this variable is the value of the **TOTALLOSSVAR=** option in the OUTPUT statement. If you do not specify the **TOTALLOSSVAR=** option, then this variable is named **_TOTALLOSS_**.

The total loss for each observation is computed as the sum of the loss estimates of individual marginals. The loss estimate of each marginal is computed by inverting the estimated EDF at the probability of that marginal.

If you specify the **MARGINALSOUT** option in the OUTPUT statement, then the table also contains the following variables:

Probability-of-marginal-variable-1 ... Probability-of-marginal-variable-D

indicates the probability values of the *D* marginal variables, where *D* is the number of marginal variables that PROC ECM analyzes. The names of these variables are the *marginal-variable* values that you specify in the respective MARGINAL statements. The value of each variable is identical to the value of the respective variable in the copula simulation data table.

Loss-for-marginal-variable-1 ... Loss-for-marginal-variable-D

indicates the estimates of the losses of the D marginal variables, where D is the number of marginal variables that PROC ECM analyzes. The names of these variables are formed by adding a suffix of `_Loss` to the name of the corresponding marginal variable. The loss for a marginal variable is the percentile estimate for the probability value of that marginal.

OUTSUM= Data Table

The OUTSUM= data table records the estimates of summary statistics, percentiles, and tail values-at-risk (TVaR) of the estimated total loss. You can control the statistics that appear in this data table by specifying the PCTLPTS= option, the TVARPTS= option, and keywords for different statistics in the OUTSUM statement.

If you specify the PCTLPTS= or TVARPTS= option in the OUTSUM statement, then the OUTSUM= data table contains percentile estimates for all the default percentile levels (1, 5, 10, 25, 50, 75, 90, 95, 99, and 99.5) in addition to the percentile levels that you specifically request.

For more information about the method of naming the variables that correspond to the summary statistics, percentiles, and TVaR estimates, see the description of the OUTSUM statement.

Displayed Output

The ECM procedure optionally produces displayed output by using the Output Delivery System (ODS). You control all output by using the PRINT= option in the PROC ECM statement or by using the DISPLAY statement. Table 18.3 relates the PRINT= options to ODS tables.

Table 18.3 ODS Tables Produced by PROC ECM

ODS Table Name	Description	Option
DataSummary	Summary of the marginal or total loss sample	PRINT=DATASUMMARY
EDFInit	Initialization summary of the EDF estimation process	PRINT=EDFINIT
EDFInitOptDetails	Optimization details of the initialization of the EDF estimation process	PRINT=EDFINIT
EDFOptDetails	Optimization details of the EDF estimation process	PRINT=EDFOPTDETAILS
EDFSummary	Summary of the EDF estimation process	PRINT=EDFSUMMARY
Percentiles	Percentiles of the total loss sample	PRINT=PERCENTILES in the PROC ECM statement and PCTLPTS= in the OUTSUM statement

Table 18.3 *continued*

ODS Table Name	Description	Option
SummaryStatistics	Summary statistics of the total loss sample	PRINT=SUMMARYSTATISTICS
TVaR	Tail value-at-risk (TVaR) estimates of the total loss sample	PRINT=TVAR in the PROC ECM statement and TVARPTS= in the OUTSUM statement
Timing	Timing information for various computational stages of the procedure	PRINT=ALL

If you do not specify the PRINT= option and if you do not specify the NOPRINT, PRINT=NONE, or PRINT=ONLY options, then by default PROC ECM produces the SummaryStatistics ODS table. If you specify the PCTLPTS= option in the OUTSUM statement, then the default displayed output also includes the Percentiles ODS table. If you specify the TVARPTS= option in the OUTSUM statement, then the default displayed output also includes the Percentiles and TVaR ODS tables.

ODS Table Details

Table 18.4 provides detailed descriptions of the tables that different PRINT= options display.

Table 18.4 Description of ODS Tables

ODS Table Name	Detailed Description
DataSummary	The “Data Summary” table contains the summary of each sample for which PROC ECM estimates the EDF.
EDFInit	The “EDF Binning Initialization” table contains, for each marginal variable, the summary of the first (initialization) stage of the EDF estimation process. This summary includes the number of bins that the first stage uses and the convergence status of the optimization process of the first stage. It also reports the estimated empirical distribution function (EDF) at the start of the tail region and the tail start value. Finally, it reports whether the second stage of the binning algorithm processes different regions of the data range in parallel by distributing the bins among worker nodes. If you specify the PCTLPTS= or TVARPTS= option in the OUTSUM statement, PROC ECM displays this table also for the total loss variable’s sample.

Table 18.4 *continued*

ODS Table Name	Detailed Description
EDFInitOptDetails	The “EDF Binning Initialization Details” table contains, for each marginal variable, the details of the optimization process of the first page. The table reports the desired proportion of counts in each initial bin, the number of iterations, and the final value of the optimization criterion (which is the difference in proportions of the bins that have the largest and the smallest counts at the end of the first stage). The table also reports the chi-square test statistic (along with its degrees of freedom) and the p-value (which indicates how close the final distribution of counts in the bins is to the desired uniform distribution). Typically, a p-value that is larger than 0.05 indicates that the final distribution of bin counts is not statistically significantly different from a uniform distribution. If you specify the PCTLPTS= or TVARPTS= option in the OUTSUM statement, PROC ECM displays this table also for the total loss variable’s sample.
EDFOptDetails	The “EDF Binning Optimization Details” table contains, for each marginal variable, the details of the optimization process that PROC ECM runs in each region that does not use data points to form bins. For each region, the table reports the desired proportion, the number of iterations, and the final value of the optimization criterion, which is the difference in proportions of the bins that have the largest and the smallest counts at the end of the optimization. The table also reports the chi-square test statistic along with its degrees of freedom and the p-value. Typically, a p-value that is larger than 0.05 indicates that the final distribution of bin counts in that region is not statistically significantly different from the desired uniform distribution. If you specify the PCTLPTS= or TVARPTS= option in the OUTSUM statement, PROC ECM displays this table also for the total loss variable’s sample.
EDFSummary	The “EDF Binning Summary” table contains, for each marginal variable, the summary of the regions for which PROC ECM processes the bins independently. For each region, it reports the range and location (body, tail, or body-tail), the number of observations in that region, and the number of bins that are formed in that region. If the status for a region indicates that PROC ECM has used the data points to form bin boundaries, then PROC ECM does not run the optimization process for that region and that region’s information does not appear in the “EDF Binning Optimization Details” table. For all other regions, PROC ECM runs the optimization process, reports the convergence status of the optimization process in this table, and reports the details of the optimization process in the “EDF Binning Optimization Details” table. If you specify the PCTLPTS= or TVARPTS= option in the OUTSUM statement, PROC ECM displays this table also for the total loss variable’s sample.

Table 18.4 *continued*

ODS Table Name	Detailed Description
Percentiles	The “Percentiles” table contains estimates of the percentiles of the total loss sample. This table includes the default set of percentiles and the percentiles that you request in the OUTSUM statement.
SummaryStatistics	The “Summary Statistics” table contains estimates of the following summary statistics of the total loss sample: number of observations, maximum value, minimum value, mean, standard deviation, variance, skewness, and kurtosis. If you specify the PCTLPTS= or TVARPTS= option in the OUTSUM statement, the table also contains the estimates of median and interquartile range.
TVaR	The “Tail Value-at-Risk Estimates” table contains the tail value-at-risk (TVaR) estimates for the percentile levels that you specify in the TVARPTS= option. The table also reports the value-at-risk (VaR) estimate for each percentile level. It also includes the probability that a loss value exceeds the VaR estimate, which is equal to $1 - p/100$ for a percentile level p .

Examples: ECM Procedure

Example 18.1: Developing an Economic Capital Model

This example illustrates all the steps of developing an economic capital model by using SAS Econometrics procedures, including the ECM procedure. The section “[Overview: ECM Procedure](#)” on page 1098 describes the process in detail. Following is a quick summary of all the steps in the process:

1. Collect and prepare data.
2. For each line of business (LoB), do the following:
 - a) Fit and select the best frequency model for the number of loss events in a particular time period.
 - b) Fit and select the best severity model for the magnitude of each loss.
 - c) Estimate a compound distribution model (CDM) by combining the best frequency and severity models to create a large sample of the aggregate loss.
3. Estimate a dependency structure among the losses of the LoBs by fitting and selecting the best copula model.
4. Use the best copula model to simulate a large sample of probabilities of loss values that are observed in all LoBs in the same time period.

5. Estimate an economic capital model by combining the copula simulation sample with the aggregate loss samples of all LoBs.

The following subsections illustrate in detail each of these steps by developing a simple ECM for a bank's operational risk.

Data Preparation

Consider that you are a bank and, as part of creating an economic capital model for your operational risk, you want to analyze losses in two LoBs: retail banking and commercial banking. In addition to collecting the loss frequency and loss severity data for each LoB, you should also collect data about the key risk indicators (KRIs), which are the external factors that you believe might affect the frequency and severity of losses in at least one LOB. Let CorpKRI1, CorpKRI2, CbKRI1, CbKRI2, CbKRI3, RbKRI1, RbKRI2, and RbKRI3 be the available KRIs, where the prefix Corp indicates corporate-level KRIs, the prefix Cb indicates KRIs specific to commercial banking, and the prefix Rb indicates KRIs specific to retail banking. Some examples of corporate-level KRIs are the ratio of temporary employees to permanent employees and the number of security breaches that are reported during a year. Some examples of commercial banking KRIs are the number of credit defaults, the proportion of financed assets that are movable, and the penalty claims against your bank that can result from processing delays. Some examples of retail banking KRIs are the number of credit cards that are reported stolen, the fraction of employees who have not undergone fraud detection training, and the number of forged drafts and checks that are presented in a year. To mimic the real-world applications, let different yet overlapping sets of KRIs be used for the frequency and severity models. Let the analysis period be one month, and let the data preparation process result in the following data tables:

- The Work.OpRiskLossCounts data table contains the frequency regressors and the variable NumLoss, which records the number of losses that an LoB incurs in a month. The LoBs are identified by the BY variable Line.
- The Work.OpRiskLosses data table contains the severity regressors and the variable LossValue, which records the severity of each loss. The LoBs are identified by the BY variable Line.
- The Work.OpRiskMatchedLosses data table contains the aggregate losses of all LoBs, where the losses in each LoB are aggregated by month. These matched losses are needed for copula modeling.

The following DATA steps prepare the preceding three data tables:

```

/*** Simulate loss counts data ***/
data OpRiskLossCounts(keep=line corpKRI1 corpKRI2
                      cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 numloss);
  call streaminit(12345);
  array cx{8} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 cb23;
  array cbetaR{9} _TEMPORARY_ (0.1 1 -0.5 0 0 0 0.5 0.75 0);
  array cbetaC{9} _TEMPORARY_ (0.5 0.75 0.3 -0.25 0.25 0.5 0 0 0.3);

  alpha = 0.35; theta = 1/alpha;
  call streaminit(12345);
  do obs=1 to 1000;
    do i=1 to dim(cx)-1;
      cx(i) = rand('NORMAL');
    end;

```

```

cb23 = cx(4) * cx(5);

line = 'CommercialBanking';
xbeta = cbetaC(1);
do i=1 to dim(cx);
    xbeta = xbeta + cx(i) * cbetaC(i+1);
end;
Mu = exp(xbeta); p = theta/(Mu+theta);
numloss = rand('NEGB',p,theta);
output;

line = 'RetailBanking';
xbeta = cbetaR(1);
do i=1 to dim(cx);
    xbeta = xbeta + cx(i) * cbetaR(i+1);
end;
lambda = exp(xbeta);
numloss = rand('POISSON', lambda);
output;
end;
run;

/** Simulate severity of losses and matched loss data */
data OpRiskLosses(keep=line corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 lossValue)
    OpRiskMatchedLosses(keep=CB RB);
    array sx{6} corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 corplrb3;
    array sbetaC{7} _TEMPORARY_ (5 1 0 0.3 0 0 0);
    array sbetaR{7} _TEMPORARY_ (3.5 0 0.5 0 -0.8 0.6 0.2);
    retain CB RB sigma alpha;
    if (_n_=1) then do;
        call streaminit(67890);
        sigma = 1.0; /* for lognormal */
        alpha = 3.0; /* for gamma */
    end;
    set OpRiskLossCounts(keep=line numloss corpKRI1 corpKRI2 cbKRI2 rbKRI1);
    sx(5) = rand('NORMAL'); /* simulate rbKRI3 value */
    corplrb3 = sx(1) * sx(5);

    if (line='CommercialBanking') then do;
        CB = 0;
        do il=1 to numloss;
            /* lognormal */
            Mu = sbetaC(1);
            do i=1 to dim(sx);
                Mu = Mu + sx(i) * sbetaC(i+1);
            end;
            lossValue = exp(Mu) * rand('LOGNORMAL')**Sigma;
            CB = CB + lossValue;
            output OpRiskLosses;
        end;
    end;
    else do; /* line='RetailBanking' */
        RB = 0;
        do il=1 to numloss;

```

```

        /* gamma */
        Mu = sbetaR(1);
        do i=1 to dim(sx);
            Mu = Mu + sx(i) * sbetaR(i+1);
        end;
        lossValue = rand('Gamma', Alpha, exp(Mu));
        RB = RB + lossValue;
        output OpRiskLosses;
    end;
    output OpRiskMatchedLosses;
end;
run;

```

The following DATA steps load the data tables in your session that is associated with the mylib libref. This example assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```

/** Load data tables into the session */
data mylib.OpRiskLossCounts;
    set OpRiskLossCounts;
run;
data mylib.OpRiskLosses;
    set OpRiskLosses;
run;
data mylib.OpRiskMatchedLosses;
    set OpRiskMatchedLosses(where=(CB > 0 and RB > 0));
run;

```

When your frequency or severity models contain regressors, you must also create an external scenario data table to contain values that you expect to observe for each regressor in some future state. The final CDM and ECM estimates depend on this scenario data table. For this example, the scenario data table is named Work.MultiConditionScenario, and it contains observations for 12 months of a future year. The following DATA steps simulate it and load it into your session:

```

/** Generate a scenario data table for multiple operating conditions */
data multiConditionScenario (keep=line opEnvId corpKRI1 corpKRI2
    cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3);
    array x{8} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3;
    call streaminit(5151);
    do opEnvId=1 to 12;
        do i=1 to dim(x);
            x(i) = rand('NORMAL');
        end;
        line = 'CommercialBanking';
        output;
        line = 'RetailBanking';
        output;
    end;
run;
proc print; run;

/** Load the scenario data table into the session */
data mylib.multiConditionScenario;
    set multiConditionScenario;
run;

```

After this critical step of collecting and preparing the data, you are ready for the subsequent modeling and simulation steps.

Frequency (Count) Modeling

The CNTSELECT procedure in SAS Econometrics estimates frequency models. It fits a count regression model, where the dependent variable is a count variable that, conditional on the regressors, has one of the following discrete probability distributions: Poisson, negative binomial with linear or quadratic variance, or Conway-Maxwell-Poisson (CMP). It can also fit a zero-inflated version of each distribution, enabling you to account for the large number of zeros that are often present in real-life loss count data. For detailed information about all the features of PROC CNTSELECT, see Chapter 10, “[CNTSELECT Procedure](#).”

The following statements use the CNTSELECT procedure to fit Poisson and negative binomial distribution models to the example data:

```

/** Fit Poisson count models */
proc cntselect data=mylib.OpRiskLossCounts store=mylib.opriskStorePoisson
  method=CONGRA;
  by line;
  model numloss=corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 cbKRI2*cbKRI3
    rbKRI1 rbKRI2/dist=poisson;
  selection;
  ods output SelectedModel.FitModelSummary=fitPoisson;
run;

/** Fit negative binomial count models */
proc cntselect data=mylib.OpRiskLossCounts store=mylib.opriskStoreNegbin
  method=CONGRA;
  by line;
  model numloss=corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 cbKRI2*cbKRI3
    rbKRI1 rbKRI2/dist=negbin(p=2);
  selection;
  ods output SelectedModel.FitModelSummary=fitNegbin
    SelectedModel.ParameterEstimates=estNegbin;
run;

```

In the PROC CNTSELECT step, the BY statement ensures that the specified model is analyzed for both LoBs. The MODEL statements specify all the possible regression effects that you believe can affect the number of losses in all LoBs. The SELECTION statements perform model (variable) selection independently for each BY group. The default settings of the SELECTION statement choose the best subset of regression effects to maximize the Schwarz Bayesian criterion (SBC). The ODS OUTPUT statements store the fit summary of the selected model in the Work.FitPoisson and Work.FitNegbin data sets. The following DATA step reads those two data sets to prepare a comparative fit summary:

```

/** Summarize count modeling results */
data countFitSummary;
  set fitPoisson (keep=line Property PropertyValue where=(Property='SBC'));
  Model='Poisson';
  output;
  set fitNegbin (keep=line Property PropertyValue where=(Property='SBC'));
  Model='NegBin(p=2)';
  output;
run;

```

[Output 18.1.1](#) shows the comparison of the fit summary of the final count models of both business lines. It indicates that the negative binomial model is the best model for the commercial banking LoB, because it has a smaller SBC value than the Poisson model. For the retail banking LoB, the Poisson model is

the best model. Occasionally, both the Poisson and negative binomial models might have identical SBC values. This usually implies that the negative binomial model is equivalent to the Poisson model, because the overdispersion parameter α is close to 0. Formally, the conditional variance of the negative binomial distribution is $V(y|x) = \mu + \alpha\mu^2$, where μ denotes the mean. As α becomes closer 0, the variance becomes closer to the mean, making it equivalent to a Poisson distribution. The estimate of the `_Alpha` parameter in [Output 18.1.2](#) confirms this.

Output 18.1.1 Comparing Fit Statistics of Count Models

line	Model	Property	PropertyValue
CommercialBanking	Poisson	SBC	4010.42
CommercialBanking	NegBin(p=2)	SBC	3608.513
RetailBanking	Poisson	SBC	2940.835
RetailBanking	NegBin(p=2)	SBC	2940.835

Output 18.1.2 Parameter Estimates of Negative Binomial Count Model (Retail Banking)

The CNTSELECT Procedure

Selected Model

line=RetailBanking

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	0.124733	0.032401	3.85	0.0001
corpKRI1	1	0.989701	0.019687	50.27	<.0001
corpKRI2	1	-0.504275	0.019627	-25.69	<.0001
rbKRI1	1	0.522683	0.019862	26.32	<.0001
rbKRI2	1	0.712896	0.020307	35.11	<.0001
_Alpha	1	1.0536712E-8	.	.	.
Restrict 1	-1	6273.095945	.	.	.

The `STORE=` options in each PROC CNTSELECT step create item-store data tables that store all the model fit information, including the selected model's parameter estimates, which are used to estimate the compound distribution model in a later step.

Severity Modeling

The SEVSELECT procedure in SAS Econometrics estimates severity models. In the simplest form, it fits multiple continuous probability distributions to the input loss severity data (which are possibly censored or truncated), and prepares a comparative summary of the fit statistics of all candidate distributions to help you choose the best candidate. PROC SEVSELECT includes a default set of 10 distributions: Burr, exponential, gamma, generalized Pareto (GPD), inverse Gaussian (Wald), lognormal, Pareto, scaled Tweedie, Tweedie, and Weibull. However, you can define and fit your own distributions, such as mixture distributions. PROC SEVSELECT models the effect of regressors on the scale parameter or the log-scale parameter of the distribution, and it supports automatic model (variable) selection. For detailed information about all the features of PROC SEVSELECT, see Chapter 26, “[SEVSELECT Procedure](#).”

The following PROC SEVSELECT step fits severity models for three candidate distributions—gamma, lognormal, and Weibull—to the example data:

```

/** Fit severity models */
proc sevselect data=mylib.OpRiskLosses store=mylib.opriskSevstore(promote=yes)
            covout print=(allfitstats);

    by line;
    loss lossValue;
    dist gamma logn weibull;
    scalemodel corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 corpKRI1*rbKRI3;
    selection method=forward(select=aicc);

run;

```

The LOSS statement specifies the severity loss variable. The DIST statement specifies the list of candidate distributions. The STORE= option names an item store to hold the final model specifications and their parameter estimates. The COVOUT option requests that the parameter covariance estimates be kept in the item store for use in a later step. The SCALEMODEL statement lists all candidate regression effects. The SELECTION statement requests that the best subset of regression effects be chosen according to the forward selection method, where candidate subsets are compared using the corrected Akaike's information criterion (AICC). Within each BY group, PROC SEVSELECT runs the regression effect selection process independently for each candidate distribution.

After the selection process completes for all distributions, PROC SEVSELECT prepares a comparative summary of the fit statistics of the final selected scale regression models of all candidate distributions, as shown in [Output 18.1.3](#). There are two classes of fit statistics: likelihood-based (AIC, AICC, and SBC) and EDF-based (KS, AD, CvM). Akaike’s information criterion (AIC) is an asymptotic version of AICC. The Kolmogorov-Smirnov (KS), Anderson-Darling (AD), and Cramér–von Mises (CvM) statistics are the EDF-based statistics, which measure the distance between the CDF and the EDF. For scale regression models, the CDF depends on the values of the regressors, which can vary from one observation to another, making it difficult to interpret the EDF-based fit statistics. So you should use the likelihood-based statistics to compare the scale regression models. For the example data, the lognormal distribution is the best for the commercial banking line, and the gamma distribution is the best for the retail banking line, according to all the likelihood-based fit statistics.

Output 18.1.3 Comparison of Statistics of Fit for the Final Selected Severity Models

The SEVSELECT Procedure

Weibull Distribution Selected Model

line=CommercialBanking

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Gamma	46964	46974	46974	47004	5.41882	190.50419	11.56643
Logn	46471*	46481*	46481*	46512*	5.09394	153.98827	12.20540
Weibull	46986	46996	46996	47027	5.05018*	140.59352*	6.80075*

Asterisk (*) denotes the best model in the column.

Output 18.1.3 *continued*

line=RetailBanking

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Gamma	30851*	30865*	30865*	30908*	12.35872	1358	62.75081
Logn	30985	30999	30999	31041	11.22101*	1037*	60.10064
Weibull	30979	30993	30993	31036	12.61826	1690	56.20766*

Asterisk (*) denotes the best model in the column.

Compound Distribution Modeling

The CCDM procedure in SAS Econometrics estimates a compound distribution model (CDM) for the aggregate loss by combining the best frequency and severity models. Before illustrating the use of PROC CCDM, it is useful to define the CDM. If N and X represent the frequency and severity random variables, respectively, then the aggregate loss S is defined as $S = \sum_{j=1}^N X_j$. If $F_X(x)$ denotes the cumulative distribution function (CDF) of X and if $\Pr(N = n)$ denotes the probability of seeing n losses as per the frequency distribution, then the CDF of S is theoretically computable as $F_S(s) = \sum_{n=0}^{\infty} \Pr(N = n) \cdot F_X^{*n}(x)$, where $F_X^{*n}(x)$ denotes the n -fold convolution of the CDF of X . The probability distribution model for S , characterized by $F_S(s)$, is called a CDM. Direct computation of F_S is usually a difficult task because the n -fold convolution needs to be computed. Some methods exist for certain combinations of severity and frequency distributions (for example, Klugman, Panjer, and Willmot 1998, Ch. 4), but even those methods do not offer closed-form or easily computable expressions for F_S . Further, when the distributions of X and N are conditional on external factors (regressors), as is usually the case, each set of regressor values results in a different distribution for N and X . For all these reasons, PROC CCDM estimates the CDM by using a Monte Carlo simulation method. For detailed information about all the features of PROC CCDM, see Chapter 7, “CCDM Procedure.” An important feature that this example uses is PROC CCDM’s ability to generate multiple perturbed aggregate loss samples, each of which corresponds to a random perturbation of the frequency and severity parameters from their mean values according to the covariance structure that the CNTSELECT and SEVSELECT procedures estimate. This feature helps assess the uncertainty (variability) in the estimates of the CDM.

The following PROC CCDM steps estimate the CDM for each LoB. Because the best count model for each LoB is different for this example, you need two separate steps, each of which uses an appropriate count model item-store table in the COUNTSTORE= option.

```

/** Simulate CDM for commercial banking LoB */
proc ccdm data=mylib.multiConditionScenario
    countstore=mylib.opriskStoreNegbin severitystore=mylib.opriskSevstore
    seed=13579 nreplicates=10000 nperturb=50 print=all;
    by line;
    severitymodel logn;
    output out=mylib.aggregateLossCB samplevar=agglossCB / perturbOut;
run;

/** Simulate CDM for retail banking LoB */
proc ccdm data=mylib.multiConditionScenario
    countstore=mylib.opriskStorePoisson severitystore=mylib.opriskSevstore
    seed=13579 nreplicates=10000 nperturb=50 print=all;

```



```

by line;
severitymodel gamma;
output out=mylib.aggregateLossRB samplevar=agglossRB / perturbOut;
run;

```

When the frequency and severity models contain regression effects, you need to use the DATA= option to specify a data table that contains an external scenario that you want to analyze. This example uses a simulated scenario in the mylib.MultiConditionScenario data table. The NPERTURB= option requests that parameter perturbation analysis be conducted to generate 50 perturbed samples. The NREPLICATES= option specifies the size of each unperturbed and perturbed aggregate loss sample. The OUTPUT statement and its PERTURBOUT option request that all 51 samples be stored in a data table. PROC CCDM creates the output data table by using parallel computations and stores it in a distributed manner across all worker nodes. These samples are useful for estimating the uncertainty in the final ECM estimates in a later stage of the ECM process.

Output 18.1.4 shows one of the tables that the perturbation analysis produces for the commercial banking LoB. You can use this table to estimate the VaR for an individual LoB. The perturbation analysis gives you an estimate of the variability in each statistic of interest, which is more informative than just the estimate of the mean.

Output 18.1.4 Perturbation Analysis of Aggregate Losses (Commercial Banking LoB)

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

line=CommercialBanking

Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	5275.4	376.48300
5	7954.1	494.75748
25	13355.5	718.13076
50	18388.8	943.74904
75	24699.3	1245.3
95	36460.4	1929.3
99	47070.2	2483.3
99.5	51639.6	2679.0
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

Copula Modeling

The CCOPULA procedure in SAS Econometrics helps you analyze the dependency among losses of multiple LoBs. It supports the dependency structures that are expressed in terms of an elliptical copula (normal and t copulas) or an Archimedean copula (Clayton, Frank, and Gumbel copulas). A copula is especially useful for modeling the dependency structure in the ECM process, because it decouples the marginal distributions—that is, the probability distributions of individual random variables—from their joint probability distribution. This decoupling is based on an important theorem by Sklar (1959), which shows that a copula can be derived from

any joint distribution function and that any copula can be combined with any set of marginal distributions to result in a multivariate distribution function. PROC CCOPULA can both fit a copula model and simulate samples from a fitted copula model. For detailed information about all the features of PROC CCOPULA, see Chapter 8, “[CCOPULA Procedure](#).”

The following PROC CCOPULA steps fit four different types of copulas to the matched loss data in the data table mylib.OpRiskMatchedLosses:

```

/** Fit different copula models */
proc ccopula data=mylib.OpRiskMatchedLosses;
  var CB RB;
  fit Gumbel / store=mylib.copStoreG;
  ods output fitmodelsummary=fitsummG;
run;
proc ccopula data=mylib.OpRiskMatchedLosses;
  var CB RB;
  fit T / store=mylib.copStoreT;
  ods output fitmodelsummary=fitsummT;
run;
proc ccopula data=mylib.OpRiskMatchedLosses;
  var CB RB;
  fit Frank / store=mylib.copStoreF;
  ods output fitmodelsummary=fitsummF;
run;
proc ccopula data=mylib.OpRiskMatchedLosses;
  var CB RB;
  fit Clayton / store=mylib.copStoreC;
  ods output fitmodelsummary=fitsummC;
run;

```

In each PROC CCOPULA step, the VAR statement specifies the marginal variables, which are CB and RB for losses in the commercial and retail banking LoBs, respectively. The FIT statement specifies the type of the copula and names the item-store data table in which to store the specification and estimates of the copula parameters. The ODS OUTPUT statement stores the fit summary of the copula, which you can analyze to produce a comparative table of the SBC fit statistics of all copulas as shown in [Output 18.1.5](#). The table shows that the Gumbel copula has the best fit for the example data by virtue of having the smallest SBC value.

Output 18.1.5 Fit Statistics for Copula Models

Copula	SBC
Gumbel	-69.6587
T	-61.4826
Frank	-48.3310
Clayton	-25.3843

Copula Simulation

The next step in the ECM process is to simulate multiple future scenarios of loss values that might be observed in all LoBs in the same time period. Each scenario is represented by the probabilities of loss values in all LoBs while taking into account the dependency structure among their losses. The use of the OUTUNIFORM= option in the OUTPUT statement of the CCOPULA procedure helps you create a large number of such simulated scenarios. The following PROC CCOPULA step illustrates that for the best-fitting Gumbel copula for this example's data:

```

/** Simulate marginal probabilities on the uniform scale from the Gumbel copula */
proc ccopula;
    simulate cop / restore=mylib.copStoreG
                ndraws=10000 seed=234
                outuniform=mylib.CbRbProbs;
run;

proc print data=mylib.CbRbProbs(obs=5) noobs; run;

```

The SIMULATE statement specifies the mylib.copStoreG item store to read the estimates of the Gumbel copula's parameter. The NDRAWS= option simulates 10,000 observations from this copula. The OUTUNIFORM= option requests that the observations be generated in the uniform scale and stored in the mylib.CbRbProbs data table. PROC CCOPULA uses the multiple nodes of a CAS session to simulate and store the data in a distributed manner. The first few simulated observations in the data table are shown in [Output 18.1.6](#). Each observation contains the probability of a loss in each LoB.

Output 18.1.6 Example of Simulated Copula Observations

CB	RB
0.0585445576	0.317802304
0.9579833083	0.9688330154
0.2114479224	0.3521376759
0.3857146159	0.9856645119
0.7071269566	0.7585000789

Economic Capital Modeling: Combining Copula Simulation with CDM Samples

The final step of the ECM process is to estimate the distribution of the total loss across all LoBs. This requires transforming each observation in the copula simulation table (which contains the probability of a loss value in each LoB) to an observation that contains an estimate of the actual loss for each LoB. For a conceptual overview of this process, see the section “[Overview of the Algorithm](#)” on page 1119.

The ECM procedure is designed specifically to help you implement this step. The following PROC ECM step converts the loss probabilities in the data table mylib.CbRbProbs to a sample of the total loss across two marginals, CB and RB, for the commercial and retail banking LoBs, respectively:

```

/* Estimate an empirical model for the total loss */
proc ecm data=mylib.CbRbProbs tailstart=0.85 edfaccuracy=1.0e-2
    samplefraction=0.7 seed=789 print=all;
    marginal CB: data=mylib.aggregateLossCB samplevar=agglossCB;
    marginal RB: data=mylib.aggregateLossRB samplevar=agglossRB;
    outsum out=mylib.tlSummary mean stddev skew kurtosis qrange=iqr
    pctlpts=50, 90, 95, 97.5, 99, 99.5 pctlpre=VaR

```

```

    tvarspts=90, 95, 97.5, 99, 99.5 pctlpre=TVaR;
run;

```

The two MARGINAL statements specify the data tables that contain the aggregate loss samples of each LoB as created by the CCDM steps. The identifying name of the marginal that you specify in the MARGINAL statement must match the corresponding name that you use in the VAR statement of the CCOPULA procedure. Because the name of the variable that contains the sample is different from the name of the marginal, you need to specify the SAMPLEVAR= option in each MARGINAL statement.

PROC ECM uses the loss sample in each marginal's data table to estimate the empirical distribution function (EDF) of that marginal. The EDF estimation process works in two stages. The first stage divides the sample values into body and tail regions. For this example, the TAILSTART= option requests that the tail region start at a value whose EDF is approximately 0.85. The actual EDF value at the start of the tail region also depends on the number of worker nodes and the sampling fraction for the body region. The second stage estimates the EDF in each region independently by using the desired accuracy of EDF estimates in each region. By default, PROC ECM attempts to achieve higher accuracy in the tail region. So for this example, you should expect more accurate loss estimates for probability values that are larger than approximately 0.85. The body region can afford to have a relatively lower EDF accuracy. For this example, the EDFACCURACCY= option specifies it to be 0.01, which means that the width of the bins along the EDF axis is approximately 0.01 in the body region. In particular, when PROC ECM processes an interval $[p_1, p_2]$ of the EDF axis, it attempts to bin the data into approximately B bins, where $B \approx (p_2 - p_1)/0.01$. The SAMPLEFRACTION= option requests that only about 70% of the observations in the body region be used for estimating the EDF in that region. This helps reduce the amount of data movement among worker nodes. The SEED= option ensures that the results are similar across multiple runs of the procedure with identical inputs.

For each observation in the DATA= data table, PROC ECM computes the percentile for the probability value of each marginal by using the estimated EDF of that marginal. Adding these percentiles across all marginals results in the estimate of the total loss for that observation. Repeating this process for all observations results in the sample of the total loss, which encodes the probability distribution of the total loss, which in turn is an empirical estimate of the economic capital model (ECM). The OUTSUM statement specifies the summary statistics and various percentile levels of risk measures, VaR and TVaR, that you want to compute for the estimated ECM. The OUT= option requests that the computed statistics and risk measures be stored in the data table mylib.tlSummary.

The PRINT=ALL option in the PROC ECM statement displays all the result tables that PROC ECM produces. For each marginal, up to five result tables are displayed, as illustrated in [Output 18.1.7](#), [Output 18.1.8](#), and [Output 18.1.9](#) for the commercial banking marginal CB.

[Output 18.1.7](#) shows the “Data Summary” table, which summarizes the data sample that PROC ECM reads for CB. It displays the data table name; the number of total, valid, and zero-valued observations; and the range of the data. When the input data table is a result of the parameter perturbation analysis that PROC CCDM conducts (as in this example), it contains multiple samples, each identified by a unique value of the _DRAWID_ variable. The “Data Summary” table displays the draw identifier, which is the value of the _DRAWID_ variable it uses to read the marginal sample. You can control this value by specifying a global value in the DRAWID= option in the PROC ECM statement or a marginal-specific value in the DRAWID= option in a MARGINAL statement. Similarly, the input data table can contain samples for multiple severity distributions when you specify multiple distributions in the SEVERITYMODEL statement of PROC CCDM. PROC ECM chooses the first severity distribution that it finds and displays its name in the “Data Summary” table.

Output 18.1.7 Data Summary (Commercial Banking)**The ECM Procedure****Marginal Variable CB**

Data Summary	
Data Table	AGGREGATELOSSCB
Severity Model	Logn
Draw Identifier	0
Observations	20000
Valid Observations	20000
Zero-valued Observations	0
Minimum	233.557351
Maximum	135451

Output 18.1.8 shows the tables that provide an insight into the first stage of the EDF estimation algorithm for the commercial banking marginal CB.

Output 18.1.8 Summary of the First Stage of EDF Estimation (Commercial Banking)

EDF Binning Initialization					
Number of Bins	7				
Convergence Status	Converged				
Tail Start EDF	0.857650				
Tail Start	23510				
Bins Distributed Among Workers	No				

EDF Binning Initialization Details					
Target	Proportion	Iterations	Criterion	DF	Chi-Square
					Pr > ChiSq
0.142857	5	0.0033000	6	1.4458	0.9630

Following is a description of the contents of each table in Output 18.1.8:

- “EDF Binning Initialization”: This table summarizes the result of the first-stage binning by displaying the number of bins and the status of the iterative bin boundary optimization algorithm. It also shows the EDF value and the actual loss value at the start of the tail region. Finally, it shows whether the bins are distributed among worker nodes at the end of the first stage.
- “EDF Binning Initialization Details”: This table displays the details of the iterative bin boundary optimization algorithm of the first stage. For B number of bins (which is 7 for this marginal), the algorithm attempts to achieve a target proportion of $1/B$ observations in each bin. After several iterations, the maximum difference in fractions of observations in the largest and smallest bins is the value of Criterion. The chi-square test for uniformity of counts in the bins uses $B - 1$ degrees of freedom. The value of the test statistic and its corresponding p -value show that the final distribution of counts in bins is close to the uniform distribution.

Output 18.1.9 shows the tables that provide an insight into the second stage of the EDF estimation process for the commercial banking marginal CB.

Output 18.1.9 Summary of the Second Stage of EDF Estimation (Commercial Banking)

EDF Binning Summary						
Region	Begin	End	Location	N	Bins	Status
1	233.55735	23509.7	Body	17153	86	Converged
2	23509.7	135451.4	Tail	2847	2848	Data points used as bins

EDF Binning Optimization Details						
Target						
Region	Proportion	Iterations	Criterion	DF	Chi-Square	Pr > ChiSq
1	0.011628	1	0.0044890	85	53.1719	0.9973

Following is a description of the contents of each table in [Output 18.1.9](#):

- “EDF Binning Summary”: This table displays the different subregions of values that the second stage processes in parallel. Typically, the number of rows in this table is proportional to the number of worker nodes. For each subregion, the table displays whether the subregion is located in the body or tail region of the distribution. You might also occasionally see the ‘Body-Tail’ location value, which indicates that a subregion straddles the body and tail regions. Next, the table displays the number of observations that are processed and the number of bins that are used. The number of observations in the body subregions reflect the sampling fraction. The number of bins reflects the specified EDF accuracy. Finally, the last column displays the status of the binning process for that subregion. If the number of bins is small enough to hold at least five observations in each bin, an iterative binning algorithm is used and its convergence status is displayed. If the desired accuracy of the region is too high, the number of bins is large and the algorithm uses data points as bin boundaries, as the table shows for the tail region.
- “EDF Binning Optimization Details”: This table displays the details of the iterative bin boundary optimization algorithm for each subregion that does not use data as bin boundaries. The Region column identifies the region, and the remaining columns are similar to the columns of the “EDF Binning Initialization Details” table. For this example, the binning algorithm needs very few iterations in both subregions to achieve a nearly uniform distribution of counts.

If the algorithm decides to use data points as bin boundaries in all the subregions, then this table is not displayed.

PROC ECM also executes the same EDF estimation algorithm for the retail banking marginal RB, and the PRINT=ALL option displays up to five similar result tables for that marginal. Further, because the OUTSUM statement requests ECM statistics that require percentile-based estimates, PROC ECM also executes the same algorithm to estimate the EDF of the generated total loss sample and then uses that EDF estimate to compute the required percentiles and percentile-based risk measures. The PRINT=ALL option displays up to five similar result tables, which contain the summary of the total loss sample and information about the first and second stages of the EDF estimation algorithm for that sample.

After displaying the results of the EDF estimation algorithms for all marginals and the total loss sample, PROC ECM displays the summary statistics and requested risk measures for the total loss sample. [Output 18.1.10](#) shows the “Summary Statistics” and “Tail Value-at-Risk Estimates” tables for this example.

Output 18.1.10 Summary Statistics and Estimates of Risk Measures (VaR and TVaR) of the ECM

Summary Statistics			
Sample Size	10000		
Minimum	1921.8		
Maximum	122147.2		
Mean	23061.3		
Standard Deviation	15640.3		
Variance	244617483		
Skewness	0.90913		
Kurtosis	0.77211		
Median	20658.4		
Interquartile Range	23487.3		

Tail Value-at-Risk Estimates			
Percentile	Pr > VaR	Value at Risk (VaR)	Tail Value at Risk
90	0.1	44943.4	54976.4
95	0.05	52019.3	61880.9
97.5	0.025	59023.3	68705.4
99	0.01	68500.6	77100.1
99.5	0.005	74608.9	82843.7

The “Summary Statistics” table displays the size of the sample, the range of the total loss values in the sample, and various moments of the sample. It also displays median and interquartile range, the robust statistics of location and scale, respectively. The “Tail Value-at-Risk Estimates” table displays the VaR and TVaR estimates for the percentile levels in the TVARPTS= option in the OUTSUM statement. These result tables together provide you a comprehensive statistical picture of the distribution of the total loss. You can especially use the VaR and TVaR estimates to decide the capital requirements for your enterprise. But, you can take this analysis one step further.

The preceding estimates are for a specific set of marginal samples. However, the perturbation analysis that PROC CCDM conducts to capture the uncertainty in the parameter estimates of the severity and frequency models results in multiple samples for each marginal. Similar to how PROC CCDM uses those multiple perturbed samples to estimate the location and dispersion (variability) of various summary statistics and percentiles of a marginal’s CDM, you can use different combinations of the perturbed samples of all the marginals to generate multiple samples of the total loss, compute the risk measures for each such total loss sample, and summarize those estimates to estimate the location and dispersion of all risk measures.

Each perturbed sample is identified by a draw identifier that PROC CCDM writes in the variable `_DRAWID_` when you use the NPerturb= option in the PROC CCDM statement and the PERTURBOUT option in the OUTPUT statement. PROC ECM enables you to specify a value of the draw identifier to use for each marginal by using the DRAWID= option in the MARGINAL statement. The following programming statements use this feature to define a SAS macro that generates multiple combinations of the perturbed samples of both the marginals of this example and invokes PROC ECM for each combination:

```

/*** Macro to generate multiple ECM samples by using multiple perturbed CDM samples,
    and to estimate mean, standard deviation, median, and interquartile range
    of various risk measures of the total loss. ***/
%macro EstimateECMvariability(maxDrawId1=10, maxDrawId2=10);

```



```

%do draw1=0 %to &maxDrawId1;
  %do draw2=0 %to &maxDrawId2;
    %put draw1=&draw1 draw2=&draw2;
    proc ecm data=mylib.CbRbProbs noprint;
      marginal CB: data=mylib.aggregateLossCB samplevar=agglossCB drawId=&draw1;
      marginal RB: data=mylib.aggregateLossRB samplevar=agglossRB drawId=&draw2;
      outsum out=mylib.tlSumm mean stddev skew kurtosis qrange=iqr
        pctlpts=50, 95, 97.5, 99.5 pctlpre=VaR
        tvarpts=95, 97.5, 99.5 pctlpre=TVaR;
    run;
    data tlSumm;
      set mylib.tlSumm;
    run;
    %if &draw1=0 and &draw2=0 %then %do;
      data allecmstats;
        set tlSumm;
      run;
    %end;
    %else %do;
      proc datasets library=work nolist;
        append base=allecmstats data=tlSumm;
      run;
    %end;
  %end;
%end;

proc means data=allecmstats(rename=(VaR50=Median)) n mean std median qrange;
  var Mean Median VaR95 TVaR95 VaR97_5 TVaR97_5 VaR99_5 TVaR99_5;
quit;
%mend;

/*** Compute ECM variability estimates ***
%EstimateECMvariability(maxDrawId1=9, maxDrawId2=9);

```

The preceding code accumulates the summary statistics and risk measures (VaR and TVaR) that each PROC ECM step computes into the data table `Work.AllEcmStats`. The PROC MEANS step then summarizes that data table to estimate the location and dispersion of each summary statistic and risk measure. The final results that the `%EstimateECMvariability` macro prepares are shown in [Output 18.1.11](#).

Output 18.1.11 Location and Dispersion Estimates of Various ECM Summary Statistics and Risk Measures

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Median	Quartile Range
MEAN	Mean	100	22624.91	508.8904821	22690.38	623.0886502
Median	50th Percentile	100	20202.09	448.1312869	20222.62	577.8018542
VaR95	95th Percentile	100	51471.24	1438.64	51693.52	1949.54
TVaR95	Tail Value-at-Risk (TVaR): 95th Percentile	100	60980.61	1670.27	61260.92	2307.31
VaR97_5	97.5th Percentile	100	58242.96	1625.26	58620.67	2290.03
TVaR97_5	Tail Value-at-Risk (TVaR): 97.5th Percentile	100	67557.64	1853.42	68042.35	2543.53
VaR99_5	99.5th Percentile	100	73266.99	1956.93	73606.28	2481.46
TVaR99_5	Tail Value-at-Risk (TVaR): 99.5th Percentile	100	81107.88	2309.65	81642.40	3107.46

You have now completed the process of developing an economic capital model. You should be able to use

this template to analyze multiple dependent lines of business or risk categories of your enterprise and get a statistically sound estimate of the economic capital model to decide the minimum capital that you need to set aside for covering the worst-case losses across the parts of the enterprise that you include in the ECM.

References

- Klugman, S. A., Panjer, H. H., and Willmot, G. E. (1998). *Loss Models: From Data to Decisions*. New York: John Wiley & Sons.
- Sklar, A. (1959). “Fonctions de répartition à n dimensions et leurs marges [Distribution functions with n dimensions and their margins].” *Publications de l’Institut de Statistique de L’Université de Paris* 8:229–231.

Chapter 19

FRONTIER Procedure

Contents

Overview: FRONTIER Procedure	1146
PROC FRONTIER Features	1146
Using CAS Sessions and CAS Engine Librefs	1147
Getting Started: FRONTIER Procedure	1148
Syntax: FRONTIER Procedure	1154
Functional Summary	1154
PROC FRONTIER Statement	1155
BOUNDS Statement	1157
BY Statement	1158
CLASS Statement	1158
DISPLAY Statement	1159
DISPLAYOUT Statement	1160
FREQ Statement	1161
INIT Statement	1161
MODEL Statement	1161
OUTPUT Statement	1162
RESTRICT Statement	1163
TEST Statement	1165
WEIGHT Statement	1166
Details: FRONTIER Procedure	1166
OUTPUT OUT= Data Table	1169
Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT State- ments	1171
Tests on Parameters	1174
ODS Table Names	1175
Examples: The FRONTIER Procedure	1176
Example 19.1: Stochastic Frontier Production Model: Truncated-Normal	1176
Example 19.2: Stochastic Frontier Cost Models	1178
References	1182

Overview: FRONTIER Procedure

The FRONTIER procedure analyzes stochastic frontier production or cost models. These are linear models with an error term that includes two components: a symmetrically distributed component that allows for random shocks of the production or cost, and a systematic positive component that adjusts for technical or cost inefficiency. The inefficiency component of the error term can be from a half-normal, exponential, or truncated-normal distribution.

Stochastic frontier production models were first introduced by Aigner, Lovell, and Schmidt (1977) and Meeusen and van den Broeck (1977). Kumbhakar and Lovell (2000) provide a good introduction to stochastic frontier analysis. Sickles and Zelenyuk (2019) provide detailed information about this topic.

For a detailed introduction to the stochastic frontier models that PROC FRONTIER supports, see the section “[Details: FRONTIER Procedure](#)” on page 1166.

PROC FRONTIER Features

The FRONTIER procedure can fit stochastic frontier production or cost models. On the basis of the parameterization of the inefficiency term, it supports three stochastic frontier models:

- the half-normal model, where the inefficiency term is half-normally distributed
- the exponential model, where the inefficiency term is exponentially distributed
- the truncated-normal model, where the inefficiency term has a truncated-normal distribution with a truncation point at μ

The inefficiency term is assumed to be independently distributed among the observations.

The following list summarizes some of the basic features of PROC FRONTIER:

- uses model-building syntax that includes **CLASS** and effect-based **MODEL** statements that are familiar from other SAS Econometrics analytic procedures
- fits the stochastic frontier models by using maximum likelihood techniques
- provides a **TEST** statement for performing linear tests of hypotheses
- provides **RESTRICT** and **BOUNDS** statements for imposing linear parameter constraints
- provides an **OUTPUT** statement for producing a data table that contains predictions, residuals, and technical efficiency terms
- uses the **WEIGHT** statement for weighted analysis
- uses the **FREQ** statement for grouped analysis

Because PROC FRONTIER runs on SAS Cloud Analytic Services (CAS), it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode
- exploits all the available cores and concurrent threads. For information about how PROC FRONTIER uses threads, see the section “[Multithreading](#)” on page 108 in Chapter 4, “[Shared Concepts](#).”

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: FRONTIER Procedure

This section illustrates how to estimate stochastic frontier production models. The following statements fit a half-normal model:

```
proc frontier data=mylib.mydata;
    model y = x1 x2 / type=half;
run;
```

The output variable *y* and the input variables *x1* and *x2* are contained in the data table *mylib.mydata*. These statements assume that your CAS engine libref is named *mylib*, but you can substitute any appropriately defined CAS engine libref. The relationship between input and output is specified in the MODEL statement. The distribution of the inefficiency term is specified using the TYPE= option in the MODEL statement. In this example, it is the half-normal distribution. TYPE=EXPONENTIAL specifies an exponential distribution, and TYPE=TRUNCATED specifies a truncated-normal distribution. By default, PROC FRONTIER fits a production model. You can specify the COST option in the MODEL statement to fit a stochastic frontier cost model.

The following example fits the stochastic frontier production model to 1957 state data for the US transportation equipment manufacturing industry. The data were originally published in Zellner and Revankar (1969). The data include observations on aggregate value added (*valueadded*), aggregate capital service flow (*capital*), aggregate hours worked (*labor*), and number of establishments (*nfirm*) for 25 US states. A Cobb-Douglas production function is estimated for the output variable, log of value added (*LNV*), using the input variables log of capital (*LNK*) and log of labor (*LNL*).

The following statements create the data set:

```
title1 'Getting Started Example: Stochastic Frontier Production Model';

data transpEqp;
    input state$ valueadded capital labor nfirm;
datalines;
Alabama      126.14800262  3.8039999008  31.551000595  68
California   3201.486084   185.44599915  452.84399414  1372
Connecticut  690.66998291   39.712001801  124.0739975   154
Florida      56.296001434   6.5469999313  19.180999756  292
Georgia      304.53100586   11.529999733  45.534000397  71
Illinois     723.02801514   58.986999512  88.39099884   275
Indiana      992.16900635   112.88400269  148.52999878  260
Iowa         35.796001434   2.6979999542  8.0170001984  75
Kansas       494.51501465  10.359999657  86.189002991  76
Kentucky     124.94799805   5.2129998207  12             31
Louisiana    73.32800293    3.7630000114  15.899999619  115
Maine        29.466999054   1.9670000076  6.4699997902  81
Maryland     415.26199341   17.545999527  69.342002869  129
Massachusetts 241.52999878   15.347000122  39.416000366  172
Michigan     4079.5539551  435.10501099  490.38400269  568
Missouri     652.08502197   32.840000153  84.831001282  125
NewJersey    667.11297607   33.291999817  83.032997131  247
NewYork      940.42999268   72.973999023  190.09399414  461
Ohio         1611.8990479  157.97799683  259.91598511  363
Pennsylvania 617.57897949   34.324001312  98.152000427  233
```

```

Texas      527.4130249  22.736000061 109.72799683 308
Virginia   174.39399719  7.1729998589 31.301000595 85
Washington 636.94799805 30.806999207 87.962997437 179
WestVirginia 22.700000763 1.5429999828 4.0630002022 15
Wisconsin  349.71099854 22.000999451 52.818000793 142
;

data transpEqp;
  set transpEqp;
  LNV = log(valueadded);
  LNK = log(capital);
  LNL = log(labor);

data mylib.transpEqp;
  set transpEqp;
run;

```

The following statements estimate a stochastic frontier half-normal production model that uses the transportation equipment manufacturing industry data:

```

/*-- Stochastic Frontier Production Model: Half-Normal --*/
proc frontier data=mylib.transpEqp;
  model LNV = LNK LNL / type=half;
run;

```

The “Observation Information” and “Summary Statistics of Dependent Variable” tables shown in [Figure 19.1](#) contain some details about the observations in the data set and the dependent variable.

Figure 19.1 Information about Observations and the Dependent Variable

Getting Started Example: Stochastic Frontier Production Model

The FRONTIER Procedure

Observation Information				
Number of Observations				25
Number of Missing Observations				0

Summary Statistics of Dependent Variable				
Standard				
Variable	Mean	Error	Minimum	Maximum
LNV	5.812092	1.375304	3.122365	8.313743

The “Model Fit Summary” table shown in [Figure 19.2](#) lists several details about the model. By default, PROC FRONTIER uses the Newton-Raphson optimization technique, and the estimated covariance matrix is calculated as the inverse of the negative Hessian. For the available optimization and estimation techniques, see the section “[PROC FRONTIER Statement](#)” on page 1155. The maximum log-likelihood value is shown, in addition to two information measures—Akaike’s information criterion (AIC) and Schwarz’s Bayesian information criterion (SBC)—that can be used to compare competing production models. Smaller values of these criteria indicate better models.

Figure 19.2 Estimation Summary Table for Half-Normal Production Model

Model Fit Summary	
Dependent Variable	LNV
Data Set	TRANSPEQP
Model	Production
Inefficiency Term Distribution	Half-normal
Log Likelihood	2.469522
Maximum Absolute Gradient	3.942E-6
Number of Iterations	11
Optimization Method	Newton-Raphson
AIC	5.060956
SBC	11.15533
Covariance Estimation	Hessian

Figure 19.3 shows the parameter estimates of the model and their standard errors. Both input variables are significant predictors of the log of value added. `_Sigma_v` is the estimate of the standard deviation of the idiosyncratic component of the error term that is specific to the firm and that could enter the model with either sign. `_Sigma_u` is the estimate of the standard deviation of the inefficiency component. When the data are in log terms, the inefficiency component becomes a measure of the percentage by which a particular observation fails to achieve an ideal production rate.

Figure 19.3 Parameter Estimates of Half-Normal Production Model

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	2.081134	0.281648	7.39	<.0001
LNK	1	0.258548	0.098766	2.62	0.0089
LNL	1	0.780245	0.119945	6.51	<.0001
<code>_Sigma_v</code>	1	0.175169	0.054265	3.23	0.0012
<code>_Sigma_u</code>	1	0.221507	0.123706	1.79	0.0734

Figure 19.4 shows the estimates of two important variance statistics that are helpful for model diagnosis. `Sigma2` is the estimate of the total error variance. It is calculated as the sum of the idiosyncratic error variance and the variance of the technical inefficiency term. `Gamma` is the estimate of how much the technical inefficiency contributes to the total variation in the model. It is calculated as the ratio of the variance of the technical inefficiency term to the total error variance. In this example, the technical inefficiency explains most of the variation in the model.

Figure 19.4 Estimates of Important Variance Statistics for Half-Normal Production Model

Variance Statistics		
Parameter	Estimate	Standard Error
<code>Sigma2</code>	0.079750	0.042699
<code>Gamma</code>	0.615244	0.385748

The following statements estimate a stochastic frontier exponential production model that uses the same data:


```

/*-- Stochastic Frontier Production Model: Exponential --*/
proc frontier data=mylib.transpEqp;
  model LNV = LNK LNL;
  output out=mylib.out_te_exp  te1=TE1_exp te2=TE2_exp copyvars=( state );
run;

```

The default model is the exponential model if you do not specify the TYPE= option in the MODEL statement.

Figure 19.5 shows some of the results from this production model.

Figure 19.5 Estimation Summary and Parameter Estimates for Exponential Production Model

Getting Started Example: Stochastic Frontier Production Model

The FRONTIER Procedure

Model Fit Summary					
Dependent Variable	LNV				
Data Set	TRANSP EQP				
Model	Production				
Inefficiency Term Distribution	Exponential				
Log Likelihood	2.86049				
Maximum Absolute Gradient	4.215E-6				
Number of Iterations	11				
Optimization Method	Newton-Raphson				
AIC	4.279021				
SBC	10.3734				
Covariance Estimation	Hessian				

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	2.069242	0.235632	8.78	<.0001
LNK	1	0.262486	0.092001	2.85	0.0043
LNL	1	0.770380	0.110964	6.94	<.0001
_Sigma_v	1	0.171392	0.038449	4.46	<.0001
_Sigma_u	1	0.135169	0.062685	2.16	0.0311

Variance Statistics		
Parameter	Estimate	Standard Error
Sigma2	0.047646	0.015793
Gamma	0.383467	0.285232

You can obtain estimates of the efficiency terms by using the OUTPUT statement. PROC FRONTIER offers two types of technical efficiency terms that are proposed by Battese and Coelli (1988) and Jondrow et al. (1982). The OUTPUT statement in the previous and following statements specify the two technical efficiency terms TE1 and TE2. These terms for each observation are written in the data set that you specify in the OUT= option. In these examples, the technical efficiency terms are calculated for each state and included in mylib.out_te_exp for the exponential model and mylib.out_te_half for the half-normal model.

```

/*-- Obtaining Technical Efficiency Terms for Half-Normal Model --*/
proc frontier data=mylib.transpEqp;
    model LNV = LNK LNL / type=half;
    output out=mylib.out_te_half te1=TE1_half te2=TE2_half copyvars=( state );
run;

```

The following statements merge the two output data sets and sort the new data set with respect to the state variable:

```

/*-- Merging the Two Output Data Sets --*/
data mylib.out_te;
    merge mylib.out_te_half mylib.out_te_exp;
    by state;
run;

proc sort data=mylib.out_te out=out_te_sorted;
    by state;
run;

proc print data = out_te_sorted;
    var State TE1_half TE1_exp TE2_half TE2_exp;
    title 'TEs from Half-Normal and Exponential Models';
run;

```

These statistics for each state are shown in [Figure 19.6](#).

Figure 19.6 Technical Efficiency for Each State
TEs from Half-Normal and Exponential Models

Obs	state	TE1_half	TE1_exp	TE2_half	TE2_exp
1	Alabama	0.8231755124	0.8690845704	0.8178030737	0.8642193711
2	Californ	0.8692654721	0.9102507468	0.8651869818	0.9073595422
3	Connecti	0.8318406668	0.8783346976	0.82667106	0.8739011963
4	Florida	0.6016341141	0.5623070763	0.5959902277	0.5541443131
5	Georgia	0.904050962	0.9329042531	0.901244131	0.9310801261
6	Illinois	0.8891712693	0.9226121891	0.8857974959	0.9203129631
7	Indiana	0.8150898568	0.8620239532	0.8095457608	0.8568378457
8	Iowa	0.7853521623	0.824843113	0.7793217245	0.8181327187
9	Kansas	0.9066430761	0.9354333085	0.9039373908	0.9337222102
10	Kentucky	0.9464481304	0.960059195	0.9452846459	0.9593259036
11	Louisian	0.8214223188	0.8651803151	0.816011214	0.8601366506
12	Maine	0.8061243279	0.8474149487	0.8004097788	0.8415937609
13	Maryland	0.8772239419	0.914381889	0.8734183876	0.9116902323
14	Massachu	0.8596337404	0.899833078	0.855242772	0.8964352583
15	Michigan	0.8582037513	0.9013389234	0.85376816	0.8980144395
16	Missouri	0.9050045164	0.9337916042	0.9022348315	0.9320073289
17	NewJerse	0.9111356273	0.9378481876	0.908606331	0.9362433382
18	NewYork	0.7636376949	0.8078160421	0.7573893975	0.8005197149
19	Ohio	0.8010178144	0.849549022	0.795215466	0.8438179328
20	Pennsylv	0.8648740981	0.9052355469	0.8606506142	0.9021007787
21	Texas	0.8217006752	0.869420456	0.816295656	0.8645707216
22	Virginia	0.8732988787	0.9106452426	0.8693571298	0.9077731451
23	Washingt	0.8984320007	0.9294176476	0.8954080069	0.9274350587
24	WestVirg	0.8602609125	0.8972449432	0.8558896629	0.8937211392
25	Wisconsi	0.8727364593	0.9103636198	0.8687754441	0.9074778831

The values of each technical efficiency term from the half-normal model are close to the corresponding values from the exponential model.

Syntax: FRONTIER Procedure

The following statements are available in the FRONTIER procedure. Items in angle brackets (< >) are optional.

```

PROC FRONTIER < options > ;
  BOUNDS bound1 < , bound2 ... > ;
  BY variables ;
  CLASS variable < (options) > ... < variable < (options) > > < / global-options > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  FREQ freq-variable ;
  INIT initialization1 < , initialization2 ... > ;
  MODEL dependent-variable = regressors < / options > ;
  OUTPUT < output-options > ;
  RESTRICT restriction1 < , restriction2 ... > ;
  TEST equation1 < , equation2 ... > < / test-options > ;
  WEIGHT variable < / option > ;

```

Functional Summary

Table 19.1 summarizes the statements and options available in the FRONTIER procedure.

Table 19.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC FRONTIER	DATA=
Writes estimates to an output data table	OUTPUT	OUT=
Declaring the Role of Variables		
Specifies BY-group processing	BY	
Specifies the classification variables	CLASS	
Specifies an optional frequency variable	FREQ	
Specifies an optional weight variable	WEIGHT	NONNORMALIZE
Printing Control Options		
Prints the correlation matrix of the estimates	PROC FRONTIER	CORRB
Prints the covariance matrix of the estimates	PROC FRONTIER	COVB
Suppresses the normal printed output	PROC FRONTIER	NOPRINT
Requests all printing options	PROC FRONTIER	PRINTALL
Prints the names used internally for the parameters	PROC FRONTIER	PRINTINTERNALNAMES
Options to Control the Optimization Process		
Selects the iterative minimization method to use	PROC FRONTIER	METHOD=
Specifies maximum number of iterations allowed	PROC FRONTIER	MAXITER=

Table 19.1 *continued*

Description	Statement	Option
Specifies maximum number of function calls	PROC FRONTIER	MAXFUNC=
Specifies the upper limit of CPU time in seconds	PROC FRONTIER	MAXTIME=
Sets boundary restrictions on parameters	BOUNDS	
Sets initial values for parameters	INIT	
Sets linear restrictions on parameters	RESTRICT	
Model Estimation Options		
Specifies the type of covariance matrix	PROC FRONTIER	COVEST=
Suppresses the intercept parameter	MODEL	NOINT
Specifies a stochastic frontier production model	MODEL	PRODUCTION
Specifies a stochastic frontier cost model	MODEL	COST
Specifies the distribution for the inefficiency term	MODEL	TYPE=
Output Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	
Outputs SAS variables to the output data table	OUTPUT	COPYVAR=
Outputs expected value of response variable	OUTPUT	PRED=
Outputs the residuals	OUTPUT	RESID=
Outputs the technical efficiency measures	OUTPUT	TE1= and/or TE2=
Outputs estimates of $\mathbf{X}\beta = \mathbf{x}_i' \boldsymbol{\beta}$	OUTPUT	XBETA=

PROC FRONTIER Statement

PROC FRONTIER <options> ;

You can specify the following *options* in the PROC FRONTIER statement.

Input Data Table Options

DATA=*libref.data-table*

names the input data table for PROC FRONTIER to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 1147.

data-table specifies the name of the input data table.

Printing Options

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

ITPRINT

prints the initial parameter estimates, convergence criteria, and all constraints of the optimization. At each iteration, the objective function value, step size, maximum gradient, and slope of search direction are also printed.

NOPRINT

suppresses all printed output.

PRINTALL

requests all printing options.

PRINTINTERNALNAMES

prints the internal names that are assigned to parameters.

Estimation Control Options

COVEST=HESSIAN | OP | QML

specifies the type of covariance matrix for the parameter estimates. You can specify the following values:

HESSIAN	specifies the covariance from the Hessian matrix.
OP	specifies the covariance from the outer-product matrix.
QML	specifies the covariance from the outer-product and Hessian matrices.

By default, COVEST=HESSIAN.

Optimization Control Options

PROC FRONTIER uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. You can specify the following *options*:

MAXFUNC=*i*

MAXFU=*i*

specifies the maximum number of function calls in the optimization process. By default, MAXFUNC=1000. The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually performed can exceed the number of calls that you specify in this option.

MAXITER=*i*

MAXIT=*i*

specifies the maximum number of iterations in the optimization process. By default, MAXITER=200.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. The default value is the largest floating-point double representation of your computer. The time that you specify in this option is checked only once at the end of each iteration. Therefore, the actual run time can be much longer than *r*. The actual run time includes the remaining time needed to finish the iteration and the time that is needed to generate the output of the results.

METHOD=*value*

specifies the iterative minimization method to use. You can specify the following *values*:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double-dogleg method.
NEWRAP	specifies the Newton-Raphson method.
NONE	specifies that no optimization be performed beyond using the ordinary least squares method to compute the parameter estimates.
NRRIDG	specifies the Newton-Raphson ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

By default, METHOD=NEWRAP.

BOUNDS Statement

BOUNDS *bound1* < , *bound2* ... > ;

The BOUNDS statement imposes simple boundary constraints on the parameter estimates. You can specify any number of BOUNDS statements.

Each *bound* is composed of parameter names, constants, and inequality operators as follows:

item operator item < *operator item* < *operator item* ... > >

Each *item* is a constant, a parameter name, or a list of parameter names. Each *operator* is <, >, <=, or >=. Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the BOUNDS statement. You can specify the PRINTINTERNALNAMES option in the PROC FRONTIER statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column that shows the internal name of each parameter is added to the “Parameter Estimates” table.

For more information about how parameters are named, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 1171.

You can use both the BOUNDS statement and the RESTRICT statement to impose boundary constraints. However, the BOUNDS statement provides a simpler syntax for specifying these kinds of constraints. For more information, see the section “[RESTRICT Statement](#)” on page 1163.

The following BOUNDS statement illustrates the use of parameter lists to specify boundary constraints. It constrains the estimates of the constant term, the intercept, to be negative; the parameters for the input price P1 to be between 0 and 1; and the technical inefficiency term to be greater than 0.5.

```
bounds intercept < 0,
          0 < P1 < 1,
          _sigma_u > 0.5;
```

Note that operators to select or construct effects, such as “-”, “*”, or “|”, are not allowed in the BOUND statement.

BY Statement

BY *variables* ;

A BY statement can be used with PROC FRONTIER to obtain separate analyses of observations in groups that are defined by the BY *variables*.

CLASS Statement

CLASS *variable* < (*options*) > . . . < *variable* < (*options*) > > < / *global-options* > ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. [Table 19.2](#) summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “[CLASS Statement](#)” on page 56 in Chapter 4, “[Shared Concepts](#).”

Table 19.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable

By default, REF=LAST.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 1175. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetted for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the output table name.

key specifies *key* as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 1175. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

FREQ Statement

FREQ *freq-variable* ;

The FREQ statement identifies a variable (*freq-variable*) that contains the frequency of occurrence of each observation. PROC FRONTIER treats each observation as if it appears n times, where n is the value of *freq-variable* for the observation. If the value for the observation is not an integer, it is truncated to an integer. If the value is less than 1 or missing, the observation is not used in fitting the model. When the FREQ statement is omitted, each observation is assigned a frequency of 1.

INIT Statement

INIT *initialization1* < , *initialization2* ... > ;

The INIT statement sets initial values for parameters in the optimization.

Each *initialization* is written as a parameter or parameter list, followed by an optional equal sign (=), followed by a number:

parameter < = > *number*

Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC FRONTIER statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column that shows the internal name of each parameter is added to the “Parameter Estimates” table.

For more information about how parameters are named, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 1171.

MODEL Statement

MODEL *dependent-variable* = *regressors* < / *options* > ;

The MODEL statement specifies the dependent variable and independent regressor variables for a stochastic frontier model.

Only one MODEL statement can be specified. You can specify the following *options* after a slash (/):

COST | PRODUCTION

specifies that the estimated model be a cost or production function.

If neither COST nor PRODUCTION is specified, a production function is estimated by default.

NOINT

suppresses the intercept parameter.

TYPE=EXPONENTIAL | HALF | TRUNCATED**DIST=EXPONENTIAL | HALF | TRUNCATED**

specifies the type of distribution for the inefficiency term. You can specify the following types:

EXPONENTIAL specifies an exponential distribution for the inefficiency term.

HALF specifies a half-normal distribution for the inefficiency term.

TRUNCATED specifies a truncated-normal distribution for the inefficiency term.

By default, TYPE=EXPONENTIAL.

OUTPUT Statement

OUTPUT OUT=*libref.data-table* < *output-options* > ;

The OUTPUT statement creates a new data table that includes variables created by the *output-options*. These variables include the estimates of $\mathbf{x}_i'\boldsymbol{\beta}$, the residual, the predicted value of the response variable, and technical efficiency measures.

When the response values are missing for an observation, all output estimates except the residual for that observation are still computed as long as none of the explanatory variables are missing. This enables you to compute these statistics for prediction.

You can specify only one OUTPUT statement.

You must specify the OUT= option:

OUT=*libref.data-table*

names the output data table for PROC FRONTIER to use. You must specify this option before any other options. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 1147.

data-table specifies the name of the output data table.

You can specify one or more of the following *output-options*:

COPYVAR=(*SAS-variable-names*)

COPYVARS=(*SAS-variable-names*)

adds SAS variables to the output data table.

PRED=*name*

MEAN=*name*

names the variable to contain the predicted value of the response variable.

RESIDUAL=*name*

RESID=*name*

names the variable to contain estimates of residuals of the response variable.

TE1=*name*

names the variable to contain estimates of technical efficiency as suggested by Battese and Coelli (1988).

TE2=*name*

names the variable to contain estimates of technical efficiency as suggested by Jondrow et al. (1982).

XBETA=*name*

names the variable to contain estimates of $\mathbf{x}_i' \boldsymbol{\beta}$.

RESTRICT Statement

RESTRICT *restriction1* <, *restriction2* ... > ;

The RESTRICT statement imposes linear restrictions on the parameter estimates. You can specify any number of RESTRICT statements.

Each *restriction* is written either as a single linear equation or as a comma-separated list of two or more linear equations. A restriction equation consists of an *expression*, followed by an equality *operator* (=) or an inequality *operator* (<, >, <=, >=), followed by a second *expression*:

expression operator expression

The *operator* can be =, <, >, <=, or >=.

RESTRICT Statement Expressions

A restriction expression is composed of parameter names, constants, and the operators times (*), plus (+), and minus (−). Each restriction expression must be a linear function of the parameters in the model. In addition, no grouping symbols (such as parentheses) are allowed, and the constant factor in any product can appear only on the left-hand side of the times (*) operator.

The following statements illustrate the use of the RESTRICT statement:

```
proc frontier data=mylib.dataset;
  model y = x1-x10 / production type=exponential;
  restrict x1*x2 <= x2 + x3;
run;
```

It is important to keep in mind that the parameters that are associated with the variables, not the variables themselves, are restricted. Thus, in the preceding RESTRICT statement, the variable names “x1”, “x2”, and “x3” refer to the parameters that are associated with the variables x1, x2, and x3, respectively, not to the variables themselves.

Parameter names are shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the RESTRICT statement. You can specify the PRINTINTERNALNAMES option in the PROC FRONTIER statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column that shows the internal name of each parameter is added to the “Parameter Estimates” table.

For more information about how parameters are named, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 1171.

All restriction equations in all RESTRICT statements are applied jointly.

RESTRICT Statement Examples

Examples of valid RESTRICT statements include the following:

```
restrict x1=0.1;
restrict a+b=1;
restrict a-b=0, b+c=1.5;
restrict 2*f=g+h, intercept+f=0;
```

Examples of invalid RESTRICT statements include the following:

```
restrict x1^2=4;
restrict x1*x3=4;
restrict x1/x3=2;
restrict sin(a)=0;
restrict a*0.5=1;
restrict 2*(f+h)=1;
```

In the first four examples of invalid RESTRICT statements, the equation is nonlinear. The fifth example is invalid because the constant factor (0.5) cannot appear on the right-hand side of the times (*) operator. The last example is invalid because grouping symbols are prohibited.

The set of restrictions should be consistent and not redundant. For example, you cannot specify the following statement because the three restrictions are not consistent:

```
restrict f-g=0,
       f-intercept=0,
       g-intercept=1;
```

Lagrange multipliers are reported in the “Parameter Estimates” table for all the active linear constraints. They are identified by the names Restrict1, Restrict2, and so on. The test statistics and *p*-values for these restrictions are computed using the Lagrange multiplier test that is described in the section “[Tests on Parameters](#)” on page 1174. Nonactive (nonbinding) restrictions have no effect on the estimation results and are not noted in the output.

TEST Statement

`<label:> TEST <'string'> equation1 <, equation2... > /<test-options> ;`

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the regression parameters that are specified in the preceding MODEL statement.

Each test is written either as a single linear *equation* or as a comma-separated list of two or more linear *equations*. A test *equation* specifies a linear hypothesis to be tested and consists of an *expression*, followed by the equality operator (=), followed by a second expression:

expression = *expression*

The rules for valid test expressions are the same as those for restriction expressions. For more information, see the section “[RESTRICT Statement Expressions](#)” on page 1163.

All hypotheses in one TEST statement are tested jointly.

You can specify the following *test-options* after a slash (/):

ALL

performs the Wald, Lagrange multiplier, and likelihood ratio tests.

LM

performs the Lagrange multiplier test.

LR

performs the likelihood ratio test.

WALD

performs the Wald test.

By default, the Wald test is performed. Each type of test is described in the section “[Tests on Parameters](#)” on page 1174.

You can add a label (which is printed in the output) to a TEST statement in two ways: by adding an unquoted *label* followed by a colon before the TEST keyword, or by adding a quoted *string* after the TEST keyword. The unquoted *label* cannot contain any spaces. If you include both an unquoted *label* and a quoted *string*, PROC FRONTIER uses only the unquoted *label*. If you specify neither an unquoted *label* nor a quoted *string*, PROC FRONTIER automatically labels the tests.

The following example illustrates the use of the TEST statement:

```
proc frontier;
  model y = x1 x2 x3;
  test x1 = 0, 1.5 * x2 + 2 * x3 = 0;
  test_int: test intercept = 0, x3 = 0.75;
run;
```

In the example, two separate tests are performed. The first test investigates the joint hypothesis that

$$\beta_1 = 0$$

and

$$1.5\beta_2 + 2\beta_3 = 0$$

The second test is labeled “test_int” and investigates the joint hypothesis that

$$\beta_{\text{Intercept}} = 0$$

and

$$\beta_3 = 0.75$$

WEIGHT Statement

WEIGHT *variable* </ *option* > ;

The WEIGHT statement specifies a variable to supply weighting values to use for each observation in estimating parameters. The log likelihood for each observation is multiplied by the corresponding weight variable value.

If the weight of an observation is nonpositive, that observation is not used in the estimation.

You can specify the following *option* in the WEIGHT statement after a slash (/):

NONNORMALIZE

does not normalize the weights. (By default, the weights are normalized so that they add up to the actual sample size. The weights w_i are normalized by multiplying them by $\frac{n}{\sum_{i=1}^n w_i}$, where n is the sample size.) If the weights must be used as they are, then you must specify this option.

Details: FRONTIER Procedure

Stochastic frontier production models were first developed by Aigner, Lovell, and Schmidt (1977) and Meeusen and van den Broeck (1977). Specification of these models allows for random shocks of the production or cost but also includes a term for technical or cost inefficiency. Assuming that the production function takes a log-linear Cobb-Douglas form, the stochastic frontier production model can be written as

$$\ln(y_i) = \beta_0 + \sum_n \beta_n \ln(x_{ni}) + \epsilon_i$$

where $\epsilon_i = v_i - u_i$. The v_i term represents the stochastic error component, and u_i is the nonnegative, technical inefficiency error component. The v_i error component is assumed to be distributed iid normal and independently from u_i . Given that $u_i > 0$, the error term, ϵ_i , is negatively skewed and represents technical inefficiency.

For the stochastic frontier cost model, $\epsilon_i = v_i + u_i$. The v_i term represents the stochastic error component, and u_i is the nonnegative, cost inefficiency error component. Given that $u_i > 0$, the error term, ϵ_i , is positively skewed and represents cost inefficiency. PROC FRONTIER models the u_i error component as a half-normal, exponential, or truncated-normal distribution.

The Normal–Half Normal Model

In case of the normal–half normal model, v_i is iid $N(0, \sigma_v^2)$ and u_i is iid $N^+(0, \sigma_u^2)$, with v_i and u_i independent of each other. Given the independence of the error terms, the joint density of v and u can be written as

$$f(u, v) = \frac{2}{2\pi\sigma_u\sigma_v} \exp \left\{ -\frac{u^2}{2\sigma_u^2} - \frac{v^2}{2\sigma_v^2} \right\}$$

Substituting $v = \epsilon + u$ into the preceding equation gives

$$f(u, \epsilon) = \frac{2}{2\pi\sigma_u\sigma_v} \exp \left\{ -\frac{u^2}{2\sigma_u^2} - \frac{(\epsilon + u)^2}{2\sigma_v^2} \right\}$$

Integrating u out to obtain the marginal density function of ϵ results in the form

$$\begin{aligned} f(\epsilon) &= \int_0^\infty f(u, \epsilon) du \\ &= \frac{2}{\sqrt{2\pi}\sigma} \left[1 - \Phi \left(\frac{\epsilon\lambda}{\sigma} \right) \right] \exp \left\{ -\frac{\epsilon^2}{2\sigma^2} \right\} \\ &= \frac{2}{\sigma} \phi \left(\frac{\epsilon}{\sigma} \right) \Phi \left(-\frac{\epsilon\lambda}{\sigma} \right) \end{aligned}$$

where $\lambda = \sigma_u/\sigma_v$ and $\sigma = \sqrt{\sigma_u^2 + \sigma_v^2}$.

In the case of a stochastic frontier cost model, $v = \epsilon - u$ and

$$f(\epsilon) = \frac{2}{\sigma} \phi \left(\frac{\epsilon}{\sigma} \right) \Phi \left(\frac{\epsilon\lambda}{\sigma} \right)$$

The log-likelihood function for the production model with N producers is written as

$$\ln L = \text{constant} - N \ln \sigma + \sum_i \ln \Phi \left(-\frac{\epsilon_i \lambda}{\sigma} \right) - \frac{1}{2\sigma^2} \sum_i \epsilon_i^2$$

The Normal-Exponential Model

Under the normal-exponential model, v_i is iid $N(0, \sigma_v^2)$ and u_i is iid exponential with scale parameter σ_u . Given the independence of the error term components v_i and u_i , the joint density of v and u can be written as

$$f(u, v) = \frac{1}{\sqrt{2\pi}\sigma_u\sigma_v} \exp \left\{ -\frac{u}{\sigma_u} - \frac{v^2}{2\sigma_v^2} \right\}$$

The marginal density function of ϵ for the production function is

$$\begin{aligned} f(\epsilon) &= \int_0^\infty f(u, \epsilon) du \\ &= \left(\frac{1}{\sigma_u} \right) \Phi \left(-\frac{\epsilon}{\sigma_v} - \frac{\sigma_v}{\sigma_u} \right) \exp \left\{ \frac{\epsilon}{\sigma_u} + \frac{\sigma_v^2}{2\sigma_u^2} \right\} \end{aligned}$$

and the marginal density function for the cost function is equal to

$$f(\epsilon) = \left(\frac{1}{\sigma_u} \right) \Phi \left(\frac{\epsilon}{\sigma_v} - \frac{\sigma_v}{\sigma_u} \right) \exp \left\{ -\frac{\epsilon}{\sigma_u} + \frac{\sigma_v^2}{2\sigma_u^2} \right\}$$

The log-likelihood function for the normal-exponential production model with N producers is

$$\ln L = \text{constant} - N \ln \sigma_u + N \left(\frac{\sigma_v^2}{2\sigma_u^2} \right) + \sum_i \frac{\epsilon_i}{\sigma_u} + \sum_i \ln \Phi \left(-\frac{\epsilon_i}{\sigma_v} - \frac{\sigma_v}{\sigma_u} \right)$$

The Normal–Truncated Normal Model

The normal–truncated normal model is a generalization of the normal–half normal model in which the mean of u_i is allowed to differ from zero. Under the normal–truncated normal model, the error term component v_i is iid $N(0, \sigma_v^2)$ and u_i is iid $N^+(\mu, \sigma_u^2)$. The joint density of v_i and u_i can be written as

$$f(u, v) = \frac{1}{2\pi\sigma_u\sigma_v\Phi(\mu/\sigma_u)} \exp \left\{ -\frac{(u - \mu)^2}{2\sigma_u^2} - \frac{v^2}{2\sigma_v^2} \right\}$$

The marginal density function of ϵ for the production function is

$$\begin{aligned} f(\epsilon) &= \int_0^\infty f(u, \epsilon) du \\ &= \frac{1}{\sqrt{2\pi}\sigma\Phi(\mu/\sigma_u)} \Phi \left(\frac{\mu}{\sigma\lambda} - \frac{\epsilon\lambda}{\sigma} \right) \exp \left\{ -\frac{(\epsilon + \mu)^2}{2\sigma^2} \right\} \\ &= \frac{1}{\sigma} \phi \left(\frac{\epsilon + \mu}{\sigma} \right) \Phi \left(\frac{\mu}{\sigma\lambda} - \frac{\epsilon\lambda}{\sigma} \right) \left[\Phi \left(\frac{\mu}{\sigma_u} \right) \right]^{-1} \end{aligned}$$

and the marginal density function for the cost function is

$$f(\epsilon) = \frac{1}{\sigma} \phi \left(\frac{\epsilon - \mu}{\sigma} \right) \Phi \left(\frac{\mu}{\sigma\lambda} + \frac{\epsilon\lambda}{\sigma} \right) \left[\Phi \left(\frac{\mu}{\sigma_u} \right) \right]^{-1}$$

The log-likelihood function for the normal–truncated normal production model with N producers is

$$\begin{aligned} \ln L &= \text{constant} - N \ln \sigma - N \ln \Phi \left(\frac{\mu}{\sigma_u} \right) + \sum_i \ln \Phi \left(\frac{\mu}{\sigma\lambda} - \frac{\epsilon_i\lambda}{\sigma} \right) \\ &\quad - \frac{1}{2} \sum_i \left(\frac{\epsilon_i + \mu}{\sigma} \right)^2 \end{aligned}$$

For more information about normal–half normal, normal-exponential, and normal-truncated models, see Kumbhakar and Lovell (2000) and Coelli, Prasada Rao, and Battese (1998).

OUTPUT OUT= Data Table

The OUTPUT statement creates a new data table that contains various estimates that you specify. You can request that the output data table contain the estimates of $\mathbf{x}_i'\boldsymbol{\beta}$, the residual, the predicted values, and the two types of technical efficiency statistics. Except for the residual, these statistics are computed for all observations in which the regressors are not missing, even if the response is missing.

Because of potential space limitations on the client workstation, the data table that the OUTPUT statement creates does not automatically contain the variables in the input data table. You can include the input variables in the output data table by specifying the COPYVAR= option.

XBeta, Predicted, and Residual

XBeta is the structural part on the right-hand side of the model. The value of Predicted is the predicted dependent variable value. The value of Residual is defined as

$$\text{Residual} = \text{Observed} - \text{Predicted}$$

Technical Efficiency: TE1, TE2

In general, the stochastic production frontier can be written as

$$y_i = f(x_i; \beta) \exp\{v_i\} \text{TE}_i$$

where y_i denotes producer i 's actual output; $f(\cdot)$ is the deterministic part of the production frontier; $\exp\{v_i\}$ is a producer-specific error term; and TE_i is the technical efficiency coefficient, which can be written as

$$\text{TE}_i = \frac{y_i}{f(x_i; \beta) \exp\{v_i\}}$$

In the case of a Cobb-Douglas production function, $\text{TE}_i = \exp\{-u_i\}$. For more information, see the section “[Details: FRONTIER Procedure](#)” on page 1166.

Cost frontier can be written in general as

$$E_i = c(y_i, w_i; \beta) \exp\{v_i\} / \text{CE}_i$$

where w_i denotes producer i 's input prices; $c(\cdot)$ is the deterministic part of the cost frontier; $\exp\{v_i\}$ is a producer-specific error term; and CE_i is the cost efficiency coefficient, which can be written as

$$\text{CE}_i = \frac{c(x_i, w_i; \beta) \exp\{v_i\}}{E_i}$$

In the case of a Cobb-Douglas cost function, $\text{CE}_i = \exp\{-u_i\}$. For more information, see the section “[Details: FRONTIER Procedure](#)” on page 1166. Hence, both technical and cost efficiency coefficients are the same. The estimates of technical efficiency are provided in the following subsections.

Normal-Half Normal Model

Define $\mu_* = -\epsilon\sigma_u^2/\sigma^2$ and $\sigma_*^2 = \sigma_u^2\sigma_v^2/\sigma^2$. Then, as shown by Jondrow et al. (1982), the conditional density is as follows:

$$f(u|\epsilon) = \frac{f(u, \epsilon)}{f(\epsilon)} = \frac{1}{\sqrt{2\pi}\sigma_*} \exp\left\{-\frac{(u - \mu_*)^2}{2\sigma_*^2}\right\} \Bigg/ \left[1 - \Phi\left(-\frac{\mu_*}{\sigma_*}\right)\right]$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\mu_*, \sigma_*^2)$.

Using this result, it follows that the estimate of technical efficiency (Battese and Coelli 1988) is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \left[\frac{1 - \Phi(\sigma_* - \mu_{*i}/\sigma_*)}{1 - \Phi(-\mu_{*i}/\sigma_*)} \right] \exp \left\{ -\mu_{*i} + \frac{1}{2}\sigma_*^2 \right\}$$

The second version of the estimate (Jondrow et al. 1982) is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \mu_{*i} + \sigma_* \left[\frac{\phi(-\mu_{*i}/\sigma_*)}{1 - \Phi(-\mu_{*i}/\sigma_*)} \right] = \sigma_* \left[\frac{\phi(\epsilon_i \lambda / \sigma)}{1 - \Phi(\epsilon_i \lambda / \sigma)} - \left(\frac{\epsilon_i \lambda}{\sigma} \right) \right]$$

Normal-Exponential Model

Define $A = -\tilde{\mu}/\sigma_v$ and $\tilde{\mu} = -\epsilon - \sigma_v^2/\sigma_u$. Then, as it is shown by Kumbhakar and Lovell (2000), the conditional density is as follows:

$$f(u|\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_v\Phi(-\tilde{\mu}/\sigma_v)} \exp \left\{ -\frac{(u - \tilde{\mu})^2}{2\sigma^2} \right\}$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\tilde{\mu}, \sigma_v^2)$.

Using this result, it follows that the estimate of technical efficiency is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \left[\frac{1 - \Phi(\sigma_v - \tilde{\mu}_i/\sigma_v)}{1 - \Phi(-\tilde{\mu}_i/\sigma_v)} \right] \exp \left\{ -\tilde{\mu}_i + \frac{1}{2}\sigma_v^2 \right\}$$

The second version of the estimate is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \tilde{\mu}_i + \sigma_v \left[\frac{\phi(-\tilde{\mu}_i/\sigma_v)}{1 - \Phi(-\tilde{\mu}_i/\sigma_v)} \right] = \sigma_v \left[\frac{\phi(A)}{\Phi(-A)} - A \right]$$

Normal-Truncated Normal Model

Define $\tilde{\mu} = (-\sigma_u^2\epsilon_i + \mu\sigma_v^2)/\sigma^2$ and $\sigma_*^2 = \sigma_u^2\sigma_v^2/\sigma^2$. Then, as it is shown by Kumbhakar and Lovell (2000), the conditional density is as follows:

$$f(u|\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_*[1 - \Phi(-\tilde{\mu}/\sigma_*)]} \exp \left\{ -\frac{(u - \tilde{\mu})^2}{2\sigma_*^2} \right\}$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\tilde{\mu}, \sigma_*^2)$.

Using this result, it follows that the estimate of technical efficiency is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \frac{1 - \Phi(\sigma_* - \tilde{\mu}_i/\sigma_*)}{1 - \Phi(-\tilde{\mu}_i/\sigma_*)} \exp \left\{ -\tilde{\mu}_i + \frac{1}{2}\sigma_*^2 \right\}$$

The second version of the estimate is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \tilde{\mu}_i + \sigma_* \left[\frac{\phi(\tilde{\mu}_i/\sigma_*)}{1 - \Phi(-\tilde{\mu}_i/\sigma_*)} \right]$$

Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements

This section describes how you can refer to the parameters in the MODEL statement when you use the RESTRICT, TEST, BOUNDS, or INIT statement. The following examples use the RESTRICT statement, but the same remarks apply to naming parameters when you use the TEST, BOUNDS, or INIT statement. The names of the parameters are shown in the “Parameter Estimates” table if you specify the PRINTINTERNALNAMES option. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the RESTRICT, TEST, BOUNDS, or INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC FRONTIER statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column that shows the internal name of each parameter is added to the “Parameter Estimates” table.

To impose a restriction on a parameter that is related to a regressor in the MODEL statement, you simply use the name of the regressor itself to refer to its associated parameter. Suppose your model is

```
model y = x1 x2 x5;
```

where x1 through x5 are continuous variables. To restrict the parameter that is associated with the regressor x5 to be greater than 1.7, use the following statement:

```
RESTRICT x5 > 1.7;
```

Referring to Class-Level Parameters

When your MODEL statement includes a classification variable, you can impose restrictions on the parameters that are associated with each level that is related to the classification variable as follows.

Suppose that your classification variable is named C and has three levels: 0, 1, 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 C / cost type=half;
```

Adding a classification variable as a regressor to your model introduces additional parameters into your model, each of which is associated with one of the levels of the classification variable. You can form the name of the parameter that is associated with a particular level of your classification variable by inserting an underscore between the name of the classification variable and the value of the level. Thus, to restrict the parameter that is associated with level 0 of the classification variable C to always be greater than 0.7, you refer to the parameter as C_0 and provide the following statement:

```
RESTRICT C_0 > 0.7;
```

Referring to Parameters Associated with Interactions between Regressors

When a regressor in your model involves an interaction between other regressors, you can impose restrictions on the parameters that are associated with the interaction.

Suppose you have the following model:

```
model y = x1 x2 x3*x4 / cost type=half;
```

You can form the name of the parameter that is associated with the interaction regressor $x3*x4$ by replacing the multiplication sign with an underscore. Thus, $x3_x4$ refers to the parameter that is associated with the interaction regressor $x3*x4$.

You use the same approach to refer to interactions between regressors and classification variables. Suppose you have a classification variable that is named C and has three levels: 0, 1, 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 C*x3 / cost type=half;
```

The interaction between the continuous variable $x3$ and the classification variable C introduces three additional parameters, which are named $x3_C_0$, $x3_C_1$, and $x3_C_2$. Note that, although the order of the terms in the interaction is C followed by $x3$, the name of the parameter associated with the interaction is formed by placing the name of the continuous variable $x3$ first, followed by an underscore, followed by the name of the classification variable C , followed by an underscore, and then followed by the level value. Once again, depending on the parameterization that you specify in your CLASS statement, for each interaction in your model that involves a classification variable, one of the parameters associated with that interaction might be dropped from your model before optimization.

The name of a parameter that is associated with a nested interaction is formed in a slightly different way. Suppose you have a classification variable that is named C and has three levels: 0, 1, 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 x3(C) / cost type=half;
```

The nested interaction between the continuous variable $x3$ and the classification variable C introduces three additional parameters, which are named $x3_C_0$, $x3_C_1$, and $x3_C_2$. Note that the name in each case is formed from the name of the regressor by replacing the left and right parentheses with underscores and then appending another underscore, followed by the level value.

Referring to Class-Level Parameters with Negative Values

When the value of a level is a negative number, you must replace the minus sign with an underscore when you form the name of the parameter that is associated with that particular level of the classification variable. For example, suppose that your classification variable is named D and has four levels: -1 , 0, 1, 2. Suppose that your model is the following:

```
class D;
model y = x1 x2 D / cost type=half;
```

To restrict the parameter that is associated with level -1 of the classification variable D to always be less than 0.4, you refer to the parameter as D_1 (note that there are two underscores in this parameter name: one

to connect the name of the classification variable to its value, and the other to replace the minus sign in the value itself) and provide the following statement:

```
RESTRICT D__1 < 0.4;
```

Dropping a Class-Level Parameter to Avoid Collinearity

Depending on the parameterization that you impose on your classification variable, one of the parameters that are associated with its levels might be dropped from your model before optimization in order to avoid collinearity. For example, when the default parameterization is imposed, the parameter that is associated with the last level of your classification variable is dropped before optimization. If you attempt to impose a restriction on a dropped parameter by using the *RESTRICT* statement, PROC FRONTIER issues an error message in the log.

For example, suppose again that your classification variable is named *C* and has three levels: 0, 1, 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 C / cost type=half;
```

Because no additional options are specified in the *CLASS* statement, the default parameterization is assumed. This means that the parameter named *C_2* (which is the parameter associated with the last level of your classification variable) is dropped from your model before the optimizer is invoked. Therefore, an error occurs if you attempt to restrict the *C_2* parameter in any way by referring to it in a *RESTRICT* statement. In this example, the following *RESTRICT* statement generates an error:

```
RESTRICT C_2 < 0.3;
```

Referring to Implicit Parameters

Unless you specify the *NOINT* option, you can impose a restriction on the parameter that is associated with the intercept in your model simply by referring to it by name: *Intercept*.

When you specify the inefficiency term as truncated-normal, PROC FRONTIER automatically adds the *_Mu* parameter, the mean of the truncated normal distribution, to your model prior to optimization. Also, for each model, PROC FRONTIER includes the parameters *_Sigma_v* and *_Sigma_u*, which are the estimates of the standard deviation of the idiosyncratic and technical inefficiency terms of the model error, respectively. You can impose restrictions on these implicit parameters.

For example, for the truncated-normal model, you can restrict *_Mu* to be positive as follows:

```
RESTRICT _Mu > 0;
```

Tests on Parameters

In general, you can write the tested hypothesis as

$$H_0 : \mathbf{h}(\theta) = 0$$

where $\mathbf{h}(\theta)$ is an $r \times 1$ vector-valued function of the parameters θ that is given by the r expressions that you specify in the TEST statement.

Let $\hat{V}$ be the estimate of the covariance matrix of $\hat{\theta}$. Let $\hat{\theta}$ be the unconstrained estimate of θ , and let $\tilde{\theta}$ be the constrained estimate of θ such that $\mathbf{h}(\tilde{\theta}) = 0$. Let

$$A(\theta) = \partial \mathbf{h}(\theta) / \partial \theta \big|_{\hat{\theta}}$$

Using this notation, you compute the test statistics for the three types of tests as follows:

- The Wald test statistic is defined as

$$W = \mathbf{h}'(\hat{\theta}) \left(A(\hat{\theta}) \hat{V} A'(\hat{\theta}) \right)^{-1} \mathbf{h}(\hat{\theta})$$

The Wald test is not invariant to reparameterization of the model (Gregory and Veall 1985; Gallant 1987, p. 219). For more information about the theoretical properties of the Wald test, see Phillips and Park (1988).

- The Lagrange multiplier test statistic is

$$LM = \lambda' A(\tilde{\theta}) \tilde{V} A'(\tilde{\theta}) \lambda$$

where λ is the vector of Lagrange multipliers from the computation of the restricted estimate $\tilde{\theta}$.

- The likelihood ratio test statistic is

$$LR = 2 \left(L(\hat{\theta}) - L(\tilde{\theta}) \right)$$

where $\tilde{\theta}$ represents the constrained estimate of θ and L is the concentrated log-likelihood value.

For each type of test, under the null hypothesis the test statistic is asymptotically distributed as a χ^2 random variable with r degrees of freedom, where r is the number of expressions that you specify in the TEST statement. The p -values that are reported for the tests are computed from the $\chi^2(r)$ distribution and are only asymptotically valid.

Monte Carlo simulations suggest that the asymptotic distribution of the Wald test is a poorer approximation of its small sample distribution than that of the other two tests. However, the Wald test has the lowest computational cost, because it does not require computation of the constrained estimate $\tilde{\theta}$.

The following statements use the TEST statement to perform a likelihood ratio test:


```

proc frontier;
  class cl;
  model y = x1 x2 cl / production type=trunc;
  test x1 = 0, x2 * .5 + 2 * x1 = 0 /lr;
run;

```

For more information, see *SAS/ETS User's Guide*.

ODS Table Names

PROC FRONTIER assigns a name to each table that it creates. You can use these names to denote the table when you use the Output Delivery System (ODS) to select tables and create output data tables. These table names are listed in [Table 19.3](#).

Table 19.3 ODS Tables Produced by PROC FRONTIER

Table Name	Description	Statement	Option
ClassInfo	Level information from the CLASS statement	CLASS	Default
ConvergenceStatus	Convergence status	MODEL	Default
CorrB	Correlation of parameter estimates	PROC FRONTIER	CORRB
CovB	Covariance of parameter estimates	PROC FRONTIER	COVB
DepVarSummary	Summary of dependent variable statistics	PROC FRONTIER	Default
FitModelSummary	Summary of nonlinear estimation	MODEL	Default
InitialParameterEstimates	Initial parameter estimates	PROC FRONTIER	PRINTALL
IterationHistory	Optimization iteration history	PROC FRONTIER	PRINTALL
NObs	Observation information	PROC FRONTIER	Default
OptimizationResults	Optimization results	PROC FRONTIER	PRINTALL
OptimizationSettings	Nonlinear optimizer settings	PROC FRONTIER	PRINTALL
ParameterEstimates	Parameter estimates	MODEL	Default
TestResults	Hypothesis test statistics	TEST	Default
VarianceStats	Variance statistics	MODEL	Default

Examples: The FRONTIER Procedure

Example 19.1: Stochastic Frontier Production Model: Truncated-Normal

This example shows how to estimate a stochastic frontier production model in which the inefficiency term has a truncated-normal distribution.

The following DATA step generates a data table that contains 100,000 observations from this model. The model contains three input variables, one of which is a classification variable.

```
data input_dataset;
  keep y x1 x2 c1 j;
  call streaminit(24569);
  do j = 1 to 100000;
    x1 = 2 + rand("normal");
    x2 = rand("uniform");
    c1 = rand("bernoulli",0.4);
    sigma_v = rand("normal");
    sigma_u = 0;
    do until(sigma_u > 0);
      sigma_u = rand("normal",0.75,1.3);
    end;
    xb = 5 - 0.5*x1 + 0.8*x2 - 0.65*(c1 = 0);
    y = xb + sigma_v - sigma_u;
    output;
  end;
run;

data mylib.input_dataset;
  set input_dataset;
run;
```

The following statements estimate the model:

```
/*-- Stochastic Frontier Production Model: Truncated-Normal --*/
proc frontier data=mylib.input_dataset method=quanew;
  class c1;
  model y = x1 x2 c1/ production type=trunc;
run;
```

Output 19.1.1 shows the results.

Output 19.1.1 Production Model with Truncated-Normal Distribution

The FRONTIER Procedure

Class Level Information		
Class	Levels	Values
c1	2	0 1

Output 19.1.1 *continued*

Observation Information					
Number of Observations		100000			
Number of Missing Observations		0			

Summary Statistics of Dependent Variable				
Variable	Mean	Standard Error	Minimum	Maximum
y	2.649665	1.503944	-4.0917	8.684373

Model Fit Summary	
Dependent Variable	y
Data Set	INPUT_DATASET
Model	Production
Inefficiency Term Distribution	Truncated normal
Log Likelihood	-172207
Maximum Absolute Gradient	0.41805
Number of Iterations	20
Optimization Method	Quasi-Newton
AIC	344428
SBC	344494.6
Covariance Estimation	Hessian

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	5.031248	0.052109	96.55	<.0001
x1	1	-0.504582	0.004228	-119.35	<.0001
x2	1	0.786293	0.014637	53.72	<.0001
cl 0	1	-0.636774	0.008630	-73.79	<.0001
cl 1	0	0	.	.	.
_Sigma_v	1	0.977354	0.015281	63.96	<.0001
_Sigma_u	1	1.344170	0.023412	57.41	<.0001
_Mu	1	0.731860	0.127985	5.72	<.0001

Variance Statistics		
Parameter	Estimate	Standard Error
Sigma2	2.762015	0.080819
Gamma	0.654158	0.007898

In addition to the parameters that are estimated for the other stochastic frontier models, PROC FRONTIER also estimates the parameter `_Mu` for truncated-normal models. `_Mu` is the mean of the truncated-normal distribution. The parameter `cl_1` indicates the classification variable for the level 1 and is dropped in the estimation to prevent perfect collinearity in the model. For more information, see the section “[Dropping a Class-Level Parameter to Avoid Collinearity](#)” on page 1173.

Example 19.2: Stochastic Frontier Cost Models

This example shows how to estimate the three types of stochastic frontier cost models.

The data for the cost model are provided by Christensen and Greene (1976). The data describe costs and production inputs of 145 US electricity producers in 1955. The model that is estimated follows the nonhomogeneous version of the Cobb-Douglas cost function:

$$\log\left(\frac{\text{Cost}}{\text{FPrice}}\right) = \beta_0 + \beta_1 \log\left(\frac{\text{KPrice}}{\text{FPrice}}\right) + \beta_2 \log\left(\frac{\text{LPrice}}{\text{FPrice}}\right) + \beta_3 \log(\text{Output}) + \beta_4 \frac{1}{2} \log(\text{Output})^2 + \epsilon$$

All dollar values are normalized by fuel price. The quadratic log of the output is added to capture nonlinearities due to scale effects in cost functions. Five new variables, `log_C_PF`, `log_PK_PF`, `log_PL_PF`, `log_y`, and `log_y_sq`, are created to reflect transformations. The following statements create the data set and transformed variables:

```

title1 'Estimating a Stochastic Frontier Cost Model';

data electricity;
    input Firm Year Cost Output LPrice LShare KPrice KShare FPrice FShare;
datalines;
1  1955  .0820  2.0  2.090  .3164  183.000  .4521  17.9000  .2315
2  1955  .6610  3.0  2.050  .2073  174.000  .6676  35.1000  .1251
3  1955  .9900  4.0  2.050  .2349  171.000  .5799  35.1000  .1852
... more lines ...

/* Data transformations */
data electricity;
    set electricity;
    label Firm="firm index"
           Year="1955 for all observations"
           Cost="Total cost"
           Output="Total output"
           LPrice="Wage rate"
           LShare="Cost share for labor"
           KPrice="Capital price index"
           KShare="Cost share for capital"
           FPrice="Fuel price"
           FShare="Cost share for fuel";
    log_C_PF=log(Cost/FPrice);
    log_PK_PF=log(KPrice/FPrice);
    log_PL_PF=log(LPrice/FPrice);
    log_y=log(Output);
    log_y_sq=log_y**2/2;
run;

data mylib.electricity;
    set electricity;
run;
```

The following statements estimate a stochastic frontier exponential cost model that uses the Christensen and Greene (1976) data:

```
/*-- Stochastic Frontier Cost Model --*/
proc frontier data=mylib.electricity;
  model log_C_PF = log_PK_PF log_PL_PF log_y log_y_sq / type=exponential cost;
run;
```

Output 19.2.1 shows the results.

Output 19.2.1 Exponential Distribution Estimating a Stochastic Frontier Cost Model

The FRONTIER Procedure

Observation Information	
Number of Observations	159
Number of Missing Observations	0

Summary Statistics of Dependent Variable				
Standard				
Variable	Mean	Error	Minimum	Maximum
log_C_PF	-1.38206	1.495751	-5.67619	1.728358

Model Fit Summary	
Dependent Variable	log_C_PF
Data Set	ELECTRICITY
Model	Cost
Inefficiency Term Distribution	Exponential
Log Likelihood	-23.3043
Maximum Absolute Gradient	0.000105
Number of Iterations	12
Optimization Method	Newton-Raphson
AIC	60.6086
SBC	82.09093
Covariance Estimation	Hessian

Parameter Estimates					
Parameter	DF	Estimate	Standard	t Value	Approx Pr > t
			Error		
Intercept	1	-4.983210	0.543328	-9.17	<.0001
log_PK_PF	1	0.090242	0.109202	0.83	0.4086
log_PL_PF	1	0.504299	0.118263	4.26	<.0001
log_y	1	0.427182	0.066680	6.41	<.0001
log_y_sq	1	0.066120	0.010079	6.56	<.0001
_Sigma_v	1	0.154998	0.020271	7.65	<.0001
_Sigma_u	1	0.265581	0.033614	7.90	<.0001

Output 19.2.1 *continued*

Variance Statistics		
Parameter	Estimate	Standard Error
Sigma2	0.094558	0.015889
Gamma	0.745929	0.083677

Similarly, the stochastic frontier cost model can be estimated using the TYPE=HALF or TYPE=TRUNCATED option, which represents half-normal or truncated-normal errors, respectively.

The following statements estimate the half-normal model:

```

/*-- Stochastic Frontier Cost Model --*/
proc frontier data=mylib.electricity;
    model log_C_PF = log_PK_PF log_PL_PF log_y log_y_sq / type=half cost;
run;

```

Output 19.2.2 shows the result.

Output 19.2.2 Half-Normal Distribution**Estimating a Stochastic Frontier Cost Model****The FRONTIER Procedure**

Model Fit Summary						
Dependent Variable		log_C_PF				
Data Set		ELECTRICITY				
Model		Cost				
Inefficiency Term Distribution		Half-normal				
Log Likelihood		-34.953				
Maximum Absolute Gradient		0.000265				
Number of Iterations		12				
Optimization Method		Newton-Raphson				
AIC		83.90607				
SBC		105.3884				
Covariance Estimation		Hessian				

Parameter Estimates						
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t	
Intercept	1	-4.434625	0.690198	-6.43	<.0001	
log_PK_PF	1	0.069623	0.136250	0.51	0.6094	
log_PL_PF	1	0.474580	0.146812	3.23	0.0012	
log_y	1	0.256874	0.080777	3.18	0.0015	
log_y_sq	1	0.088051	0.011817	7.45	<.0001	
_Sigma_v	1	0.207637	0.039222	5.29	<.0001	
_Sigma_u	1	0.373810	0.073605	5.08	<.0001	

Output 19.2.2 *continued*

Variance Statistics		
Parameter	Estimate	Standard Error
Sigma2	0.182847	0.042585
Gamma	0.764212	0.132839

The following statements estimate the truncated-normal model:

```

/*-- Stochastic Frontier Cost Model --*/
proc frontier data=mylib.electricity;
  model log_C_PF = log_PK_PF log_PL_PF log_y log_y_sq / type=truncated cost;
run;

```

Output 19.2.3 shows the results.

Output 19.2.3 Truncated-Normal Distribution
Estimating a Stochastic Frontier Cost Model

The FRONTIER Procedure

Model Fit Summary					
Dependent Variable		log_C_PF			
Data Set		ELECTRICITY			
Model		Cost			
Inefficiency Term Distribution		Truncated normal			
Log Likelihood		-36.5435			
Maximum Absolute Gradient		0.007395			
Number of Iterations		18			
Optimization Method		Newton-Raphson			
AIC		89.0871			
SBC		113.6383			
Covariance Estimation		Hessian			

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	-3.904334	37.001296	-0.11	0.9160
log_PK_PF	1	0.071737	0.143794	0.50	0.6179
log_PL_PF	1	0.462589	0.155598	2.97	0.0029
log_y	1	0.130496	0.058363	2.24	0.0254
log_y_sq	1	0.103576	0.009958	10.40	<.0001
_Sigma_v	1	0.302404	0.633030	0.48	0.6329
_Sigma_u	1	0.035631	5.365220	0.01	0.9947
_Mu	1	0.192831	36.995070	0.01	0.9958

Variance Statistics		
Parameter	Estimate	Standard Error
Sigma2	0.092718	0.010408
Gamma	0.013693	4.123680

References

- Aigner, C., Lovell, C. A. K., and Schmidt, P. (1977). "Formulation and Estimation of Stochastic Frontier Production Function Models." *Journal of Econometrics* 6:21–37.
- Battese, G. E., and Coelli, T. J. (1988). "Prediction of Firm-Level Technical Efficiencies with a Generalized Frontier Production Function and Panel Data." *Journal of Econometrics* 38:387–399.
- Christensen, L. R., and Greene, W. H. (1976). "Economies of Scale in U.S. Electric Power Generation." *Journal of Political Economy* 84:655–676.
- Coelli, T. J., Prasada Rao, D. S., and Battese, G. E. (1998). *An Introduction to Efficiency and Productivity Analysis*. London: Kluwer Academic.
- Gallant, A. R. (1987). *Nonlinear Statistical Models*. New York: John Wiley & Sons.
- Gregory, A. W., and Veall, M. R. (1985). "On Formulating Wald Tests for Nonlinear Restrictions." *Econometrica* 53:1465–1468.
- Jondrow, J., Lovell, C. A. K., Materov, I. S., and Schmidt, P. (1982). "On the Estimation of Technical Efficiency in the Stochastic Frontier Production Function Model." *Journal of Econometrics* 19:233–238.
- Kumbhakar, S. C., and Lovell, C. A. K. (2000). *Stochastic Frontier Analysis*. New York: Cambridge University Press.
- Meeusen, W., and van den Broeck, J. (1977). "Efficiency Estimation from Cobb-Douglas Production Functions with Composed Error." *International Economic Review* 18:435–444.
- Phillips, C. B., and Park, J. Y. (1988). "On Formulating Wald Tests of Nonlinear Restrictions." *Econometrica* 56:1065–1083.
- Sickles, R. C., and Zelenyuk, V. (2019). *Measurement of Productivity and Efficiency: Theory and Practice*. Cambridge: Cambridge University Press.
- Zellner, A., and Revankar, N. S. (1969). "Generalized Production Functions." *Review of Economic Studies* 36:241–250.

Chapter 20

HMM Procedure

Contents

Overview: HMM Procedure	1184
Using CAS Sessions and CAS Engine Librefs	1186
Getting Started: HMM Procedure	1187
Gaussian Hidden Markov Model	1187
Gaussian Hidden Markov Model for Cross-Sectional Time Series Data	1198
Gaussian Mixture Hidden Markov Model for Time Series Data and Cross-Sectional Time Series Data	1202
Regime-Switching Regression Model	1222
Regime-Switching Autoregression Model	1228
Finite Hidden Markov Model	1233
Poisson Hidden Markov Model	1241
Syntax: HMM Procedure	1244
Functional Summary	1244
PROC HMM Statement	1248
DECODE Statement	1250
DISPLAY Statement	1250
DISPLAYOUT Statement	1251
ESTIMATE Statement	1252
EVALUATE Statement	1253
FILTER Statement	1253
FORECAST Statement	1254
ID Statement	1255
INITIAL Statement	1255
MODEL Statement	1256
OPTIMIZE Statement	1259
PRIOR Statement	1264
SCORE Statement	1266
SMOOTH Statement	1267
STORE Statement	1267
Details: HMM Procedure	1268
Hidden Markov Model	1268
Gaussian Hidden Markov Model	1272
Gaussian Mixture Hidden Markov Model	1272
Regime-Switching Regression Model	1273
Regime-Switching Autoregression Model	1274
Regime-Switching Mean-Adjusted Autoregression Model	1276

Finite Hidden Markov Model	1278
The %FiniteHMM Macro	1278
Poisson Hidden Markov Model	1303
Parameter Estimation Methods	1304
Optimization Algorithms	1309
Distributed Multistart	1310
Matrix Expression	1311
Label Switching Problem	1321
Analytic Store Technology	1333
Information Criteria	1337
Missing Values	1337
Data Table Output	1338
ODS Table Names	1342
Several Types of Models as Special Cases of Hidden Markov Models	1344
Examples: HMM Procedure	1350
Example 20.1: Discovering the Hidden Market States by Using the Regime-Switching Autoregression Model	1350
Example 20.2: Clustering Time Series by Using the Gaussian Hidden Markov Model .	1384
Example 20.3: Analysis of the Business Cycle by Using the Regime-Switching Mean- Adjusted Autoregression Model	1395
Example 20.4: Analysis of English Characters by Using a Finite HMM	1407
References	1412

Overview: HMM Procedure

The HMM procedure supports hidden Markov models (HMMs), which have been widely applied in economics, finance, science, and engineering. The hidden Markov models have many well-known aliases, such as the general state space model (GSSM), regime-switching model (RSM), Markov-switching model (MSM), and Markov regime-switching model (MRSM). PROC HMM supports the finite hidden Markov model (finite HMM), the Poisson hidden Markov model (Poisson HMM), the Gaussian hidden Markov model (Gaussian HMM), the Gaussian mixture hidden Markov model (GM HMM), the regime-switching regression model (RS REG, also known as the regression hidden Markov model or REG HMM), and the regime-switching autoregression model (RS AR, also known as the autoregressive hidden Markov model or AR HMM) in both the standard and mean-adjusted forms.

The HMM procedure supports the following:

- cross-sectional time series data
- the maximum likelihood (ML) method and the maximum a posteriori (MAP) method for parameter estimation, which is a nonlinear optimization problem
- the prior distribution for parameters when you use the MAP method

- four types of optimization algorithms: the expectation-maximization (EM) algorithm, the active set algorithm, the interior point algorithm, and the stochastic gradient descent (SGD) algorithm
- multistart mode for the active set and interior point optimization algorithms; in this mode, the local solver approaches the problem from multiple starting points, possibly finding a better local optimum as a result. However, the computing cost in multistart mode can be huge.
- specification of initial values for optimization
- state-independent constraints on parameters
- output of parameter estimates to a data table, which can be read back later in order to use those parameter estimates as the initial values for a new estimation
- output of the covariance matrix of parameter estimates
- specification of the order in which to label the states (and components in the case of GM HMMs) in order to solve the label switching problem, and output of the changes of state labels (and component labels in the case of GM HMMs)
- estimation of several models that have different numbers of states or different orders of autoregressive process (or both) in order to help you with the important and difficult task of model selection
- output of the filtering, smoothing, and decoding results
- output of the mean, standard error, covariance matrix, median, and confidence interval of forecasts of dependent variables
- output of the probabilities of states in the forecasts
- multistep forecasts after each observation
- output of the log likelihood up to any observation
- output of the following information criteria: Akaike's information criterion (AIC), the corrected AIC (AICC), the Bayesian information criterion (BIC), and the Hannan-Quinn criterion (HQC)
- saving the model information and using it later to score other new data
- the analytic store technology. You can even apply HMMs to new data on a platform that does not support the HMM procedure.

PROC HMM requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: HMM Procedure

This section provides several examples of the types of models that the HMM procedure supports.

Gaussian Hidden Markov Model

Let $\mathbf{Y}_t = (y_{1t}, \dots, y_{pt})'$, $t = 1, \dots, T$, denote a p -dimensional time series vector of random variables. Conditional on a latent variable $\mathbf{S}_t$, $\mathbf{Y}_t$ follows a normal distribution,

$$\mathbf{Y}_t | \mathbf{S}_t \sim N(\mu_{\mathbf{S}_t}, \Sigma_{\mathbf{S}_t})$$

where $\mu_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the variable $\mathbf{S}_t$. The variable $\mathbf{S}_t$, the so-called state, follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the variable $\mathbf{S}_t$ is unobservable and follows a Markov chain and because $\mathbf{Y}_t$ follows a Gaussian distribution conditional on $\mathbf{S}_t$, the model is called the Gaussian hidden Markov model (Gaussian HMM). The Gaussian HMM is sometimes described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{\mu_i, \Sigma_i, i = 1, \dots, K\}$, consisting of parameters that define the state-dependent distribution of observable variables.

Consider a bivariate Gaussian HMM that has the following parameter values:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.001 & 0.999 \\ 0.999 & 0.001 \end{pmatrix}$$

$$\mu_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The following statements simulate the bivariate vector time series from the previous model to provide test data for the HMM procedure:

```

%let pi1 = 0.5;
%let a11 = 0.001;
%let a22 = 0.001;
%let mu1_1 = 1;
%let mu1_2 = 1;
%let sigma1_11 = 1;
%let sigma1_21 = 0;
%let sigma1_22 = 1;
%let mu2_1 = -1;
%let mu2_2 = -1;
%let sigma2_11 = 1;
%let sigma2_21 = 0;
%let sigma2_22 = 1;
%let T = 10000;
%let seed = 1234;

data DGPone;
  retain cd1_11 cd1_21 cd1_22 cd2_11 cd2_21 cd2_22;
  do t = 1 to &T.;
    if(t=1) then do;
      /* initial probability distribution */
      p = &pi1.;
      /* Cholesky decomposition of sigma1 */
      cd1_11 = sqrt(&sigma1_11.);
      cd1_21 = &sigma1_21./sqrt(&sigma1_11.);
      cd1_22 = sqrt(&sigma1_22.-&sigma1_21.*&sigma1_21./&sigma1_11.);
      /* Cholesky decomposition of sigma2 */
      cd2_11 = sqrt(&sigma2_11.);
      cd2_21 = &sigma2_21./sqrt(&sigma2_11.);
      cd2_22 = sqrt(&sigma2_22.-&sigma2_21.*&sigma2_21./&sigma2_11.);
    end;
    else do;
      /* transition probability matrix */
      if(lags=1) then p = &a11.;
      else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e1 = normal(&seed.); e2 = normal(&seed.);
    if(s=1) then do;
      /* (x,y) ~ N(mu1, Sigma1) at state 1 */
      x = &mu1_1. + cd1_11*e1;
      y = &mu1_2. + cd1_21*e1+cd1_22*e2;
    end;
    else do;
      /* (x,y) ~ N(mu2, Sigma2) at state 2 */
      x = &mu2_1. + cd2_11*e1;
      y = &mu2_2. + cd2_21*e1+cd2_22*e2;
    end;
    output;
    lags = s;
  end;
end;

```

```
run;

data One;
  set DGPone;
  keep t x y;
run;
```

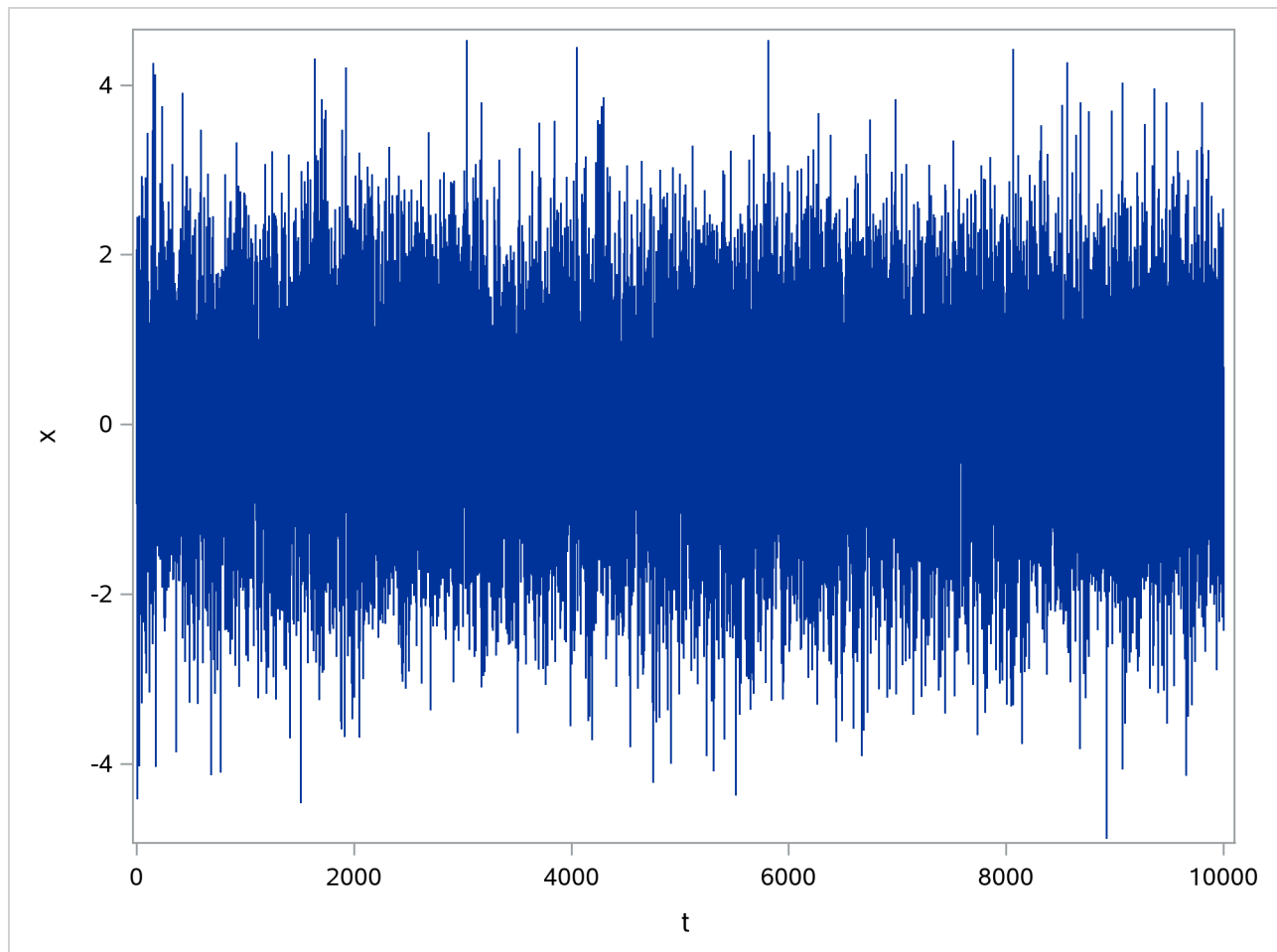
In general, the data generating process is unknown. What can be seen is the data table One, but not the data table DGPone.

The following statements plot the simulated series x :

```
proc sgplot data=One;
  series x=t y=x;
run;
```

Figure 20.1 shows the plot of x , whose points seem to fall randomly around 0.

Figure 20.1 Plot of Time Series x

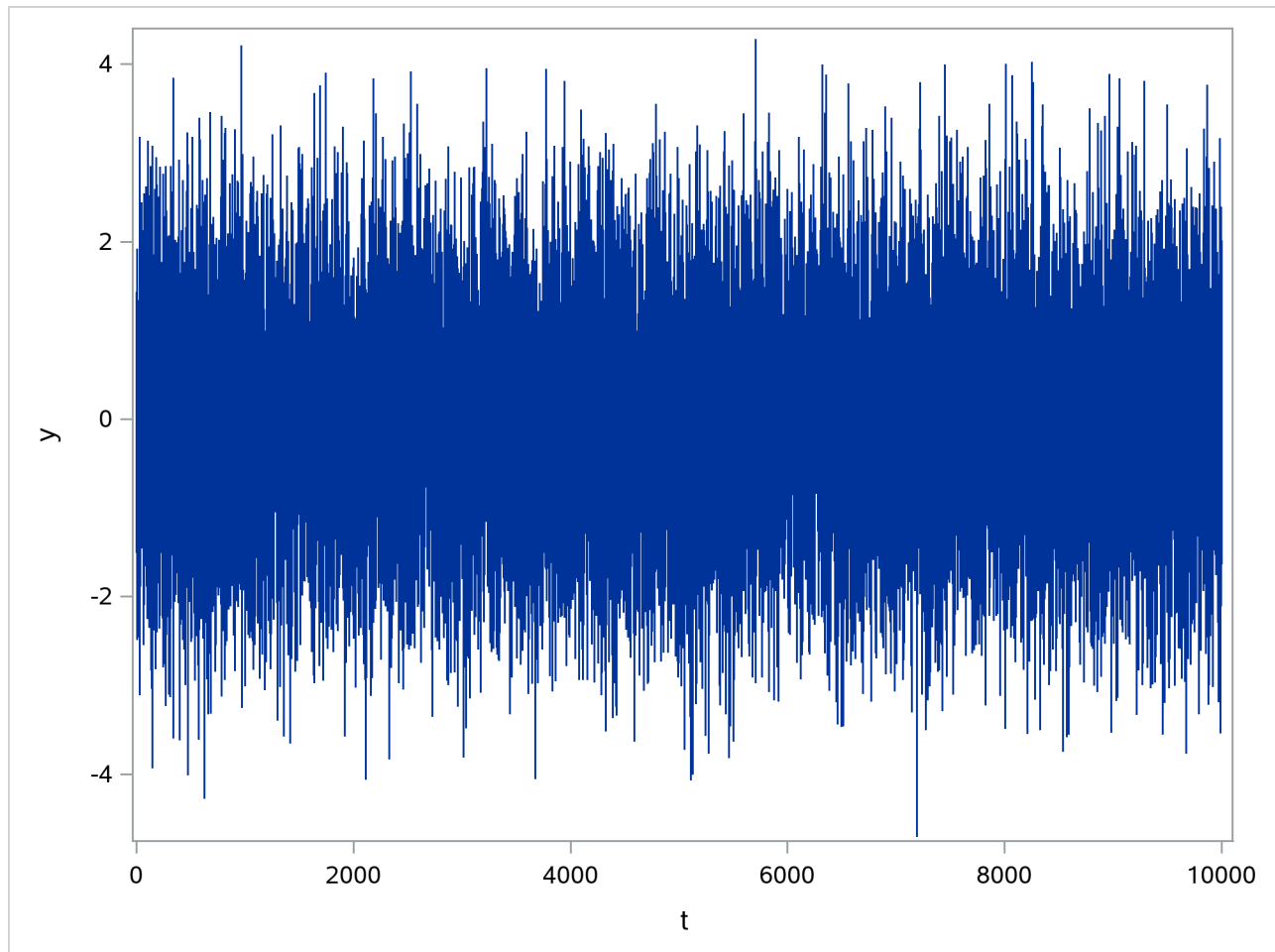


The following statements plot the simulated series y :

```
proc sgplot data=One;
  series x=t y=y;
run;
```

Figure 20.2 shows the plot of y , whose points also seem to fall randomly around 0.

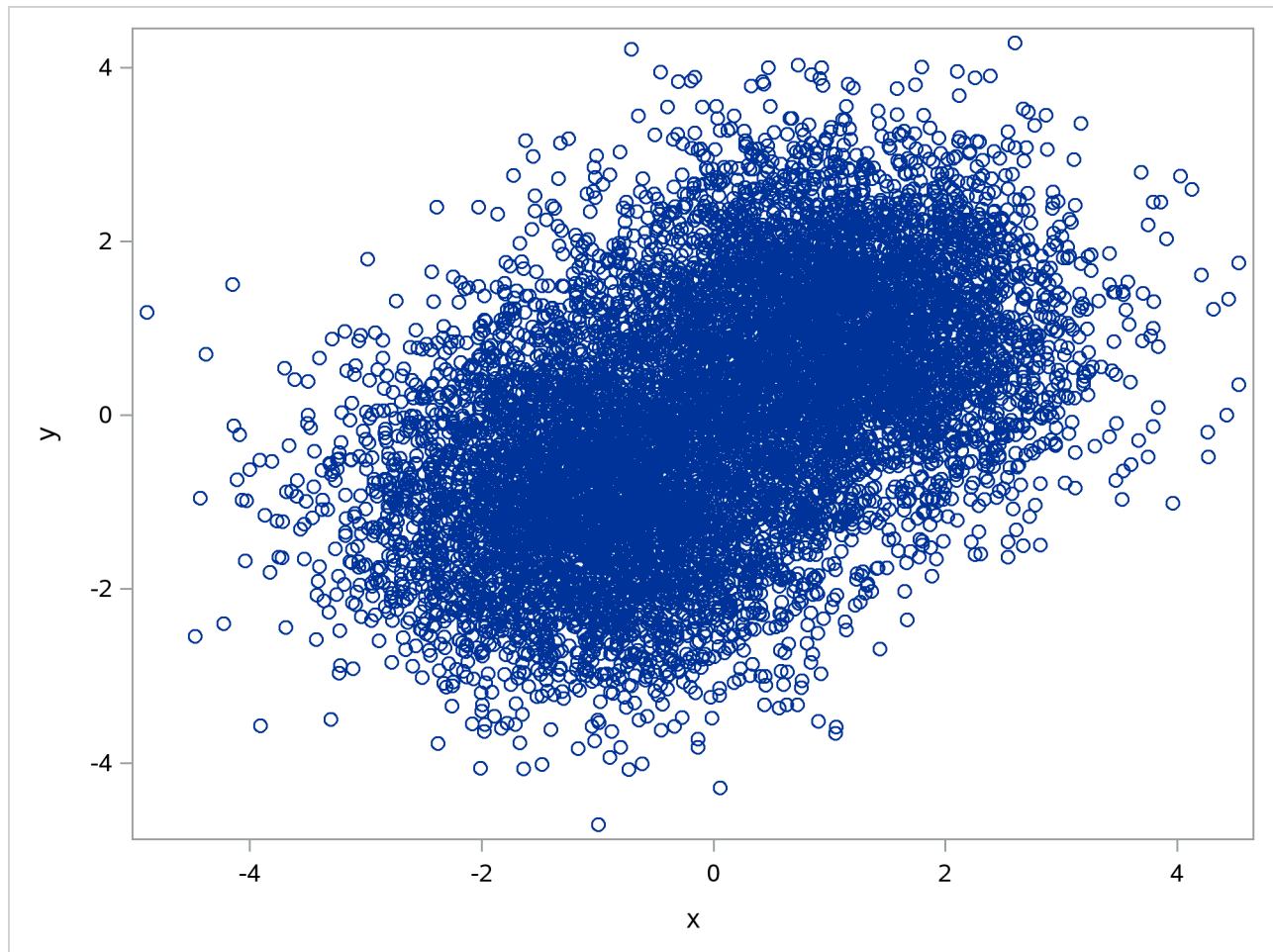
Figure 20.2 Plot of Time Series y



Ignoring the sequential information, the following statements plot the observed points (x, y) :

```
proc sgplot data=One;
  scatter y=y x=x;
run;
```

As shown in Figure 20.3, it seems that the points come from two different cluster centers, and many of them lie between two cluster centers.

Figure 20.3 Unclassified Observations

It is very difficult for any cluster analysis methods that cannot use the sequential information to find the correct states for the points that lie between two cluster centers. The simplest cluster analysis method to classify the points, because of the symmetry shown in Figure 20.3, is to draw a line $y = -x$, classify all points above the line to one state, and classify all points below the line to another state.¹ The following statements implement this simplest cluster analysis method and calculate the accuracy, based on the data table DGPone:

```
/* classify the data points by the line  $y = -x$  */
data Cluster;
  set One;
  if (y > -x) then state = 1;
  else state = 2;
  keep t state;
run;

data clusterCheck;
  merge DGPone(in=a) Cluster(in=b);
```

¹In theory, no cluster analysis method, without using the sequential information, could do a better job on this data table than this simplest cluster analysis method.

```

    by t;
    if(s=state) then correct = 1;
    else correct = 0;
    if(a=b);
    keep correct;
run;

data clusterAccuracy;
    set clusterCheck;
    retain count 0 correctCount 0;
    count = count + 1;
    correctCount = correctCount + correct;
    if(count>=T.-0.5) then do;
        accuracy = correctCount / count;
        if (accuracy<0.5) then accuracy = 1 - accuracy;
        output;
    end;
    keep accuracy;
run;

proc print data = clusterAccuracy noobs; run;

```

The accuracy of the cluster analysis, which is shown in [Figure 20.4](#), is 92.02%. This implies that the cluster analysis cannot choose the correct states for about 800 data points!

Figure 20.4 Accuracy of Cluster Analysis

<u>accuracy</u>
0.9202

What is the big difference that you get by using the sequential information through HMMs? Before estimating any model, make a copy of the data in the library:

```
data mylib.One; set One; run;
```

The following statements estimate the two-state Gaussian HMM and then classify the data points through decoding; here decoding means to find the best possible state path for the observed data. The SORT=DESC(MU) option requests that the states be labeled in descending order of the mean parameter values; for more information about the label switching problem, see the section “[Label Switching Problem](#)” on page 1321.

```

proc hmm data=mylib.One labelSwitch=(sort=desc(mu));
    id time=t;
    model x y / type=gaussian nstate=2;
    optimize printLevel=3 printIterFreq=1;
    decode out=mylib.oneDecode;
run;

```

The number of observations and model information are shown in [Figure 20.5](#).

Figure 20.5 Number of Observations and Model Information

Observation Information	
Number of Observations	10000
Number of Missing Observations	0

Model Information	
Type of Model	Gaussian HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	2

The default method, which is the maximum likelihood method, is applied; this is a nonlinear optimization problem. The initial parameter values and the objective function value at the start of the optimization are shown in [Figure 20.6](#).

Figure 20.6 Initial Parameter Values and Objective Function Value

Optimization Start	
Parameter	Estimate
TPM1_1	0.500000
TPM1_2	0.500000
TPM2_1	0.500000
TPM2_2	0.500000
MU1_1	0.642059
MU1_2	0.513308
MU2_1	-1.609545
MU2_2	-1.329369
SIGMA1_1_1	1.152038
SIGMA1_2_1	0.320856
SIGMA1_2_2	1.478814
SIGMA2_1_1	0.525065
SIGMA2_2_1	-0.243910
SIGMA2_2_2	0.892018

Initial Value of Objective Function	-34510.54106
-------------------------------------	--------------

The details of the optimization algorithm are shown in [Figure 20.7](#).

Figure 20.7 Optimization Algorithm Details

Algorithm Settings	
Maximum Number of Iterations	128
Tolerance for Optimality Error	1E-6
Tolerance for Infeasibility	1E-6
Maximum Allowed Time (seconds)	1.797693E308
Maximum Magnitude of Objective Function Value	1E20
Random Seed	1
Solution Type	1
Multi-Start Globalization Scheme	0
Multi-Start Globalization Bound Range	2

The iterations of the optimization process are shown in [Figure 20.8](#).

Figure 20.8 Iterations of the Optimization Process

Iteration History						
Iteration	Objective Function	Optimality Error	Infeasibility	Function Calls	Gradient Calls	Hessian Calls
1	-34510.54106	610.24392	0.00000	2	1	0
2	-29343.35393	350.54614	0.00000	11	10	1
3	-28606.45004	78.05203	0.00000	19	18	2
4	-28467.87680	1.42097	0.00000	28	27	3
5	-28460.41282	0.34998	0.00000	37	36	4
6	-28459.08728	0.10030	0.00000	46	45	5
7	-28458.80039	0.02968	0.00000	55	54	6
8	-28458.73576	0.00812	0.00000	64	63	7
9	-28458.72552	0.00167	0.00000	73	72	8
10	-28458.72484	0.00018	0.00000	82	81	9
11	-28458.72483	0.00000	0.00000	91	90	10

The final parameter values and the objective function value at the end of the optimization are shown in [Figure 20.9](#).

Figure 20.9 Final Parameter Values and Objective Function Value

Optimization Results	
Parameter	Estimate
TPM1_1	0.002071
TPM1_2	0.997929
TPM2_1	0.999863
TPM2_2	0.000137
MU1_1	1.006466
MU1_2	0.994184
MU2_1	-0.993036
MU2_2	-1.007869
SIGMA1_1_1	1.011152
SIGMA1_2_1	0.006387
SIGMA1_2_2	0.988329
SIGMA2_1_1	0.992333
SIGMA2_2_1	-0.005378
SIGMA2_2_2	1.008722
Final Value of Objective Function	
-28458.72483	

The estimates of the initial state probability vector (ISPV) are shown in [Figure 20.10](#).

Figure 20.10 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	Estimation
1	0.50048
2	0.49952

The estimates of the transition probability matrix (TPM) are shown in [Figure 20.11](#).

Figure 20.11 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	1	2
1	0.00207	0.99793
2	0.99986	0.00014

The estimates of the mean vector of the Gaussian distribution for each state are shown in [Figure 20.12](#).

Figure 20.12 Estimate of Mean Vector for Each State

Mu		
State	x	y
1	1.00647	0.99418
2	-0.99304	-1.00787

The estimates of the covariance matrix of the Gaussian distribution for each state are shown in [Figure 20.13](#).

Figure 20.13 Estimate of Covariance Matrix for Each State

Sigma			
State	Variable	x	y
1	x	1.01115	0.00639
	y	0.00639	0.98833
2	x	0.99233	-0.00538
	y	-0.00538	1.00872

The estimates of all parameters are shown in [Figure 20.14](#), which displays columns for the parameter name, estimate value, standard error, *t*-value, and *p*-value.

Figure 20.14 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.002071	0.000711	2.91	0.0036
TPM1_2	0.997929	0.000711	1403.15	<.0001
TPM2_1	0.999863	0.000342	2926.28	<.0001
TPM2_2	0.000137	0.000342	0.40	0.6888
MU1_1	1.006466	0.014228	70.74	<.0001
MU1_2	0.994184	0.014054	70.74	<.0001
MU2_1	-0.993036	0.014096	-70.45	<.0001
MU2_2	-1.007869	0.014217	-70.89	<.0001
SIGMA1_1_1	1.011152	0.020252	49.93	<.0001
SIGMA1_2_1	0.006387	0.014139	0.45	0.6514
SIGMA1_2_2	0.988329	0.019757	50.02	<.0001
SIGMA2_1_1	0.992333	0.019859	49.97	<.0001
SIGMA2_2_1	-0.005378	0.014159	-0.38	0.7041
SIGMA2_2_2	1.008722	0.020194	49.95	<.0001

The log likelihood and information criteria are shown in [Figure 20.15](#).

Figure 20.15 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-28458.72483
AIC	56941.449663
AICC	56941.480904
BIC	57027.973748
HQC	56970.737507

The accuracy of classification through the Gaussian HMM is calculated by the following statements:

```

data gHmmCheck;
  merge DGPone(in=a) mylib.oneDecode(in=b);
  by t;
  if(s=state) then do; correct=1; ds = s; end;
  else do; correct=0; ds = -s; end;
  if(a=b);
  keep t x y s correct ds;
run;

%let qFlipState = -1;
data gHmmAccuracy;
  set gHmmCheck;
  retain count 0 correctCount 0;
  count = count + 1;
  correctCount = correctCount + correct;
  if(count>&T.-0.5) then do;
    accuracy = correctCount / count;
    if(accuracy<0.5) then call symputx("qFlipState",1,'G');
    else call symputx("qFlipState",0,'G');
    if(accuracy<0.5) then accuracy = 1 - accuracy;
    output;
  end;
  keep accuracy;
run;

data gHmmCheck;
  set gHmmCheck;
  if(&qFlipState.=1) then do;
    ds = -ds;
    correct = 1 - correct;
  end;
run;

proc print data = gHmmAccuracy noobs; run;

```

The accuracy of the Gaussian HMM, which is shown in [Figure 20.16](#), is 99.99%. This means that the Gaussian HMM successfully takes advantage of the sequential information: only 1 data point cannot be correctly classified, compared to about 800 points that cannot be correctly classified by the cluster analysis method without the sequential information.

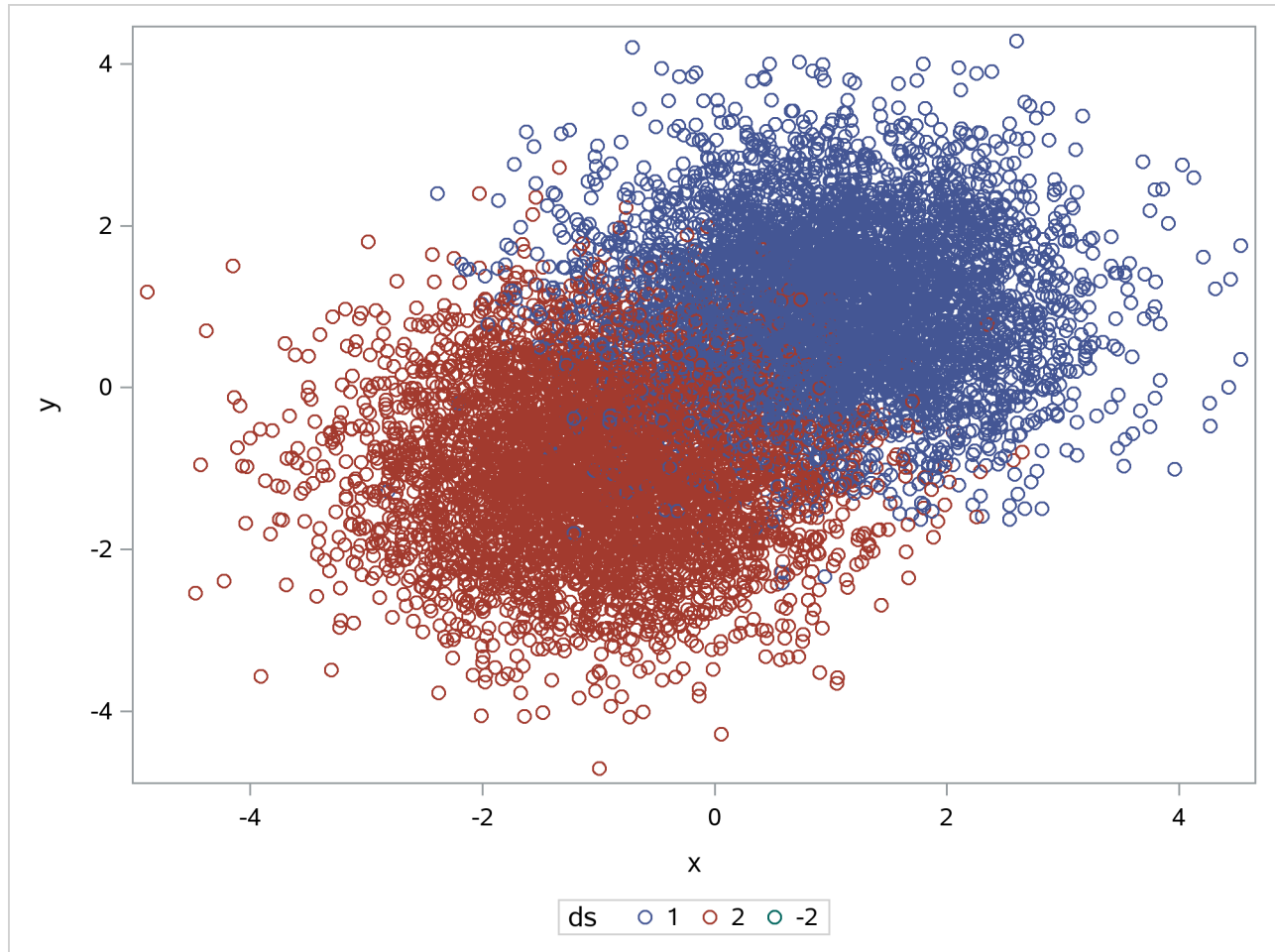
Figure 20.16 Accuracy of Decoding by the Gaussian HMM

accuracy
0.9999

To show how well the data are classified, the previous statements label any incorrectly classified data points with negative state values. In this example, only 1 point is not correctly classified. You plot the classified data points by using the following statements. The plot is shown in [Figure 20.17](#).

```
proc sgplot data=gHmmCheck;
  scatter y=y x=x / group=ds;
run;
```

Figure 20.17 Classified Data Points by Gaussian HMM



Gaussian Hidden Markov Model for Cross-Sectional Time Series Data

Cross-sectional time series data consist of observations over both time and many subjects (such as individuals, objects, firms, or geographical areas). Let $\mathbf{Y}_{n,t} = (y_{n,1t}, \dots, y_{n,pt})'$, where $n = 1, \dots, N$ and $t = 1, \dots, T_n$, denote a p -dimensional vector of random variables at the t th position in the n th section, where T_n is the sample size of the n th section. The $\mathbf{Y}_{n,t}$ follows the Gaussian HMM; that is,

$$\mathbf{Y}_{n,t} | \mathbf{S}_{n,t} \sim N(\mu_{\mathbf{S}_{n,t}}, \Sigma_{\mathbf{S}_{n,t}})$$

where $\mu_{\mathbf{S}_{n,t}}$ and $\Sigma_{\mathbf{S}_{n,t}}$ are mean and covariance parameters, respectively, whose values depend on the state variable $\mathbf{S}_{n,t}$. $\mathbf{S}_{n,t}$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_{n,t} | \mathbf{S}_{n,t-1}, \mathbf{S}_{n,t-2}, \dots, \mathbf{S}_{n,1}) = p(\mathbf{S}_{n,t} | \mathbf{S}_{n,t-1})$$

where $p(.|.)$ denotes the conditional probability. $\mathbf{S}_{n,t}$ is independent of any $\mathbf{S}_{m,s}, m = 1, \dots, N, s = 1, \dots, T_m$, and $m \neq n$. The range of $\mathbf{S}_{n,t}$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_{n,t} = j | \mathbf{S}_{n,t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The initial state probability vector (ISPV), π , of the first state $\mathbf{S}_{n,1}$ is

$$\pi = \{\pi_i = p(\mathbf{S}_{n,1} = i), i = 1, \dots, K\}$$

The Gaussian HMM for cross-sectional time series data, which is the same as the Gaussian HMM for time series data, can be described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{\mu_i, \Sigma_i, i = 1, \dots, K\}$, which consists of parameters that define the state-dependent distribution of observable variables.

Consider a univariate six-state Gaussian HMM that has the following parameter values:

$$\pi = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.95 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.95 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.95 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.95 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.95 \end{pmatrix}$$

$$\mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \\ \mu_5 \\ \mu_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \\ \Sigma_3 \\ \Sigma_4 \\ \Sigma_5 \\ \Sigma_6 \end{pmatrix} = \begin{pmatrix} 0.0625 \\ 0.25 \\ 1 \\ 4 \\ 16 \\ 64 \end{pmatrix} \right\}$$

The following statements simulate the 10,000-section time series from the previous model to provide test data for the HMM procedure:

```
%let nSections = 10000;
%let T = 100;
%let seed = 1234;
%let nStates = 6;
%let mu1 = 0;
%let sigma1 = 0.0625;
%let mu2 = 0;
%let sigma2 = 0.25;
%let mu3 = 0;
%let sigma3 = 1;
%let mu4 = 0;
%let sigma4 = 4;
```

```

%let mu5 = 0;
%let sigma5 = 16;
%let mu6 = 0;
%let sigma6 = 64;
%let selfTransProb = 0.95;

%macro createCstsTable(tableName);
  data &tableName.;
    do sec = 1 to &nSections.;
      state = ceil(uniform(&seed.)*&nStates.);
      do t = 1 to &T.;
        %do i = 1 %to &nStates.;
          if(state=&i.) then do;
            y = &mu&i.. + sqrt(&sigma&i..)*normal(&seed.);
            output;
          end;
        %end;
        u = uniform(&seed.);
        if(u>&selfTransProb.) then do;
          u = (u-&selfTransProb.)/(1-&selfTransProb.)*(&nStates.-1);
          state=state + ceil(u);
          if(state>&nStates.) then state = state - &nStates.;
        end;
      end;
    end;
  run;
%mend;

%createCstsTable(cstsDGP);

data mylib.cstsDGP;
  set cstsDGP;
run;

```

The following statements estimate the Gaussian HMM:

```

proc hmm data=mylib.cstsDGP
  outstat=mylib.cstsStat;
  id section=sec time=t;
  model y / method=ml type=gaussian nstate=6;
  optimize ALGORITHM=activeset printlevel=3 printIterFreq=1;
  estimate out=mylib.cstsEst;
  evaluate out=mylib.cstsEval;
  decode out=mylib.cstsDecode;
  filter out=mylib.cstsFilter;
  smooth out=mylib.cstsSmooth;
  forecast out=mylib.cstsForecast;
  score outmodel=mylib.cstsModel;
run;

```

The estimates of the ISPV, TPM, mean parameters, and covariance parameters are shown in [Figure 20.18](#), [Figure 20.19](#), [Figure 20.20](#), and [Figure 20.21](#), respectively. They are all very close to the true parameters that are used in the data generating process.

Figure 20.18 Estimates of the Initial State Probability Vector (ISPV)

Initial State Probabilities	
State	Estimation
1	0.16814
2	0.16340
3	0.16812
4	0.16424
5	0.16594
6	0.17016

Figure 20.19 Estimates of the Transition Probability Matrix (TPM)

Estimated Transition Probability Matrix						
State	1	2	3	4	5	6
1	0.94993	0.00963	0.01013	0.01034	0.00970	0.01028
2	0.00992	0.95071	0.01050	0.00906	0.00942	0.01037
3	0.01040	0.00973	0.94935	0.00950	0.01067	0.01035
4	0.01010	0.00976	0.00973	0.94994	0.01052	0.00995
5	0.01015	0.00939	0.01032	0.01017	0.95012	0.00985
6	0.01002	0.00963	0.01049	0.01009	0.00932	0.95045

Figure 20.20 Estimates of the Mean Parameters

Mu	
State	y
1	0.00025
2	-0.00200
3	-0.00254
4	0.00695
5	-0.02130
6	0.01899

Figure 20.21 Estimates of the Covariance Parameters

Sigma		
State	Variable	y
1	y	0.06257
2	y	0.24731
3	y	0.99708
4	y	3.97252
5	y	15.95629
6	y	64.01087

Gaussian Mixture Hidden Markov Model for Time Series Data and Cross-Sectional Time Series Data

Let $\mathbf{Y}_t = (y_{1t}, \dots, y_{pt})'$, $t = 1, \dots, T$, denote a p -dimensional time series vector of random variables. Conditional on a latent variable $\mathbf{S}_t$, $\mathbf{Y}_t$ follows a Gaussian mixture distribution,

$$\mathbf{Y}_t | \mathbf{S}_t \sim \text{GM}(\{w_{\mathbf{S}_t,1}, \dots, w_{\mathbf{S}_t,M}\}, \{\mu_{\mathbf{S}_t,1}, \dots, \mu_{\mathbf{S}_t,M}\}, \{\Sigma_{\mathbf{S}_t,1}, \dots, \Sigma_{\mathbf{S}_t,M}\})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\text{GM}(\dots)$ represents the Gaussian mixture distribution; M is the number of components at each state; and the $w_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_{\mathbf{S}_t,j} \geq 0$, $j = 1, \dots, M$, and $\sum_{j=1}^M w_{\mathbf{S}_t,j} = 1$). The $\mu_{\mathbf{S}_t,j}$ and $\Sigma_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are the mean and covariance parameters for the j th Gaussian component at state $\mathbf{S}_t$, and their values depend on the variable $\mathbf{S}_t$. The state variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the variable $\mathbf{S}_t$ is unobservable and follows a Markov chain and because $\mathbf{Y}_t$ follows a Gaussian mixture distribution conditional on $\mathbf{S}_t$, the model is called the Gaussian mixture hidden Markov model (GM HMM). The GM HMM is sometimes described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{w_{ij}, \mu_{ij}, \Sigma_{ij}, i = 1, \dots, K, j = 1, \dots, M\}$, and it consists of parameters that define the state-dependent distribution of observable variables.

The GM HMM for cross-sectional time series data is similar to the extension from Gaussian HMM to Gaussian HMM for cross-sectional time series data; for more information, see the section “[Gaussian Hidden Markov Model for Cross-Sectional Time Series Data](#)” on page 1198.

Consider a bivariate bi-component GM HMM that has the following parameter values. The ISPV and TPM are

$$\pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.05 \\ 0.15 & 0.85 \end{pmatrix}$$

At state 1, the two components have weights

$$c_{11} = 0.6, c_{12} = 0.4$$

and the mean and covariance parameters for two components are

$$\mu_{11} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma_{11} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_{12} = \begin{pmatrix} -3 \\ -3 \end{pmatrix}, \Sigma_{12} = \begin{pmatrix} 1.69 & 1.00 \\ 1.00 & 1.69 \end{pmatrix}$$

At state 2, the two components have weights

$$c_{21} = 0.7, c_{22} = 0.3$$

and the mean and covariance parameters for two components are

$$\mu_{21} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \Sigma_{22} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_{22} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \Sigma_{22} = \begin{pmatrix} 1.69 & 1.00 \\ 1.00 & 1.69 \end{pmatrix}$$

The following statements simulate the bivariate vector time series from the previous model to provide test data for the HMM procedure:

```
%let a11 = 0.98;
%let a22 = 0.95;
%let c1_1 = 0.6;
%let c1_2 = 0.4;
%let c2_1 = 0.7;
%let c2_2 = 0.3;
%let mu1_1_1 = 1;
%let mu1_1_2 = 1;
%let mu1_2_1 = -3;
%let mu1_2_2 = -3;
%let sigma1_1_11 = 1;
%let sigma1_1_21 = 0;
%let sigma1_1_22 = 1;
%let sigma1_2_11 = 1.69;
%let sigma1_2_21 = 1;
%let sigma1_2_22 = 1.69;
%let mu2_1_1 = -1;
%let mu2_1_2 = -1;
%let mu2_2_1 = 3;
%let mu2_2_2 = 3;
%let sigma2_1_11 = 1;
%let sigma2_1_21 = 0;
%let sigma2_1_22 = 1;
%let sigma2_2_11 = 1.69;
%let sigma2_2_21 = 1;
%let sigma2_2_22 = 1.69;
%let seed = 1234;
%let T = 200000;

data gmDGP;
    retain cd1_1_11 cd1_1_21 cd1_1_22
           cd1_2_11 cd1_2_21 cd1_2_22
           cd2_1_11 cd2_1_21 cd2_1_22
           cd2_2_11 cd2_2_21 cd2_2_22;
```

```

do t = 1 to &T.;
  if(t=1) then do;
    * initial probability distribution;
    p = (1-&a22.)/((1-&a11.)+(1-&a22.));;
    * Cholesky decomposition;
    cd1_1_11 = sqrt(&sigma1_1_11.);
    cd1_1_21 = &sigma1_1_21./cd1_1_11;
    cd1_1_22 = sqrt(&sigma1_1_22.-cd1_1_21*cd1_1_21);
    cd1_2_11 = sqrt(&sigma1_2_11.);
    cd1_2_21 = &sigma1_2_21./cd1_2_11;
    cd1_2_22 = sqrt(&sigma1_2_22.-cd1_2_21*cd1_2_21);
    cd2_1_11 = sqrt(&sigma2_1_11.);
    cd2_1_21 = &sigma2_1_21./cd2_1_11;
    cd2_1_22 = sqrt(&sigma2_1_22.-cd2_1_21*cd2_1_21);
    cd2_2_11 = sqrt(&sigma2_2_11.);
    cd2_2_21 = &sigma2_2_21./cd2_2_11;
    cd2_2_22 = sqrt(&sigma2_2_22.-cd2_2_21*cd2_2_21);
  end;
  else do;
    * transition probability matrix;
    if(lags=1) then p = &a11.;
    else p = 1-&a22.;
  end;
  u = uniform(&seed.);
  if(u<=p) then s=1;
  else s = 2;
  e1 = normal(&seed.); e2 = normal(&seed.);
  if(s=1) then do;
    * choose component;
    u = uniform(&seed.);
    if(u<=&c1_1.) then do;
      * (x,y) ~ N(mu, Sigma) at state 1, component 1;
      x = &mu1_1_1. + cd1_1_11*e1;
      y = &mu1_1_2. + cd1_1_21*e1+cd1_1_22*e2;
    end;
    else do;
      * (x,y) ~ N(mu, Sigma) at state 1, component 2;
      x = &mu1_2_1. + cd1_2_11*e1;
      y = &mu1_2_2. + cd1_2_21*e1+cd1_2_22*e2;
    end;
  end;
  else do;
    * choose component;
    u = uniform(&seed.);
    if(u<=&c2_1.) then do;
      * (x,y) ~ N(mu, Sigma) at state 2, component 1;
      x = &mu2_1_1. + cd2_1_11*e1;
      y = &mu2_1_2. + cd2_1_21*e1+cd2_1_22*e2;
    end;
    else do;
      * (x,y) ~ N(mu, Sigma) at state 2, component 2;
      x = &mu2_2_1. + cd2_2_11*e1;
      y = &mu2_2_2. + cd2_2_21*e1+cd2_2_22*e2;
    end;
  end;
end;

```

```

        end;
        output;
        lags = s;
    end;
run;

data gmTrain;
    set gmDGP(where=(t<=&T./2));
    keep t x y;
run;

data gmTest;
    set gmDGP(where=(t>&T./2));
    keep t x y;
run;

```

In general, the data generating process is unknown. The data tables gmTrain and gmTest can be seen, but the data table gmDGP cannot be seen.

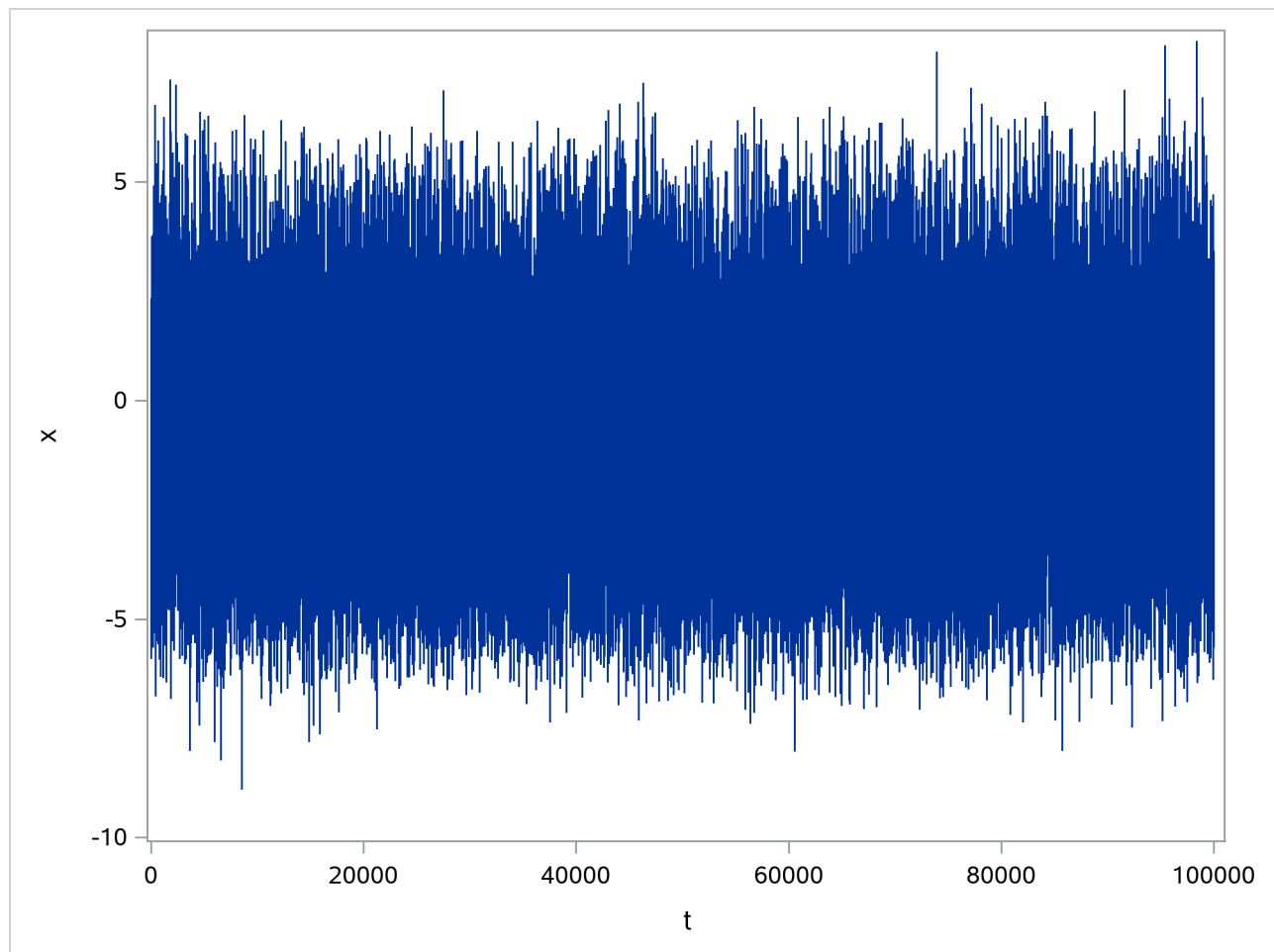
First, assume that the following statements plot the simulated series x that is contained in the data table gmTrain:

```

proc sgplot data=gmTrain;
    series x=t y=x;
run;

```

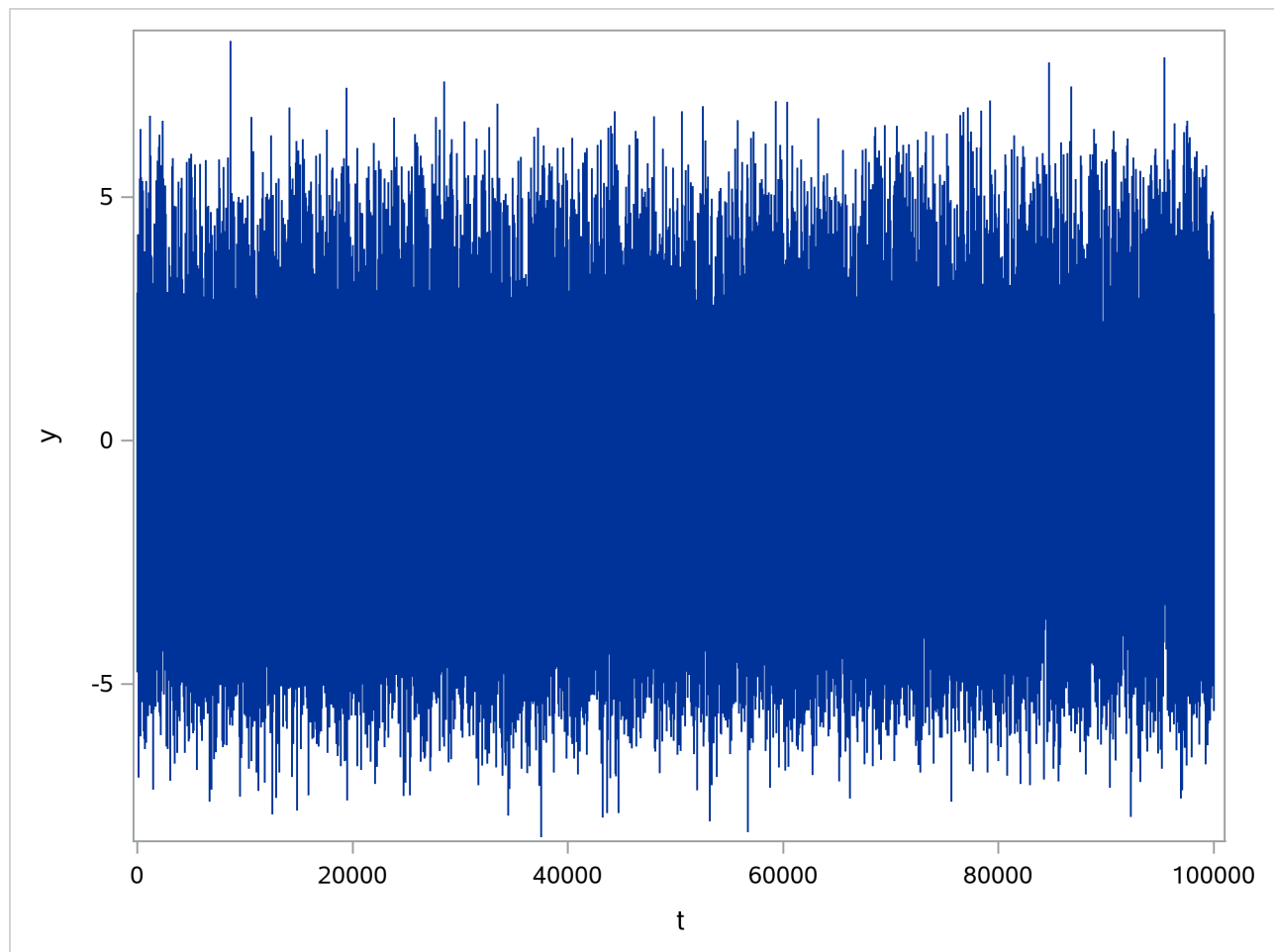
Figure 20.22 shows the plot of x , whose points seem to fall randomly around 0.

Figure 20.22 Plot of Time Series x 

The following statements plot the simulated series y that is contained in the data set `gmTrain`:

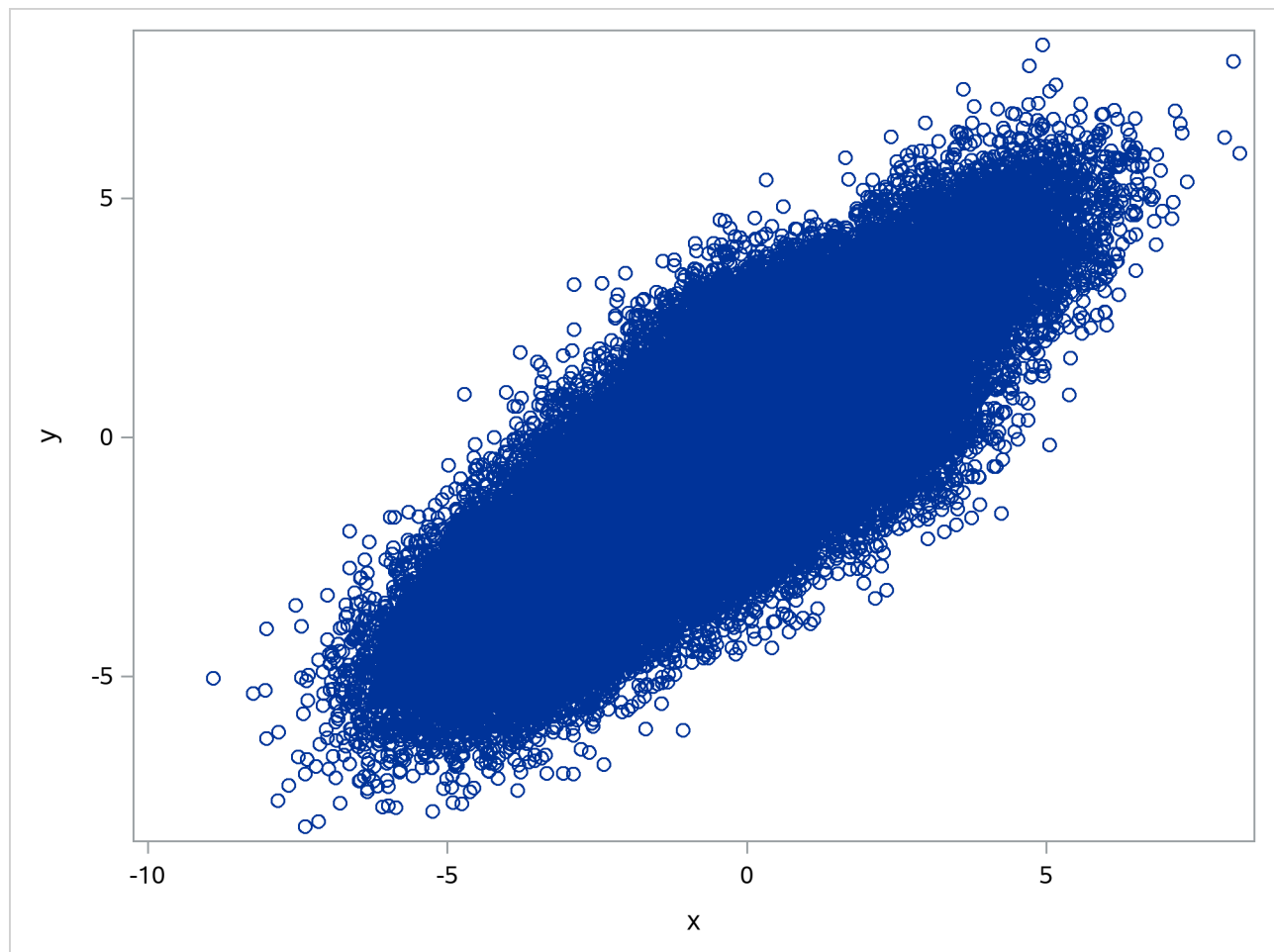
```
proc sgplot data=gmTrain;  
  series x=t y=y;  
run;
```

Figure 20.23 shows the plot of y , whose points also seem to fall randomly around 0.

Figure 20.23 Plot of Time Series y 

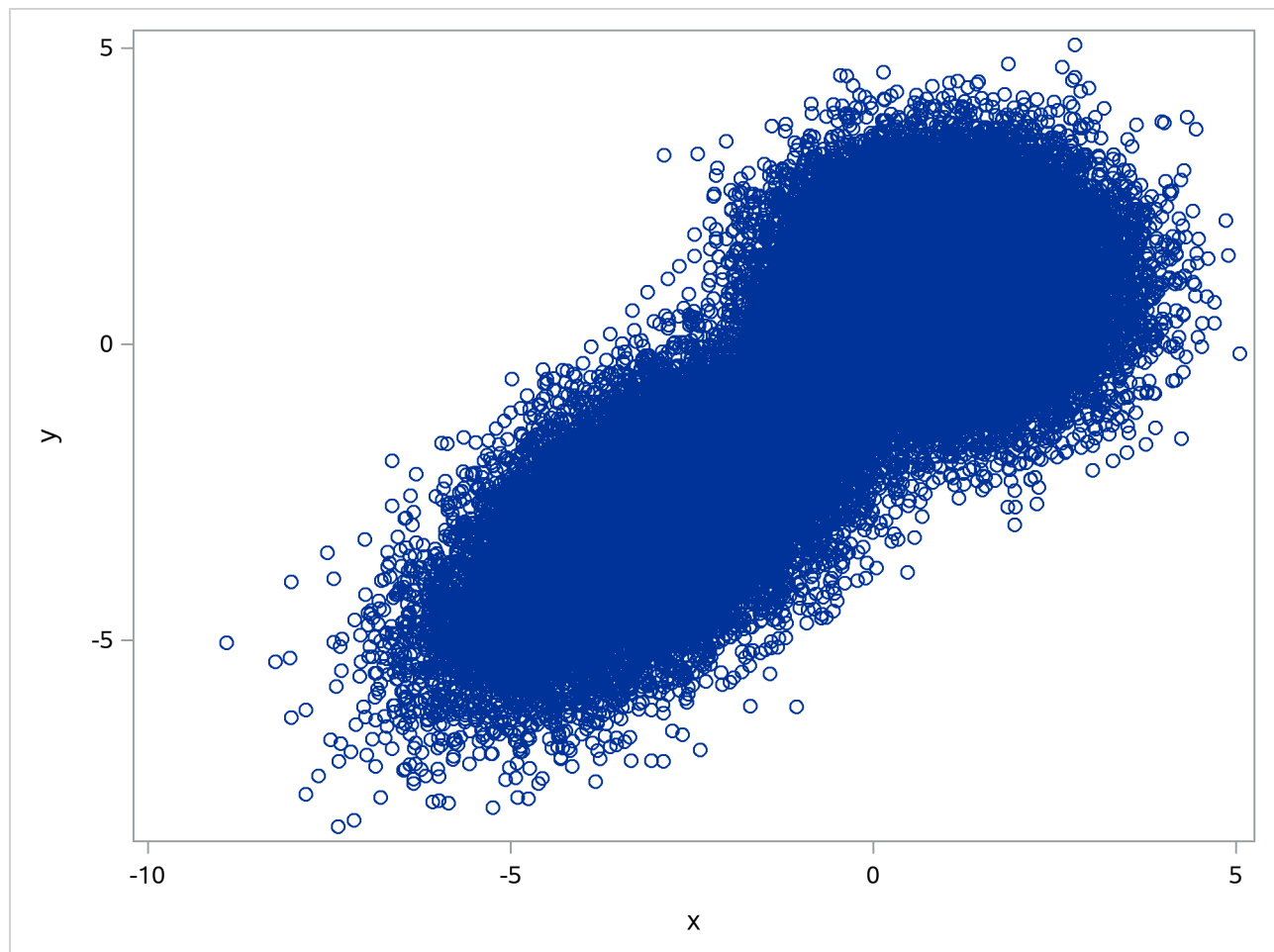
Ignoring the sequential information, the following statements plot the observed points (x, y) that are contained in the data table gmTrain:

```
proc sgplot data=gmTrain;  
    scatter y=y x=x;  
run;
```

Figure 20.24 Unclassified Observations

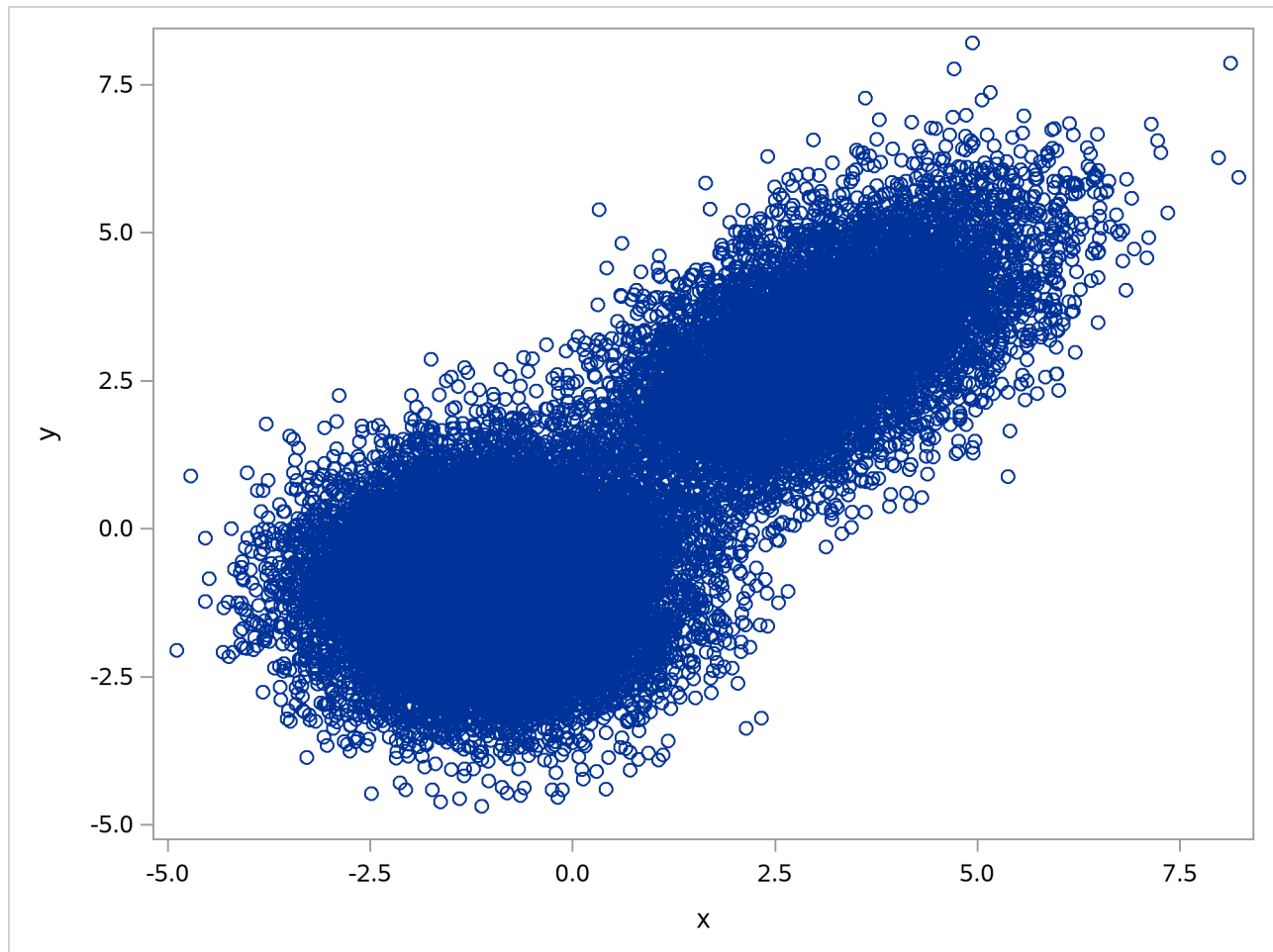
As shown in [Figure 20.24](#), it seems that all points come from one bivariate Gaussian distribution that has a high positive correlation. However, according to the true DGP, some points are from state 1, in which there are two clusters, as shown in [Figure 20.25](#):

```
proc sgplot data=gmdgp (where=(t<=&t./2 and s=1));
  scatter y=y x=x;
run;
```

Figure 20.25 Observations from State 1

Other points are from state 2 in which there are two clusters, as shown in [Figure 20.26](#):

```
proc sgplot data=gmdGP (where=(t<=&T./2 and s=2));
  scatter y=y x=x;
run;
```

Figure 20.26 Observations from State 2

Without considering the sequential order, it is impossible to distinguish the two states.

Before you estimate any model, you should make a copy of the data in the library, as in the following DATA steps:

```
data mylib.gmTrain; set gmTrain; run;  
data mylib.gmTest; set gmTest; run;
```

The following statements estimate the bivariate bi-component GM HMM. The type of model is specified in the `TYPE=` option in the `MODEL` statement. The number of states is specified in the `NSTATE=` option in the `MODEL` statement. The number of components for each state is specified in the `NCOMPONENT=` option in the `MODEL` statement. The `OUTMODEL=` option in the `SCORE` statement outputs the model information for future scoring on new data table. The `LABELSWITCH=(SORT=NONE)` option in the `PROC HMM` statement suppresses the relabeling of states and components; for more information about the label switching problem, see the section “[Label Switching Problem](#)” on page 1321.

```
proc hmm data=mylib.gmTrain labelSwitch=(sort=none);
  id time=t;
  model x y / type=gaussianMixture nstate=2 ncomponent=2;
  optimize printLevel=3 printIterFreq=1;
  score outmodel=mylib.gmModel;
run;
```

These statements assume that your engine libref is named `mylib`, but you can substitute any appropriately defined engine libref.

The number of observations and model information are shown in [Figure 20.27](#).

Figure 20.27 Number of Observations and Model Information

Observation Information	
Number of Observations	100000
Number of Missing Observations	0

Model Information	
Type of Model	Gaussian Mixture HMM
Stationary	Yes
Number of States	2
Number of Components for a State	2
Number of Dependent Variables	2

The default method, which is the maximum likelihood method, is applied; this is a nonlinear optimization problem. The initial parameter values and the objective function value at the start of the optimization are shown in [Figure 20.28](#).

Figure 20.28 Initial Parameter Values and Objective Function Value

Optimization Start	
Parameter	Estimate
TPM1_1	0.500000
TPM1_2	0.500000
TPM2_1	0.500000
TPM2_2	0.500000
MCP1_1	0.500000
MCP1_2	0.500000
MCP2_1	0.500000
MCP2_2	0.500000
MU1_1_1	0.416654
MU1_1_2	0.961871
MU1_2_1	-3.422684
MU1_2_2	-3.125473
MU2_1_1	2.963828
MU2_1_2	1.590644
MU2_2_1	-0.815362
MU2_2_2	-2.039325
SIGMA1_1_1_1	1.354409
SIGMA1_1_2_1	0.762768
SIGMA1_1_2_2	1.393576
SIGMA1_2_1_1	0.961355
SIGMA1_2_2_1	0.661717
SIGMA1_2_2_2	1.581451
SIGMA2_1_1_1	1.025889
SIGMA2_1_2_1	1.399630
SIGMA2_1_2_2	2.884377
SIGMA2_2_1_1	0.749535
SIGMA2_2_2_1	0.265184
SIGMA2_2_2_2	0.633097

Initial Value of Objective Function		-401529.3793
-------------------------------------	--	--------------

The details of the optimization algorithm are shown in [Figure 20.29](#).

Figure 20.29 Optimization Algorithm Details

Algorithm Settings	
Maximum Number of Iterations	128
Tolerance for Optimality Error	1E-6
Tolerance for Infeasibility	1E-6
Maximum Allowed Time (seconds)	1.797693E308
Maximum Magnitude of Objective Function Value	1E20
Random Seed	1
Solution Type	1
Multi-Start Globalization Scheme	0
Multi-Start Globalization Bound Range	2

The iterations of the optimization process are shown in Figure 20.30.

Figure 20.30 Iterations of the Optimization Process

Iteration	Objective Function	Iteration History		Function Calls	Gradient Calls	Hessian Calls
		Optimality Error	Infeasibility			
1	-401529.3793	2887.16466	0.00000	2	1	0
2	-389554.1158	2133.36152	0.00000	10	9	1
3	-386179.6546	1199.52355	0.00000	19	18	2
4	-371985.2011	848.19186	0.00000	29	28	3
5	-367516.0600	249.88933	0.00000	38	37	4
6	-366322.8416	161.33937	0.00000	49	48	5
7	-365982.2352	46.74362	0.00000	59	58	6
8	-365945.5123	5.29758	0.00000	69	68	7
9	-365945.0948	0.06672	0.00000	79	78	8
10	-365945.0946	0.00000	0.00000	89	88	9

The final parameter values and the objective function value at the end of the optimization are shown in Figure 20.31.

Figure 20.31 Final Parameter Values and Objective Function Value

Optimization Results	
Parameter	Estimate
TPM1_1	0.979636
TPM1_2	0.020364
TPM2_1	0.049729
TPM2_2	0.950271
MCP1_1	0.600332
MCP1_2	0.399668
MCP2_1	0.297811
MCP2_2	0.702189
MU1_1_1	0.993524
MU1_1_2	0.996707
MU1_2_1	-3.003906
MU1_2_2	-3.005379
MU2_1_1	3.005306
MU2_1_2	3.009691
MU2_2_1	-0.997604
MU2_2_2	-1.005076
SIGMA1_1_1_1	1.000305
SIGMA1_1_2_1	0.007588
SIGMA1_1_2_2	1.003672
SIGMA1_2_1_1	1.704270
SIGMA1_2_2_1	1.003413
SIGMA1_2_2_2	1.688212
SIGMA2_1_1_1	1.712932
SIGMA2_1_2_1	1.014865
SIGMA2_1_2_2	1.704181
SIGMA2_2_1_1	1.015895
SIGMA2_2_2_1	0.012899
SIGMA2_2_2_2	1.006142
Final Value of Objective Function	
-365945.0946	

The estimates of the initial state probability vector (ISPV) are shown in [Figure 20.32](#).

Figure 20.32 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	Estimation
1	0.70947
2	0.29053

The estimates of the transition probability matrix (TPM) are shown in [Figure 20.33](#).

Figure 20.33 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	1	2
1	0.97964	0.02036
2	0.04973	0.95027

The estimates of the mixture component probabilities (MCP) are shown in [Figure 20.34](#).

Figure 20.34 Estimates of Mixture Component Probabilities

Estimated Mixture Component Probabilities		
Component		
State	1	2
1	0.60033	0.39967
2	0.29781	0.70219

The estimates of the mean vector of the Gaussian distribution for each component at each state are shown in [Figure 20.35](#).

Figure 20.35 Estimate of Mean Vector for Each Component at Each State

Mu				
State	Component	x	y	
1	1	0.99352	0.99671	
	2	-3.00391	-3.00538	
2	1	3.00531	3.00969	
	2	-0.99760	-1.00508	

The estimates of the covariance matrix of the Gaussian distribution for each component at each state are shown in [Figure 20.36](#).

Figure 20.36 Estimate of Covariance Matrix for Each Component at Each State

Sigma				
State	Component	Variable	x	y
1	1	x	1.00031	0.00759
		y	0.00759	1.00367
	2	x	1.70427	1.00341
		y	1.00341	1.68821
2	1	x	1.71293	1.01486
		y	1.01486	1.70418
	2	x	1.01589	0.01290
		y	0.01290	1.00614

The estimates of all parameters are shown in [Figure 20.37](#), which displays columns for the parameter name, estimate value, standard error, t value, and p -value. The state labels (“1” and “2” in this example) and the

component labels (“1” and “2” in this example) might exchange randomly, compared to the true DGP. In the preceding estimation, the state labels are the same as the labels in the DGP. The component labels for state 1 are also the same as the ones in the DGP, but the component labels for state 2 do exchange! Compared to the parameter values in the DGP, the estimates are very accurate.

Figure 20.37 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.979636	0.000600	1632.32	<.0001
TPM1_2	0.020364	0.000600	33.93	<.0001
TPM2_1	0.049729	0.001422	34.98	<.0001
TPM2_2	0.950271	0.001422	668.37	<.0001
MCP1_1	0.600332	0.001987	302.17	<.0001
MCP1_2	0.399668	0.001987	201.17	<.0001
MCP2_1	0.297811	0.002966	100.42	<.0001
MCP2_2	0.702189	0.002966	236.78	<.0001
MU1_1_1	0.993524	0.005202	191.00	<.0001
MU1_1_2	0.996707	0.005225	190.77	<.0001
MU1_2_1	-3.003906	0.009259	-324.43	<.0001
MU1_2_2	-3.005379	0.009199	-326.71	<.0001
MU2_1_1	3.005306	0.017586	170.89	<.0001
MU2_1_2	3.009691	0.017492	172.06	<.0001
MU2_2_1	-0.997604	0.007672	-130.04	<.0001
MU2_2_2	-1.005076	0.007645	-131.47	<.0001
SIGMA1_1_1_1	1.000305	0.007559	132.33	<.0001
SIGMA1_1_2_1	0.007588	0.005615	1.35	0.1765
SIGMA1_1_2_2	1.003672	0.007625	131.63	<.0001
SIGMA1_2_1_1	1.704270	0.018040	94.47	<.0001
SIGMA1_2_2_1	1.003413	0.015372	65.27	<.0001
SIGMA1_2_2_2	1.688212	0.017808	94.80	<.0001
SIGMA2_1_1_1	1.712932	0.033960	50.44	<.0001
SIGMA2_1_2_1	1.014865	0.028939	35.07	<.0001
SIGMA2_1_2_2	1.704181	0.033648	50.65	<.0001
SIGMA2_2_1_1	1.015895	0.011295	89.94	<.0001
SIGMA2_2_2_1	0.012899	0.008289	1.56	0.1197
SIGMA2_2_2_2	1.006142	0.011248	89.45	<.0001

The log likelihood and information criteria are shown in [Figure 20.38](#).

Figure 20.38 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-365945.0946
AIC	731942.18929
AICC	731942.20333
BIC	732189.52535
HQC	732017.24975

The estimates from the data table `gmTrain` are used to score the testing data table `gmTest` to check the predictability in the following statements:

```
proc hmm data=mylib.gmTest;
  score inmodel=mylib.gmModel;
  decode out=mylib.gmDecode;
run;
```

The accuracy of classification through the GM HMM is calculated by the following statements:

```
data gmHmmCheck;
  merge gmDGP(in=a where=(t>&T./2)) mylib.gmDecode(in=b);
  by t;
  if(s=state) then do; correct=1; ds = s; end;
  else do; correct=0; ds = -s; end;
  if(a=b);
  keep t x y s correct ds;
run;

%let qFlipState = -1;
data gmHmmAccuracy;
  set gmHmmCheck;
  retain count 0 correctCount 0;
  count = count + 1;
  correctCount = correctCount + correct;
  if(count>&T./2-0.5) then do;
    accuracy = correctCount / count;
    if(accuracy<0.5) then call symputx("qFlipState",1,'G');
    else call symputx("qFlipState",0,'G');
    if(accuracy<0.5) then accuracy = 1 - accuracy;
    output;
  end;
  keep accuracy;
run;

data gmHmmCheck;
  set gmHmmCheck;
  if(&qFlipState.=1) then do;
    ds = -ds;
    correct = 1 - correct;
  end;
run;

proc print data = gmHmmAccuracy noobs; run;
```

The accuracy of the GM HMM, which is shown in [Figure 20.39](#), is about 98%. This means that the GM HMM successfully takes advantage of the sequential information.

Figure 20.39 Accuracy of Decoding by the GM HMM

accuracy
0.97835

Now, consider the GM HMM for cross-sectional time series data. Each section's data follow the same GM HMM as described previously. The following statements generate the data table for analysis. There are 200 sections in the data table, and each section of time series consist of 1,000 observations.

```
%let a11 = 0.98;
%let a22 = 0.95;
%let c1_1 = 0.6;
%let c1_2 = 0.4;
%let c2_1 = 0.7;
%let c2_2 = 0.3;
%let mu1_1_1 = 1;
%let mu1_1_2 = 1;
%let mu1_2_1 = -3;
%let mu1_2_2 = -3;
%let sigma1_1_11 = 1;
%let sigma1_1_21 = 0;
%let sigma1_1_22 = 1;
%let sigma1_2_11 = 1.69;
%let sigma1_2_21 = 1;
%let sigma1_2_22 = 1.69;
%let mu2_1_1 = -1;
%let mu2_1_2 = -1;
%let mu2_2_1 = 3;
%let mu2_2_2 = 3;
%let sigma2_1_11 = 1;
%let sigma2_1_21 = 0;
%let sigma2_1_22 = 1;
%let sigma2_2_11 = 1.69;
%let sigma2_2_21 = 1;
%let sigma2_2_22 = 1.69;
%let seed = 1234;
%let T = 200000;
%let nSections = 200;

data gmDGP;
  retain cd1_1_11 cd1_1_21 cd1_1_22
         cd1_2_11 cd1_2_21 cd1_2_22
         cd2_1_11 cd2_1_21 cd2_1_22
         cd2_2_11 cd2_2_21 cd2_2_22;
  do t = 1 to &T.;
    sec = ceil(t*&nSections./&T.);
    if(t=1) then do;
      * initial probability distribution;
      p = (1-&a22.)/((1-&a11.)+(1-&a22.));;
      * Cholesky decomposition;
      cd1_1_11 = sqrt(&sigma1_1_11.);
      cd1_1_21 = &sigma1_1_21./cd1_1_11;
      cd1_1_22 = sqrt(&sigma1_1_22.-cd1_1_21*cd1_1_21);
      cd1_2_11 = sqrt(&sigma1_2_11.);
      cd1_2_21 = &sigma1_2_21./cd1_2_11;
      cd1_2_22 = sqrt(&sigma1_2_22.-cd1_2_21*cd1_2_21);
      cd2_1_11 = sqrt(&sigma2_1_11.);
      cd2_1_21 = &sigma2_1_21./cd2_1_11;
      cd2_1_22 = sqrt(&sigma2_1_22.-cd2_1_21*cd2_1_21);
      cd2_2_11 = sqrt(&sigma2_2_11.);
      cd2_2_21 = &sigma2_2_21./cd2_2_11;
```

```

        cd2_2_22 = sqrt(&sigma2_2_22.-cd2_2_21*cd2_2_21);
    end;
    else do;
        * transition probability matrix;
        if(lags=1) then p = &a11.;
        else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e1 = normal(&seed.); e2 = normal(&seed.);
    if(s=1) then do;
        * choose component;
        u = uniform(&seed.);
        if(u<=&c1_1.) then do;
            * (x,y) ~ N(mu, Sigma) at state 1, component 1;
            x = &mu1_1_1. + cd1_1_11*e1;
            y = &mu1_1_2. + cd1_1_21*e1+cd1_1_22*e2;
        end;
        else do;
            * (x,y) ~ N(mu, Sigma) at state 1, component 2;
            x = &mu1_2_1. + cd1_2_11*e1;
            y = &mu1_2_2. + cd1_2_21*e1+cd1_2_22*e2;
        end;
    end;
    else do;
        * choose component;
        u = uniform(&seed.);
        if(u<=&c2_1.) then do;
            * (x,y) ~ N(mu, Sigma) at state 2, component 1;
            x = &mu2_1_1. + cd2_1_11*e1;
            y = &mu2_1_2. + cd2_1_21*e1+cd2_1_22*e2;
        end;
        else do;
            * (x,y) ~ N(mu, Sigma) at state 2, component 2;
            x = &mu2_2_1. + cd2_2_11*e1;
            y = &mu2_2_2. + cd2_2_21*e1+cd2_2_22*e2;
        end;
    end;
    output;
    lags = s;
end;
run;

data gmTrain;
    set gmDGP(where=(sec<=&nSections./2));
    keep sec t x y;
run;

data gmTest;
    set gmDGP(where=(sec>&nSections./2));
    keep sec t x y;
run;

```

The following statements estimate the GM HMM for the cross-sectional time series data after making the copies of the data tables in the library. The SECTION= option in the ID statement requests that the cross-sectional time series data be analyzed.

```
data mylib.gmTrain; set gmTrain; run;
data mylib.gmTest; set gmTest; run;

proc hmm data=mylib.gmTrain labelSwitch=(sort=none);
  id time=t section=sec;
  model x y / type=gaussianMixture nstate=2 ncomponent=2;
  optimize printLevel=3 printIterFreq=1;
  score outmodel=mylib.gmModel;
  filter out=mylib.gmFilter;
  evaluate out=mylib.gmEval;
run;
```

The model information is shown in [Figure 20.40](#). There are 100 sections in the data.

Figure 20.40 Model Information

Model Information	
Type of Model	Gaussian Mixture HMM
Stationary	Yes
Number of States	2
Number of Components for a State	2
Number of Dependent Variables	2
Number of Sections	100

The estimates of all parameters are shown in [Figure 20.41](#); the estimates are very close to the parameter values in the DGP.

Figure 20.41 Parameter Estimates

Parameter Estimates				
Parameter	Standard			
	Estimate	Error	t Value	Pr > t
TPM1_1	0.979697	0.000598	1639.23	<.0001
TPM1_2	0.020303	0.000598	33.97	<.0001
TPM2_1	0.049991	0.001419	35.23	<.0001
TPM2_2	0.950009	0.001419	669.51	<.0001
MCP1_1	0.600330	0.001987	302.13	<.0001
MCP1_2	0.399670	0.001987	201.15	<.0001
MCP2_1	0.297864	0.002966	100.41	<.0001
MCP2_2	0.702136	0.002966	236.70	<.0001
MU1_1_1	0.993469	0.005202	190.97	<.0001
MU1_1_2	0.996750	0.005224	190.80	<.0001
MU1_2_1	-3.004280	0.009259	-324.45	<.0001
MU1_2_2	-3.005705	0.009202	-326.64	<.0001
MU2_1_1	3.004517	0.017596	170.75	<.0001
MU2_1_2	3.008912	0.017510	171.84	<.0001
MU2_2_1	-0.997409	0.007674	-129.97	<.0001
MU2_2_2	-1.005153	0.007644	-131.50	<.0001
SIGMA1_1_1_1	1.000332	0.007560	132.32	<.0001
SIGMA1_1_2_1	0.007321	0.005614	1.30	0.1922
SIGMA1_1_2_2	1.003283	0.007624	131.60	<.0001
SIGMA1_2_1_1	1.703717	0.018034	94.47	<.0001
SIGMA1_2_2_1	1.002882	0.015368	65.26	<.0001
SIGMA1_2_2_2	1.687759	0.017806	94.78	<.0001
SIGMA2_1_1_1	1.713186	0.033969	50.43	<.0001
SIGMA2_1_2_1	1.015588	0.028964	35.06	<.0001
SIGMA2_1_2_2	1.705412	0.033689	50.62	<.0001
SIGMA2_2_1_1	1.015835	0.011301	89.89	<.0001
SIGMA2_2_2_1	0.012931	0.008291	1.56	0.1188
SIGMA2_2_2_2	1.005816	0.011243	89.46	<.0001

You can apply the estimates from the data table gmTrain to the testing data table gmTest to check the predictability in the following statements:

```
proc hmm data=mylib.gmTest;
  score inmodel=mylib.gmModel;
  decode out=mylib.gmDecode;
  filter out=mylib.gmFilter2;
  evaluate out=mylib.gmEval2;
run;

data gmHmmCheck;
  merge gmDGP(in=a where=(sec>&nSections./2)) mylib.gmDecode(in=b);
  by t;
  if(s=state) then do; correct=1; ds = s; end;
  else do; correct=0; ds = -s; end;
  if(a=b);
  keep sec t x y s correct ds;
```

```

run;

%let qFlipState = -1;
data gmHmmAccuracy;
  set gmHmmCheck;
  retain count 0 correctCount 0;
  count = count + 1;
  correctCount = correctCount + correct;
  if(count>&T./2-0.5) then do;
    accuracy = correctCount / count;
    if(accuracy<0.5) then call symputx("qFlipState",1,'G');
    else call symputx("qFlipState",0,'G');
    if(accuracy<0.5) then accuracy = 1 - accuracy;
    output;
  end;
  keep accuracy;
run;

data gmHmmCheck;
  set gmHmmCheck;
  if(&qFlipState.=1) then do;
    ds = -ds;
    correct = 1 - correct;
  end;
run;

proc print data = gmHmmAccuracy noobs; run;

```

The accuracy of the GM HMM for cross-sectional time series data, which is shown in [Figure 20.42](#), is also around 98%.

Figure 20.42 Accuracy of Decoding by the GM HMM for Cross-Sectional Time Series Data

accuracy
0.97827

Regime-Switching Regression Model

Let $\mathbf{Y}_t = (y_{1t}, \dots, y_{pt})'$, $t = 1, \dots, T$, denote a p -dimensional time series vector of random variables. The $\mathbf{Y}_t$ can be modeled in the regression

$$\mathbf{Y}_t = \beta_{\mathbf{S}_t} z_t + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, the regressor z_t is observable, the latent variable $\mathbf{S}_t$ is the so-called state, and $\beta_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the state $\mathbf{S}_t$. The variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(\cdot|\cdot)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the regression parameters depend on the state, the $\mathbf{Y}_t$ follows different regressions in different regimes; hence, this type of model is called the regime-switching regression model.

Consider a univariate regime-switching regression model that has two regimes and two regressors (an intercept and an exogenous variable):

$$y_t = \begin{cases} x_t + \varepsilon_t, \varepsilon_t \sim N(0, 2.56) & \text{if } \mathbf{S}_t = 1 \\ 4 + 1.5x_t + \varepsilon_t, \varepsilon_t \sim N(0, 4) & \text{if } \mathbf{S}_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$$

The following statements simulate the time series from the previous model to provide test data for the HMM procedure:

```
%let x_lb = -8;
%let x_ub = 0;
%let pi1 = 0.5;
%let a11 = 0.95;
%let a22 = 0.95;
%let const1_1 = 0;
%let x11_0_1_1 = 1;
%let cov1_11 = 2.56;
%let const2_1 = 4;
%let x12_0_1_1 = 1.5;
%let cov2_11 = 4;
%let T = 4000;
%let seed = 1234;

data rsregdgp;
  retain cd1_11 cd2_11;
  do t = 1 to &T.;
    if(t=1) then do;
      /* initial probability distribution */
      p = &pi1.;
      /* Cholesky decomposition of COV1 */
```

```

        cd1_11 = sqrt(&cov1_11.);
        /* Cholesky decomposition of COV2 */
        cd2_11 = sqrt(&cov2_11.);
    end;
    else do;
        /* transition probability matrix */
        if(lags=1) then p = &a11.;
        else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    x = &x_lb. + (&x_ub. - &x_lb.)*uniform(&seed.);
    e = normal(&seed.);
    if(s=1) then do;
        /* y ~ N(beta1_0+beta1_1*x, Sigma1) at state 1 */
        y = &const1_1. + &x11_0_1_1 * x + cd1_11*e;
    end;
    else do;
        /* y ~ N(beta2_0+beta2_1*x, Sigma2) at state 2 */
        y = &const2_1. + &x12_0_1_1 * x + cd2_11*e;
    end;
    output;
    lags = s;
end;
run;

data rsreg;
    set rsregdgp;
    keep t y x;
run;

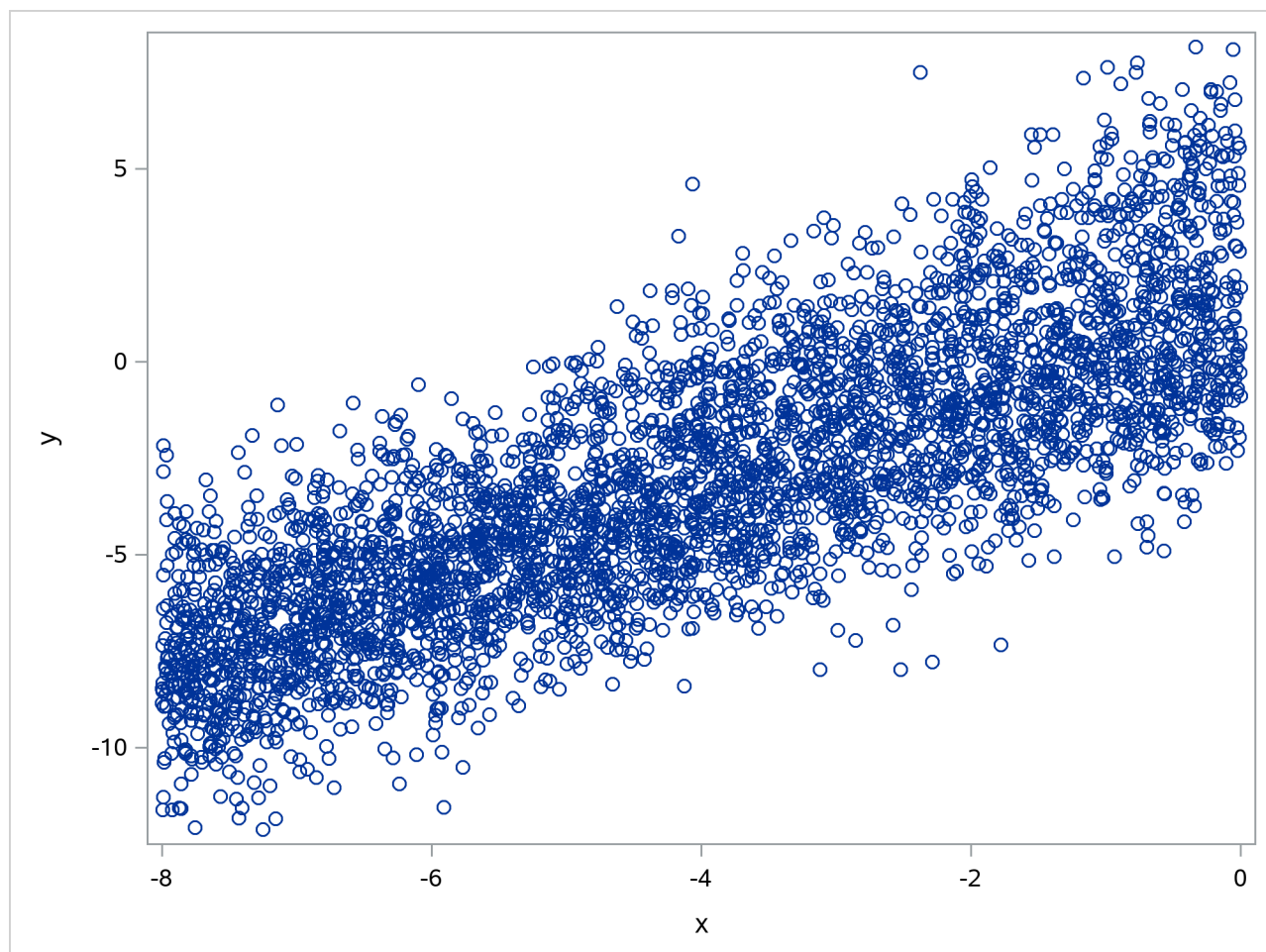
```

The following statements create the scatter plot in [Figure 20.43](#):

```

proc sgplot data=rsreg;
    scatter y=y x=x;
run;

```

Figure 20.43 Scatter Plot of Data Points

As shown in Figure 20.43, when the time dependence is ignored, it is difficult—if not impossible—to distinguish the two regimes.

The following code uploads the data table and estimates the regime-switching regression model:

```
data mylib.rsreg; set rsreg; run;
proc hmm data=mylib.rsreg;
  id time=t;
  model y = x / type=reg nstate=2 method=ml;
  optimize algorithm=activeset maxiter=256 printLevel=3 printIterFreq=1;
  learn out=mylib.mylearn;
  filter out=mylib.myfilter;
  smooth out=mylib.mysmooth;
  decode out=mylib.mydecode;
  evaluate out=mylib.myeval;
run;
```

These statements assume that your engine libref is named mylib, but you can substitute any appropriately defined engine libref.

The estimates of all parameters are shown in [Figure 20.44](#), which displays columns for the parameter name, estimate value, standard error, t value, and p -value. The parameter estimates are very close to the true parameter values that are used in the data generating process.

Figure 20.44 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.950779	0.006259	151.91	<.0001
TPM1_2	0.049221	0.006259	7.86	<.0001
TPM2_1	0.057042	0.007335	7.78	<.0001
TPM2_2	0.942958	0.007335	128.56	<.0001
CONST1_1	-0.058860	0.079933	-0.74	0.4616
CONST2_1	4.105147	0.113280	36.24	<.0001
XL1_0_1_1	0.986513	0.016304	60.51	<.0001
XL2_0_1_1	1.513899	0.022254	68.03	<.0001
COV1_1_1	2.563899	0.094248	27.20	<.0001
COV2_1_1	3.839914	0.143172	26.82	<.0001

The accuracy of classification through the regime-switching regression model is calculated by the following statements:

```
data rsregCheck;
    merge rsregdgp(in=a) mylib.mydecode(in=b);
    by t;
    if(s=state) then do; correct=1; ds = s; end;
    else do; correct=0; ds = -s; end;
    if(a=b);
    keep t x y s correct ds;
run;

%let qFlipState = -1;
data rsregAccuracy;
    set rsregCheck;
    retain count 0 correctCount 0;
    count = count + 1;
    correctCount = correctCount + correct;
    if(count>&T.-0.5) then do;
        accuracy = correctCount / count;
        if(accuracy<0.5) then call symputx("qFlipState",1,'G');
        else call symputx("qFlipState",0,'G');
        if(accuracy<0.5) then accuracy = 1 - accuracy;
        output;
    end;
    keep accuracy;
run;

data rsregCheck;
    set rsregCheck;
    if(&qFlipState.=1) then do;
        ds = -ds;
        correct = 1 - correct;
    end;
run;
```

```
proc print data = rsregAccuracy noobs; run;
```

The accuracy of the regime-switching regression model, which is shown in [Figure 20.45](#), is close to 90%. This means that the regime-switching regression model successfully distinguishes the two regimes.

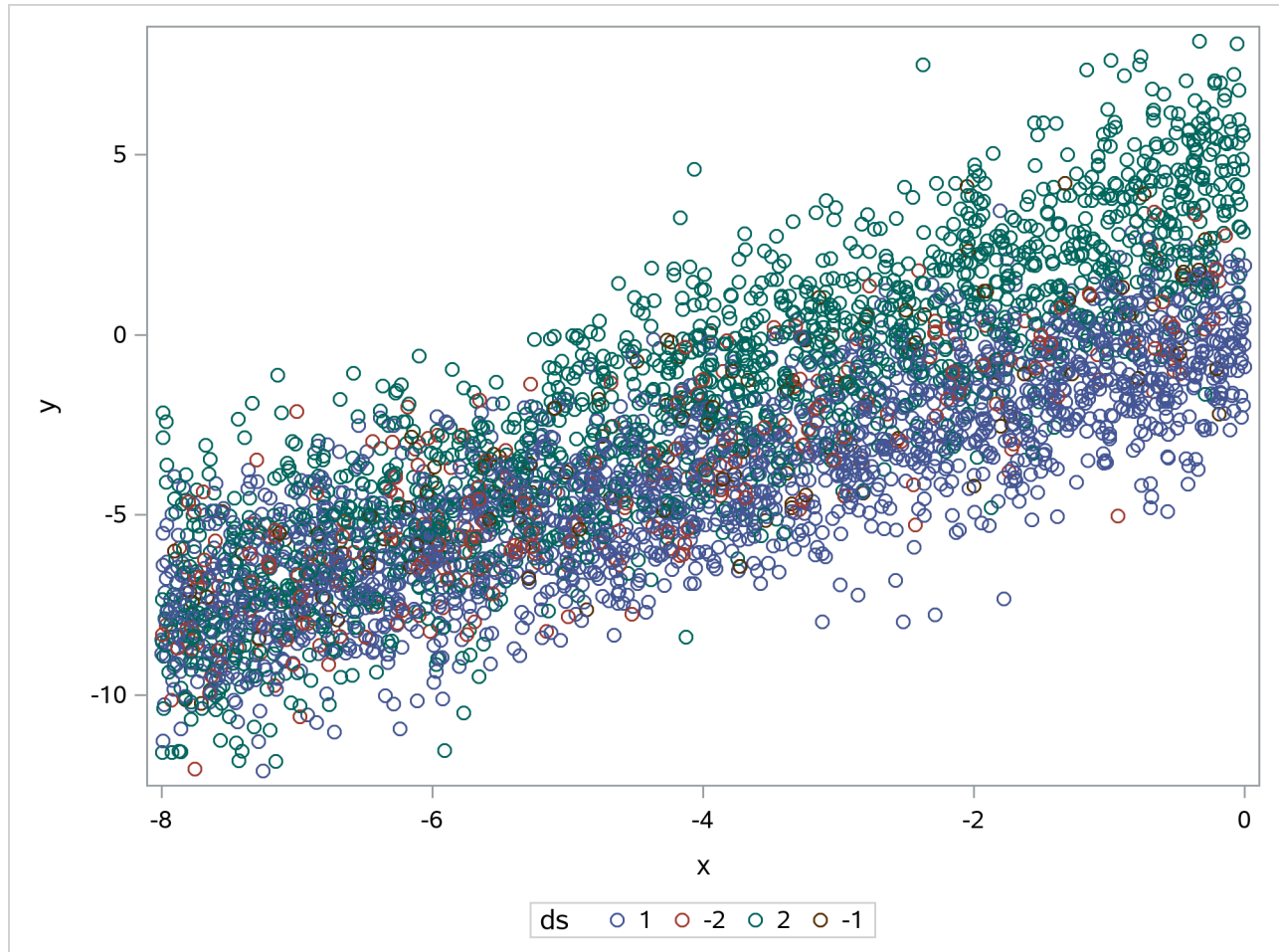
Figure 20.45 Accuracy of Decoding by the Regime-Switching Regression Model

accuracy
0.89925

The following statements create the scatter plot in [Figure 20.46](#) to show how well the data points are classified:

```
proc sgplot data=rsregCheck;
  scatter y=y x=x / group=ds;
run;
```

Figure 20.46 Scatter Plot of Data Points



Regime-Switching Autoregression Model

Let $\mathbf{Y}_t = (y_{1t}, \dots, y_{pt})'$, $t = 1, \dots, T$, denote a p -dimensional time series vector of random variables. The $\mathbf{Y}_t$ can be modeled in the regression

$$\mathbf{Y}_t = \beta_{\mathbf{S}_t} z_t + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, the regressor z_t is observable and contains the lagged dependent variables, the latent variable $\mathbf{S}_t$ is the so-called state, and $\beta_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the state $\mathbf{S}_t$. The variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the lagged dependent variables are included in the regressors and the autoregression parameters depend on the state, the $\mathbf{Y}_t$ follows different autoregressions in different regimes; hence, this type of model is called the regime-switching autoregression model.

Consider a univariate regime-switching autoregression model that has two regimes:

$$y_t = \begin{cases} 0.8 * y_{t-1} + \varepsilon_t, \varepsilon_t \sim N(0, 2.56) & \text{if } \mathbf{S}_t = 1 \\ -0.7 * y_{t-1} + \varepsilon_t, \varepsilon_t \sim N(0, 4) & \text{if } \mathbf{S}_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix}$$

The following statements simulate the time series from the previous model to provide test data for the HMM procedure:

```
%let pi1 = 0.5;
%let a11 = 0.95;
%let a22 = 0.95;
%let ar1_1_1_1 = 0.8;
%let cov1_11 = 2.56;
%let ar2_1_1_1 = -0.7;
```

```

%let cov2_11 = 4;
%let T = 1000;
%let seed = 1234;

data rsardgp;
  retain cd1_11 cd2_11;
  ylag = 0;
  do t = 1 to &T.;
    if(t=1) then do;
      /* initial probability distribution */
      p = &p11.;
      /* Cholesky decomposition of COV1 */
      cd1_11 = sqrt(&cov1_11.);
      /* Cholesky decomposition of COV2 */
      cd2_11 = sqrt(&cov2_11.);
    end;
    else do;
      /* transition probability matrix */
      if(lags=1) then p = &a11.;
      else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e = normal(&seed.);
    if(s=1) then do;
      /* y ~ N(beta1*ylag, Sigma1) at state 1 */
      y = &ar1_1_1_1 * ylag + cd1_11*e;
    end;
    else do;
      /* y ~ N(beta2*ylag, Sigma2) at state 2 */
      y = &ar2_1_1_1 * ylag + cd2_11*e;
    end;
    output;
    lags = s;
    ylag = y;
  end;
run;

data rsar;
  set rsardgp;
  keep t y;
run;

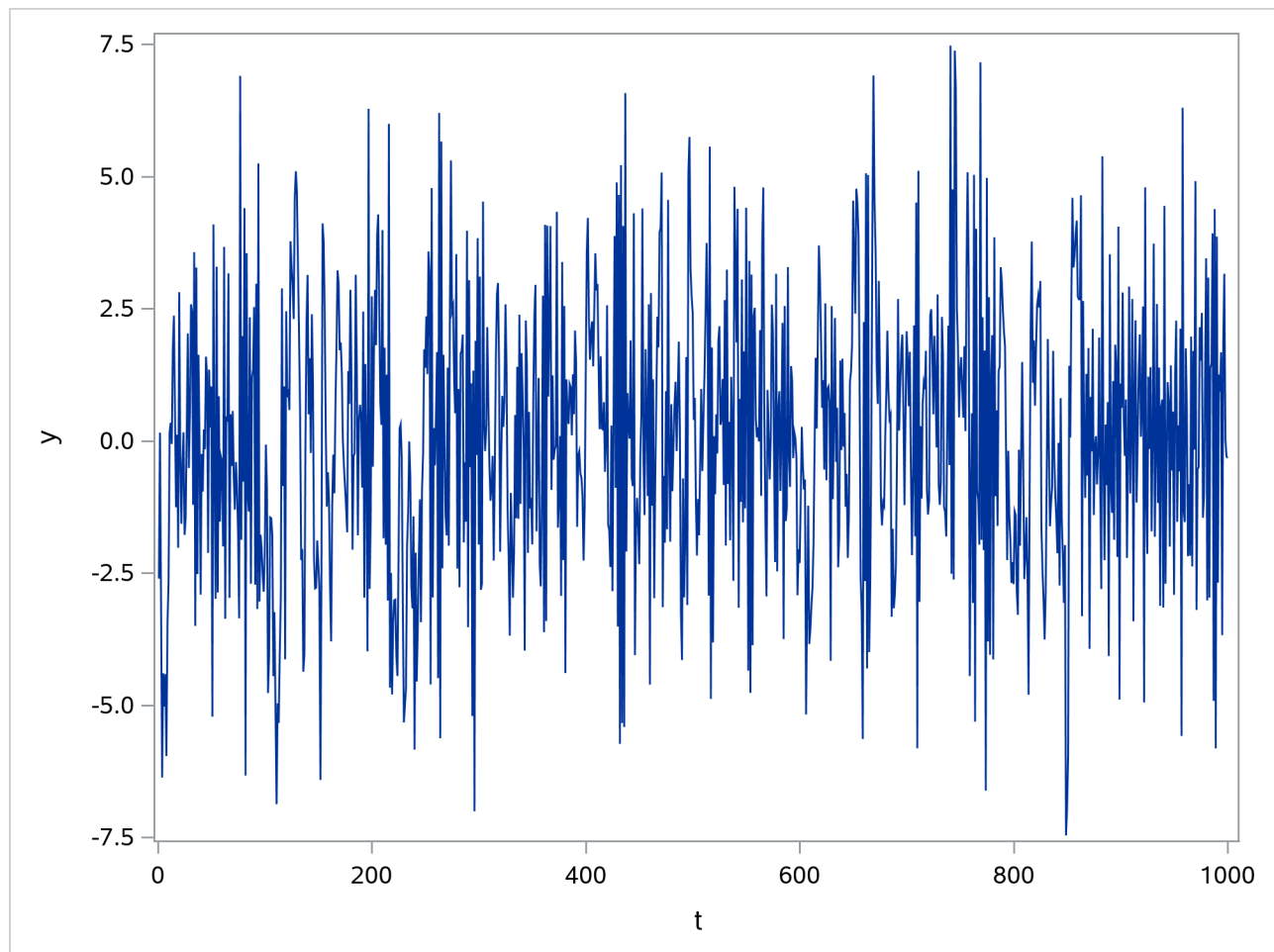
```

The following statements create the series plot in [Figure 20.47](#). It is difficult to tell directly from the plot that the series are from two different regimes.

```

proc sgplot data=rsar;
  series y=y x=t;
run;

```

Figure 20.47 Series Plot of Data Points

The following code uploads the data table and estimates the regime-switching autoregression model:

```
data mylib.rsar; set rsar; run;
proc hmm data=mylib.rsar;
  id time=t;
  model y / type=ar noint ylag=1 nstate=2 method=ml;
  optimize algorithm=activeset printLevel=3 printIterFreq=1;
  learn out=mylib.mylearn;
  filter out=mylib.myfilter;
  smooth out=mylib.mysmooth;
  decode out=mylib.mydecode;
  evaluate out=mylib.myeval;
run;
```

These statements assume that your engine libref is named mylib, but you can substitute any appropriately defined engine libref.

The estimates of all parameters are shown in [Figure 20.48](#), which displays columns for parameter name, estimate value, standard error, t value, and p -value. The parameter estimates are very close to the true parameter values that are used in the data generating process.

Figure 20.48 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard		
		Error	t Value	Pr > t
TPM1_1	0.937990	0.013985	67.07	<.0001
TPM1_2	0.062010	0.013985	4.43	<.0001
TPM2_1	0.056595	0.013425	4.22	<.0001
TPM2_2	0.943405	0.013425	70.27	<.0001
AR1_1_1_1	0.759087	0.032724	23.20	<.0001
AR2_1_1_1	-0.630710	0.037261	-16.93	<.0001
COV1_1_1	2.687805	0.198060	13.57	<.0001
COV2_1_1	4.225692	0.280319	15.07	<.0001

The accuracy of classification through the regime-switching autoregression model is calculated by the following statements:

```

data rsarCheck;
    merge rsardgp(in=a) mylib.mydecode(in=b);
    by t;
    if(s=state) then do; correct=1; ds = s; end;
    else do; correct=0; ds = -s; end;
    if(a=b);
    keep t y s correct ds;
run;

%let qFlipState = -1;
data rsarAccuracy;
    set rsarCheck;
    retain count 0 correctCount 0;
    count = count + 1;
    correctCount = correctCount + correct;
    if(count>&T.-0.5) then do;
        accuracy = correctCount / count;
        if(accuracy<0.5) then call symputx("qFlipState",1,'G');
        else call symputx("qFlipState",0,'G');
        if(accuracy<0.5) then accuracy = 1 - accuracy;
        output;
    end;
    keep accuracy;
run;

data rsarCheck;
    set rsarCheck;
    if(&qFlipState.=1) then do;
        ds = -ds;
        correct = 1 - correct;
    end;
run;

proc print data = rsarAccuracy noobs; run;

```

The accuracy of the regime-switching autoregression model, which is shown in [Figure 20.49](#), is more than 92%. This means that the regime-switching autoregression model successfully distinguishes the two regimes.

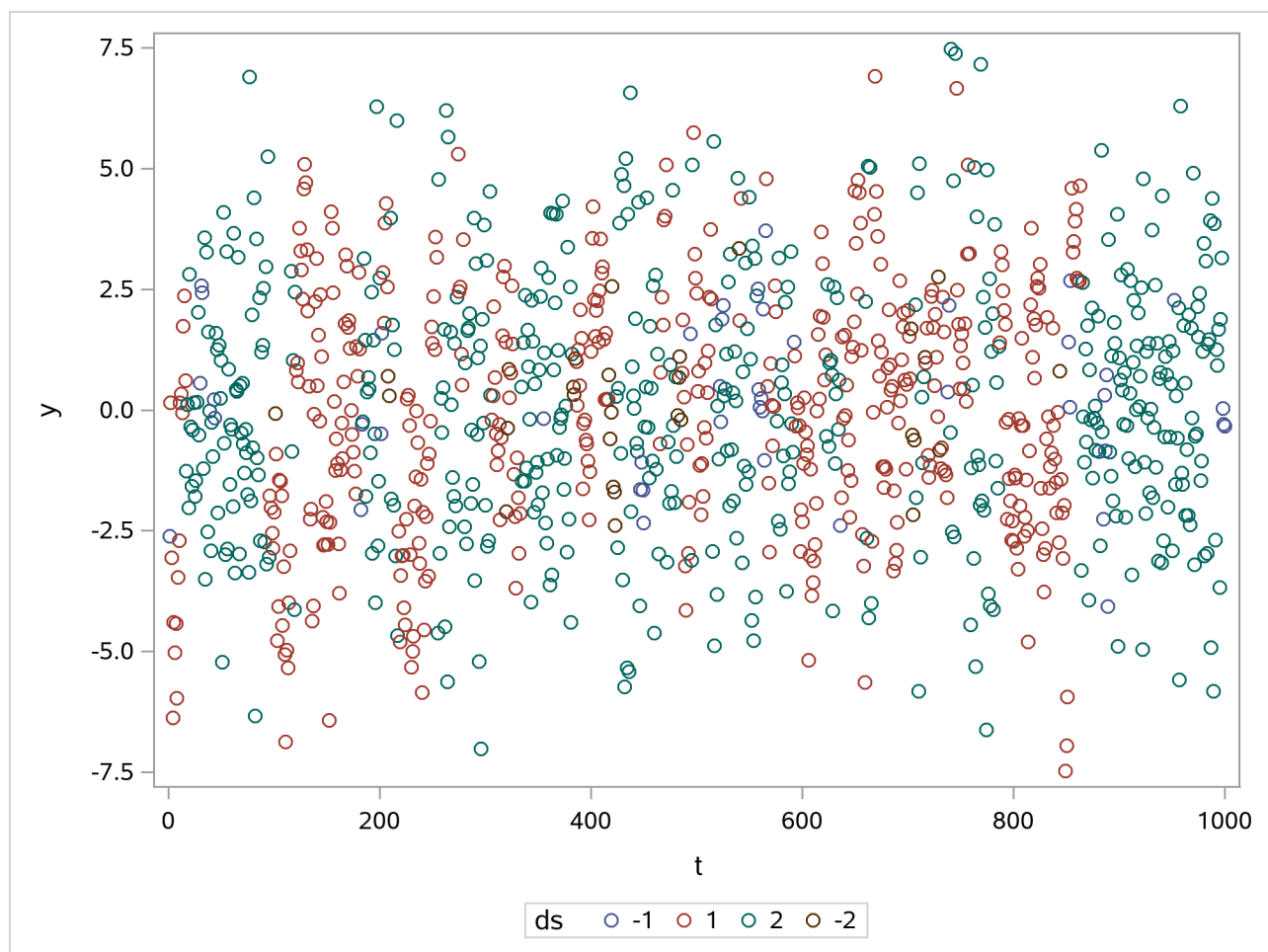
Figure 20.49 Accuracy of Decoding by the Regime-Switching Autoregression Model

accuracy
0.927

The following statements create the scatter plot in [Figure 20.50](#) to show how well the data points are classified:

```
proc sgplot data=rsarCheck;
  scatter y=y x=t / group=ds;
run;
```

Figure 20.50 Scatter Plot of Data Points



Finite Hidden Markov Model

Let $\mathbf{Y}_t, t = 1, \dots, T$, denote a random discrete variable whose sample space is a finite set of nominal classes. These values from this sample space can be represented by the finite integers $1, \dots, M$. $\mathbf{Y}_t$ can be modeled in a categorical distribution,

$$\mathbf{Y}_t | \mathbf{S}_t = i \sim C(b_{i1}, \dots, b_{iM})$$

where the category probability b_{ij} is the probability of observing $Y_t = j, j = 1, \dots, M$, conditional on the latent variable $\mathbf{S}_t = i, i = 1, \dots, K$. The $K \times M$ category probability matrix (CPM) $\mathbf{B} \equiv (b_{ij})$ is used to define the state-dependent categorical distribution of observable variables. The variable $\mathbf{S}_t$ is the so-called state and follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state, $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the variable $\mathbf{S}_t$ is unobservable and follows a Markov chain and $\mathbf{Y}_t$ is from a finite support, the model is called the finite hidden Markov model (finite HMM). The finite HMM is sometimes described as $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Consider a standard die with six faces, numbered 1 through 6. A fair die has an equal probability of displaying 1 to 6; that is, $\mathbf{Y}_t \sim C(1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$. In order to tell whether a die is fair, checking whether the opportunity of displaying each number is equal might not be enough.

The following data record the results from the roll of a die 1,000 times:

```
data mylib.finitel;
    input t    y ;
datalines;
1      6
2      3
3      6
4      5
5      1
6      2
7      1
8      6

... more lines ...
```

```

993 2
994 3
995 6
996 6
997 5
998 5
999 2
1000 6
;

```

The following statements estimate the unistate finite HMM:

```

proc hmm data=mylib.finitel;
  id time=t;
  model y / type=finite nstate=1 nCategory=6;
  estimate outall=mylib.myEstimate;
run;

```

The number of observations and model information are shown in [Figure 20.51](#).

Figure 20.51 Number of Observations and Model Information

Observation Information	
Number of Observations	1000
Number of Missing Observations	0

Model Information	
Type of Model	Finite HMM
Stationary	Yes
Number of States	1
Number of Dependent Variables	1

The estimates of the initial state probability vector (ISPV) and the transition probability matrix are trivial. The category probability matrix is shown in [Figure 20.52](#), which shows the probabilities of occurrence of each number.

Figure 20.52 Estimates of Category Probability Matrix

Estimated Category Probability Matrix						
Category						
State	1	2	3	4	5	6
1	0.16300	0.15400	0.16600	0.16700	0.17400	0.17600

The estimates of all parameters are shown in [Figure 20.53](#), which displays columns for parameter name, estimated value, standard error, t value, and p -value.

Figure 20.53 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard		
		Error	t Value	Pr > t
TPM1_1	1.000000	0		
CPM1_1	0.163000	0.011680	13.96	<.0001
CPM1_2	0.154000	0.011414	13.49	<.0001
CPM1_3	0.166000	0.011766	14.11	<.0001
CPM1_4	0.167000	0.011795	14.16	<.0001
CPM1_5	0.174000	0.011988	14.51	<.0001
CPM1_6	0.176000	0.012043	14.61	<.0001

The following statements run a Wald test under the null hypothesis that all category probabilities are equal to 1/6:

```
proc iml;
  use mylib.myEstimate;
  read all var {Estimate cov_2 cov_3 cov_4 cov_5 cov_6} into m;
  close mylib.myEstimate;

  cov = m[2:6,2:6];
  invcov = inv(cov);
  v = m[2:6,1];
  c = 1/6 // 1/6 // 1/6 // 1/6 // 1/6 ;
  v1 = v-c;

  stat = v1`*invcov*v1;
  df = 5;
  pvalue = 1- cdf('CHISQ',stat,df);
  varname={"stat" "df" "pvalue"};
  create table var varname;
  append;
  close table;
  quit;

proc print data=table noobs label;
  label stat="Stat" df = "DF" pvalue = "Pr > ChiSq";
run;
```

The p -value shown in [Figure 20.54](#) is very large, which means that the test cannot reject the null hypothesis that the category probabilities are equal to 1/6 at the 10% significance level. The die looks fair.

Figure 20.54 The Wald Test

Stat	DF	Pr > ChiSq
1.93169	5	0.85851

The log likelihood and information criteria are shown in [Figure 20.55](#).

Figure 20.55 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-1790.807551
AIC	3593.6151015
AICC	3593.6996936
BIC	3623.0616332
HQC	3604.8068383

The following code estimates a bistate finite HMM:

```
proc hmm data=mylib.finitel;
  id time=t;
  model y / type=finite nstate=2 nCategory=6;
  score outmodel=mylib.myScore;
run;
```

The number of observations and model information are shown in [Figure 20.56](#).

Figure 20.56 Number of Observations and Model Information

Observation Information	
Number of Observations	1000
Number of Missing Observations	0

Model Information	
Type of Model	Finite HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	1

The estimates of the ISPV are shown in [Figure 20.57](#).

Figure 20.57 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	Estimation
1	0.52660
2	0.47340

The estimates of the TPM are shown in [Figure 20.58](#).

Figure 20.58 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	1	2
1	0.74650	0.25350
2	0.28200	0.71800

The estimates of the CPM are shown in Figure 20.59.

Figure 20.59 Estimates of Category Probability Matrix

Estimated Category Probability Matrix						
Category						
State	1	2	3	4	5	6
1	0.00000	0.01525	0.01435	0.31762	0.31804	0.33474
2	0.34372	0.30784	0.33414	0.00000	0.01430	0.00000

The estimates of all parameters are shown in Figure 20.60, which displays columns for parameter name, estimated value, standard error, t value, and p -value.

Figure 20.60 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.746496	0.027822	26.83	<.0001
TPM1_2	0.253504	0.027822	9.11	<.0001
TPM2_1	0.281997	0.025626	11.00	<.0001
TPM2_2	0.718003	0.025626	28.02	<.0001
CPM1_1	9.9114962E-9	0.000017508	0.00	0.9995
CPM1_2	0.015249	0.017745	0.86	0.3904
CPM1_3	0.014352	0.017406	0.82	0.4098
CPM1_4	0.317621	0.022761	13.95	<.0001
CPM1_5	0.318039	0.024580	12.94	<.0001
CPM1_6	0.334739	0.023261	14.39	<.0001
CPM2_1	0.343724	0.025061	13.72	<.0001
CPM2_2	0.307839	0.025495	12.07	<.0001
CPM2_3	0.334138	0.025467	13.12	<.0001
CPM2_4	1.799124E-10	0.000002004	0.00	0.9999
CPM2_5	0.014298	0.018616	0.77	0.4426
CPM2_6	4.390824E-10	0.000004237	0.00	0.9999

In Figure 20.60, you see that the category probabilities for small numbers (1, 2, 3) in the first state and the category probabilities for large numbers (4, 5, 6) in the second state are not significant at the 10% significance level. This indicates that the small numbers appear only in one state and the large numbers appear only in the other state.

The log likelihood and information criteria are shown in Figure 20.61.

Figure 20.61 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-1697.866801
AIC	3423.7336013
AICC	3424.1599972
BIC	3492.4421752
HQC	3449.8476538

All the information criteria from the bivariate finite HMM are smaller than those from the univariate finite HMM. This indicates that the bivariate finite HMM is a better model. The trivariate finite HMM, which is not shown here, is also beaten by the bivariate finite HMM for this case.

The casino game sic bo involves whether you will get a small or big number when you roll a die. A small bet is when you predict that the die will display a 1, 2, or 3. A big bet is when you predict that the die will display a 4, 5, or 6. If your guess is correct, you get a reward; otherwise you lose the bet.

You can use PROC HMM to test the bivariate finite HMM's ability to forecast the result of this game. The following data record the results from the roll of a die 100 times:

```
data mylib.finite2;
    input t y ;
datalines;
1    6
2    4
3    6
4    1
5    1
6    1
7    1
8    4

... more lines ...

93   6
94   6
95   5
96   2
97   1
98   6
99   4
100  6
;
```

The following statements perform one-step online (continuous) forecasting of betting big with the estimated parameters from the preceding PROC HMM statements and compare the prediction with the finite2 data set. If the forecasting is correct, you gain one unit. Otherwise you lose one unit.

```
proc hmm data=mylib.finite2;
    id time=t;
    score inmodel=mylib.myScore;
    forecast out=mylib.myforecast online;
run;
data mylib.forecast;
    set mylib.myforecast;
    if (Category1+Category2+Category3>0.5) then bet = 0;
    else bet = 1;
run;
data mylib.forecast;
    set mylib.forecast;
    t=t+step; *so t equals the time when forecast is on;
run;
```



```

data mylib.finite2;
  set mylib.finite2;
  if(y<=3) then big=0;
  else big=1;
run;
data fnHmmCheck;
  merge mylib.finite2(in=a) mylib.forecast(in=b);
  by t;
run;
data fnHmmCheck;
  set fnHmmCheck;
  *for first round bet according to initial probability;
  if(t=1) then bet=1;
  if(big=bet) then gain=1;
  else gain=-1;
  keep t y gain;
run;
data totalgain;
  set fnHmmCheck;
  Totalgain + gain;
  if (t=100) then do
    Zscore = Totalgain/10;
    pvalue = 2*(1- cdf('NORMAL', Zscore));
    output totalgain;
  end;
  keep Totalgain Zscore pvalue;
run;
proc print data = totalgain noobs label;
  label totalgain = "Total gain" Zscore="Z-score" pvalue="Pr > |z|";
run;

```

The total gain from rolling the die 100 times is shown in Figure 20.62. Under the null hypothesis that the die is fair, according to the central limit theorem the expected return of betting 100 times is 0 with a standard error of 10 units. The values shown in Figure 20.62 indicate that the null hypothesis is rejected even at the 1% significance level. This means that the finite HMM successfully detects the loaded die.

Figure 20.62 Gain of Forecasting by the Finite HMM

Total gain	Z-score	Pr > z
48	4.8	.000001587

The data generating process behind the observed data sets is as follows:

$$Y_t \sim \begin{cases} C(1/3, 1/3, 1/3, 0, 0, 0) & \text{if } S_t = 1 \\ C(0, 0, 0, 1/3, 1/3, 1/3) & \text{if } S_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, A = \begin{pmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{pmatrix}$$

The following statements simulate the original time series data for this example:

```
%let pil = 0.5; *initial probability in state 1;
%let a11 = 0.75;
%let a22 = 0.75;

%macro simData(name, T, seed);
data mylib.&name.(keep = t y);
  p = &pil.;
  call streaminit(&seed);
  do t = 1 to &T.;
    if(t=1) then
      /* initial probability distribution */
      p = &pil.;
    else
      /* transition probability matrix */
      if(lagx=1) then p = &a11.;
      else p = 1-&a22.;
      u = rand("Uniform");
      if(u<=p) then x=1;
      *Using the random value u to control regimes, set x=number of regimes;
      else x = 2;
      if(x=1) then
        do;
          /* at state 1 */
          y = rand("Table", 1/3, 1/3, 1/3, 0 , 0 , 0);
        end;
      else do;
        /* at state 2 */
        y = rand("Table", 0 , 0 , 0 , 1/3, 1/3, 1/3);
      end;
      output;
      lagx=x;
    end;
  run;
%mend;

%simData(finite1,1000, 3);
%simData(finite2,100, 4);
```

PROC HMM assumes that values of the response variables are consecutive natural numbers that start from 1. If this is not the case, you might need to preprocess the input data set before running PROC HMM. You can use the SAS macro %FiniteHMM for automatic preprocessing and postprocessing. For more information about %FiniteHMM, see the section “[The %FiniteHMM Macro](#)” on page 1278.

Poisson Hidden Markov Model

Let $y_t, t = 1, \dots, T$, denote a discrete time-series random variable whose sample space is a set of nonnegative integers.

The variable y_t is said to follow a Poisson HMM distribution, with the parameter $\lambda_{s_t} > 0$, if it follows a Poisson distribution in each state S_t . The conditional probability distribution is denoted as $y_t | S_t = i \sim \text{Pois}(\lambda_{S_t})$, where i is a positive integer.

The variable S_t is the so-called state and follows the first-order Markov chain; that is,

$$p(S_t | S_{t-1}, S_{t-2}, \dots, S_1) = p(S_t | S_{t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. The range of S_t is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(S_t = j | S_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state, S_1 :

$$\pi = \{\pi_i = p(S_1 = i), i = 1, \dots, K\}$$

Because the state variable S_t is unobservable and follows a Markov chain, and because y_t is from a Poisson distribution that depends on a different state, the model is called the Poisson hidden Markov model (Poisson HMM).

Consider a univariate Poisson HMM model $y_t \sim \text{Pois}(\lambda_{s_t})$ that has two regimes, $\lambda_1 = 5$ and $\lambda_2 = 10$.

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

```
%let pi1 = 0.7;
%let a11 = 0.9;
%let a22 = 0.8;
%let lambda1 = 5;
%let lambda2 = 10;

%macro dgpSHMM(table, T, nSections, seed);
data &table;
  call streaminit(&seed.);
  secLen = ceil(&T./&nSections.);
  do t = 1 to &T.;
    sec = floor((t-1)/secLen)+1;
    if(t=1) then do;
```

```

/* initial probability distribution */
p = &pil.;
end;
else do;
/* transition probability matrix */
if(lags=1) then p = &a11.;
else p = 1-&a22.;
end;
u = rand("Uniform");
if(u<=p) then s=1;
else s = 2;
if(s=1) then do;
/* at state 1 */
y = rand("Poisson",&lambda1.);
end;
else do;
/* at state 2 */
y = rand("Poisson",&lambda2.);
end;
output;
lags = s;
end;
run;
proc sort data=&table.; by sec t; run;
%mend;

%let T = 1000 ;
%let nSections = 2;
%let seed = 1234;
%dgppSHMM(hmmpsdt, &T., &nSections., &seed.);

data mylib.hmmpsdt;
set hmmpsdt;
run;

```

The following code estimates a bivariate Poisson HMM:

```

proc hmm data=mylib.hmmpsdt;
id time=t;
model y / type=Poisson nstate=2;
run;

```

The number of observations and model information are shown in [Figure 20.63](#).

Figure 20.63 Number of Observations and Model Information

Observation Information	
Number of Observations	1000
Number of Missing Observations	0

Model Information	
Type of Model	Poisson HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	1

The estimates of the ISPV are shown in [Figure 20.64](#).

Figure 20.64 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	Estimation
1	0.71828
2	0.28172

The estimates of the TPM are shown in [Figure 20.65](#).

Figure 20.65 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	1	2
1	0.93328	0.06672
2	0.17010	0.82990

The estimates of the lambda are shown in [Figure 20.66](#).

Figure 20.66 Estimates of Lambda of Poisson Distribution

Lambda	
State	Estimation
1	5.15141
2	9.84789

The estimates of all parameters are shown in [Figure 20.67](#), which displays columns for parameter name, estimated value, standard error, t value, and p -value.

Figure 20.67 Parameter Estimates

Parameter Estimates				
		Standard		
Parameter	Estimate	Error	t Value	Pr > t
TPM1_1	0.933281	0.014084	66.26	<.0001
TPM1_2	0.066719	0.014084	4.74	<.0001
TPM2_1	0.170104	0.031678	5.37	<.0001
TPM2_2	0.829896	0.031678	26.20	<.0001
LAMBDA1	5.151408	0.108184	47.62	<.0001
LAMBDA2	9.847886	0.254877	38.64	<.0001

Syntax: HMM Procedure

```

PROC HMM DATA=libref.data-table < options > ;
  DECODE < option > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  ESTIMATE < options > ;
  EVALUATE < option > ;
  FILTER < option > ;
  FORECAST < options > ;
  ID TIME= variable < SECTION= variable > ;
  INITIAL equation < , ... , equation > ;
  MODEL dependents < = regressors > < / options > ;
  OPTIMIZE < options > ;
  PRIOR distribution < , ... , distribution > ;
  SCORE < options > ;
  SMOOTH < option > ;
  STORE < OUT= > libref.data-table ;

```

Functional Summary

The statements and options available in the HMM procedure are summarized in [Table 20.1](#).

Table 20.1 Functional Summary

Description	Statement	Option
Input Data Table Options		
Specifies the input data table	PROC HMM	DATA=
Specifies the input data table for parameter estimation	ESTIMATE	IN=
Specifies the input data table for model information	SCORE	INMODEL=

Table 20.1 *continued*

Description	Statement	Option
Output Data Table Options		
Writes the log likelihood and information criteria to the specified output data table	PROC HMM	OUTSTAT=
Writes the parameter estimates to the specified output data table	ESTIMATE	OUT=
Writes the parameter estimates and their standard errors and covariance matrix to the specified output data table	ESTIMATE	OUTALL=
Writes the evaluation results to the specified output data table	EVALUATE	OUT=
Writes the decoding results to the specified output data table	DECODE	OUT=
Writes the filtering results to the specified output data table	FILTER	OUT=
Writes the forecasting results to the specified output data table	FORECAST	OUT=
Writes the state distributions, means, standard errors, confidence intervals, medians, and covariance matrices of forecasts for dependent variables to the specified output data table	FORECAST	OUTALL=
Writes the model information to the specified output data table	SCORE	OUTMODEL=
Writes the smoothing results to the specified output data table	SMOOTH	OUT=
Writes the model information to the specified output data table that can be processed by the ASTORE procedure	STORE	OUT=
ID Variable		
Specifies the variable that identifies the time or sequence	ID	TIME=
Specifies the variable that identifies the section	ID	SECTION=
Model Options		
Specifies the mean form in a regime-switching autoregression model	MODEL	ARMEAN=
Estimates the initial state probability vector (ISPV)	MODEL	ESTISPV
Specifies the estimation method	MODEL	METHOD=
Specifies the number of categories in a state for a finite HMM	MODEL	NCATEGORY
Specifies the number of components in a state in a Gaussian mixture HMM (GM HMM)	MODEL	NCOMPONENT
Suppresses the present values of the exogenous variables	MODEL	NOCURRENTX

Table 20.1 *continued*

Description	Statement	Option
Suppresses the constant (intercept) parameters	MODEL	NOINT
Specifies the number of seasonal periods	MODEL	NSEASON=
Specifies the number or the range of numbers of hidden states	MODEL	NSTATE=
Centers seasonal dummies	MODEL	SCENTER
Specifies the state-independent parameters	MODEL	STATEINDEPENDENT=
Specifies the degree of deterministic time trend	MODEL	TREND=
Specifies the type of model of interest	MODEL	TYPE=
Specifies the lags of exogenous (independent) variables	MODEL	XLAG=
Specifies the order or the range of orders of the autoregressive process	MODEL	YLAG=
Optimization Options		
Specifies a nonnegative integer to use as the seed for generating random number sequences	PROC HMM	SEED=
Specifies the initial parameter values for optimization	INITIAL	
Specifies the optimization algorithm	OPTIMIZE	ALGORITHM=
Specifies whether to calculate the covariance of parameters	OPTIMIZE	COVOFPARM=
Specifies the maximum number of iterations in the optimization process	OPTIMIZE	MAXITER=
Specifies the maximum time (in seconds) allowed for optimization	OPTIMIZE	MAXTIME=
Specifies the print iteration frequency	OPTIMIZE	PRINTITERFREQ=
Specifies the print level	OPTIMIZE	PRINTLEVEL=
Specifies the priors for parameters	PRIOR	
Optimization Suboptions for the Active-Set or Interior Point Algorithm		
Specifies the range from which each variable can take values during the sampling process in multistart mode	OPTIMIZE	MSRANGE=
Specifies whether to enable multistart mode	OPTIMIZE	MULTISTART=
Specifies the upper limit on the magnitude of the objective value	OPTIMIZE	OBJLIMIT=
Specifies the tolerance for infeasibility	OPTIMIZE	FEASTOL=
Specifies the tolerance for the optimality error	OPTIMIZE	OPTTOL=
Specifies whether the optimizer should return only a solution that is locally optimal	OPTIMIZE	SOLTYPE=
Optimization Suboption for the Expectation-Maximization (EM) Algorithm		
Specifies the minimum number of iterations in the optimization process	OPTIMIZE	MINITER=

Table 20.1 *continued*

Description	Statement	Option
Executes the EM algorithm in a single thread on a single node	OPTIMIZE	SINGLE
Optimization Suboptions for the Stochastic Gradient Descent (SGD) Algorithm		
Specifies the annealing rate	OPTIMIZE	ANNEALINGRATE=
Specifies the learning rate	OPTIMIZE	LEARNINGRATE=
Specifies the size of the minibatches to use	OPTIMIZE	MINIBATCHSIZE=
Specifies the value for momentum	OPTIMIZE	MOMENTUM=
Suppresses the adaptive gradient descent algorithm	OPTIMIZE	NONADUPDATE
Specifies the value of adaptive rate	OPTIMIZE	ADAPTIVEDECAY=
Requests that computational threads share a common weight vector and that they update the weight vector without race conditions	OPTIMIZE	USELOCKING=
Specifies the length of a chain in a minibatch	OPTIMIZE	CHAIN=(LENGTH=)
Specifies the buffer length on the left side of a chain in a minibatch	OPTIMIZE	CHAIN=(BUFFER=)
Optimization Suboptions for the Distributed Multistart Option		
Specifies the number of CAS subsessions that are dedicated to the function evaluations	OPTIMIZE	FESESSIONS=
Specifies the number of computing nodes in each CAS subsession that are dedicated to the function evaluations	OPTIMIZE	FESESSNODES=
Specifies the maximum number of starting points to be used by the multistart algorithm	OPTIMIZE	MAXSTARTS=
Specifies the ratio of the total number of sample points to be generated to the value of the MAXSTARTS= option	OPTIMIZE	SAMPSELRATIO=
Specifies the number of computing nodes in the CAS session dedicated to the multistart algorithm	OPTIMIZE	SESSNODES=
Forecasting Options		
Specifies a nonnegative integer to use as the seed for generating random number sequences	PROC HMM	SEED=
Specifies the size of confidence limits for forecasting	FORECAST	ALPHA=
Starts forecasting before the end of the input data	FORECAST	BACK=
Specifies how many periods to forecast	FORECAST	LEAD=
Specifies the number of paths to simulate in the forecasting of a regime-switching autoregression model	FORECAST	NSIM=
Specifies whether to perform forecasts after each observation	FORECAST	ONLINE

Table 20.1 *continued*

Description	Statement	Option
Output Control Options		
Specifies the order in which to label the states (and components)	PROC HMM	LABELSWITCH=
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	

PROC HMM Statement

PROC HMM DATA=*libref.data-table* < options > ;

The PROC HMM statement invokes the HMM procedure. You must specify the following *option*:

DATA=*libref.data-table*

names the input data table for PROC HMM to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

data-table specifies the name of the input data table.

You can also specify the following *options*:

LABELSWITCH=(*options*)

specifies the order in which to label the states (and components). The label switching problem is a common issue for HMMs because any two HMMs are mathematically equivalent if the only difference between two models is that the states are labeled in different ways. For example, consider the following two univariate bistate Gaussian HMMs, which are described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, μ_1 and Σ_1 are mean and covariance parameters for state 1, and μ_2 and Σ_2 are mean and covariance parameters for state 2:

$$\text{model 1: } \pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{pmatrix}, \mu_1 = 1, \Sigma_1 = 1, \mu_2 = -1, \Sigma_2 = 4$$

$$\text{model 2: } \pi = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{pmatrix}, \mu_1 = -1, \Sigma_1 = 4, \mu_2 = 1, \Sigma_2 = 1$$

In theory, these two models are equivalent, and just the state labels “1” and “2” are switched. When the labels are switched, it is inconvenient to compare any state-related results, such as filtered or smoothed probabilities or decoded path. The LABELSWITCH= option helps you define the order of the state (and component) labels. You can specify the following *options*:

OUT=*libref.data-table*

writes the changes of state labels (and component labels in the case of GM HMMs) to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

REORDER=(*number-list*)

specifies the new order of the state labels, where the *number-list* directly specifies the new order of the states. The *number-list* must be a permutation of numbers 1 to K , where K is the number of states; that is, the size of the *number-list* must be K , and each element in the *number-list* must be unique and between 1 and K . For example, **REORDER=**(3 1 2) implies that the original state 1 becomes state 3, state 2 becomes state 1, and state 3 becomes state 2. If you specify this option, the **SORT=** option is ignored.

REORDERCOMPONENT=(*number-list*)

specifies the new order of the component labels in the case of GM HMMs, where the *number-list* directly specifies the new order of the components for each state. For a K -state M -component GM HMM, the size of the *number-list*, $\{n_{11}, \dots, n_{1M}, \dots, n_{K1}, \dots, n_{KM}\}$, must be KM , and each sublist $\{n_{i1}, \dots, n_{iM}\}$, $i = 1, \dots, K$, must be a permutation of numbers 1 to M —that is, each element in the sublist $\{n_{i1}, \dots, n_{iM}\}$ must be unique and between 1 and M . For example, **REORDERCOMPONENT=**(3 2 1 2 1 3) for a bistate tricomponent GM HMM implies that the original components 1, 2, and 3 for the original state 1 become components 3, 2, and 1, respectively, for the original state 1, and the original components 1, 2, and 3 for the original state 2 become components 2, 1, and 3, respectively, for the original state 2. If you specify this option, the labels for components in each state are not sorted.

SORT=*type* <(*name-list*)>

specifies whether to sort the labels of states (and components) and in which type of order. You can specify the following *types*:

ASC <(*name-list*)> sorts the labels in ascending order. You can add a *name-list* in parentheses in order to sort the labels of states (and components). The possible names that can appear in the *name-list* are ISPV, TPM, MCP, CPM, LAMBDA, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV; the meanings of those names are defined in Table 20.4. Any parameters that are not in the model are ignored.

DESC <(*name-list*)> sorts the labels in descending order. You can add a *name-list* in parentheses in order to sort the labels of states (and components). The possible names that can appear in the *name-list* are ISPV, TPM, MCP, CPM, LAMBDA, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV; the meanings of those names are defined in Table 20.4. Any parameters that are not in the model are ignored.

NONE does not sort the labels.

By default, if the **INMODEL=** option is not specified in the **SCORE** statement, **SORT=ASC(SIGMA)**; otherwise, **SORT=NONE**. For example, when the **ESTISPV** option is not specified, the ISPV is not in the stationary HMM; if you include ISPV in the *name-list*,

it is ignored. For a GM HMM, the labels of components for each state are first sorted by the parameters state by state if the REORDERCOMPONENT= option is not specified; then the labels of states are sorted by the parameters that are stacked in the new order of components.

For more details and examples, see the section “[Label Switching Problem](#)” on page 1321.

OUTSTAT=*libref.data-table*

writes the log likelihood and information criteria to the specified output data table. The information criteria include Akaike’s information criterion (AIC), the corrected Akaike’s information criterion (AICC), the Bayesian information criterion (BIC, also referred to as the Schwarz Bayesian criterion, SBC), and the Hannan-Quinn criterion (HQC). *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1186.

SEED=*number*

specifies a nonnegative integer to use as the seed for generating random number sequences. When the seed value is 0, random number sequences are generated by using the time of day from the computer’s clock. Seed values greater than 0 generate reproducible random number sequences. By default, SEED=1.

DECODE Statement

DECODE < *option* > ;

The DECODE statement solves the decoding problem by finding the best possible state path for the observations. You can specify the following *option*:

OUT=*libref.data-table*

writes the decoding results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1186.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE

statement is taking precedence because it can further filter the tables. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 1342. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 1342. You cannot specify the ODS table named `OutputCasTables` in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The `DISPLAYOUT` statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

ESTIMATE Statement

ESTIMATE <options> ;

LEARN <options> ;

The `ESTIMATE` statement estimates the parameters. You can specify the following *options*:

IN=libref.data-table

specifies the input data table for parameter estimation. In general, the data table is an output data table that is produced by the `OUT=` option in the `ESTIMATE` statement in a previous call of the HMM procedure. To estimate each model that is specified in the `MODEL` statement, PROC HMM first checks whether an `INITIAL` statement is specified for the model. If the `INITIAL` statement exists for the model, the initial values that are specified in the `INITIAL` statement are used; otherwise, PROC HMM searches the `IN=` data table to see whether it contains a model that has the same model specification. If the `IN=` data table contains a matched model, its values are used as the initial values for the model that is specified in the `MODEL` statement. **NOTE:** When PROC HMM checks for matched models, the `ESTISPV` model option, which calculates the `ISPV` parameter, presents a special case. If the current model specification does not include the `ESTISPV` option, and the `IN=` table is a parameter estimate output table from a previous model specification that does include the `ESTISPV` option, then values in the `IN=` table are not used as initial values, even if all other specifications of the two models are the same.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1186.

OUT=libref.data-table

writes the parameter estimates to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

OUTALL=libref.data-table

writes the parameter estimates and their standard errors and covariance matrix to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

EVALUATE Statement

EVALUATE < option > ;

The EVALUATE statement evaluates the model by calculating the log likelihood. You can specify the following *option*:

OUT=libref.data-table

writes the evaluation results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

FILTER Statement

FILTER < option > ;

The FILTER statement solves the filtering problem, which is to find the probability distribution of the current state, conditional on the observations that precede and include the current observation. You can specify the following *option*:

OUT=libref.data-table

writes the filtering results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

FORECAST Statement

FORECAST <options> ;

The FORECAST statement finds the probability distribution of the future states and the dependent variables. You can specify the following *options*:

ALPHA=number

specifies the forecast confidence limit size, where *number* is between 0 and 1. When you specify this option, the upper and lower confidence limits define the $100\alpha\%$ confidence interval. By default, ALPHA=0.95, which produces 95% confidence intervals.

BACK=number

specifies the number of observations before the end of the data at which the multistep forecasts begin, where *number* must be less than or equal to the number of observations minus the number of lagged regressors in the model. By default, BACK=0, which means that the forecasts start at the end of the available data.

LEAD=number

specifies the number of multistep forecast values to compute, where *number* must be a nonnegative integer. By default, LEAD=1, which means that a one-step-ahead forecast is performed.

METHOD=ANALYTIC | SIMULATION

specifies the forecasting method. The SIMULATION option works only when you specify the TYPE=AR option in the MODEL statement. You can specify the following values:

ANALYTIC specifies the analytical method of forecasting.

SIMULATION specifies the simulation method of forecasting for autoregressive HMM.

By default, METHOD=SIMULATION for autoregressive HMM and METHOD=ANALYTIC for other types of HMM.

NSIM=number

specifies the number of paths to be simulated in the forecasting of a regime-switching autoregression model, where *number* must be a positive integer. By default, NSIM=100.

ONLINE

requests that the (multistep) forecasts be done after each observation. When you specify this option, the BACK= option is ignored.

OUT=libref.data-table

writes the forecasting results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

OUTALL=libref.data-table

writes the state distributions, means, standard errors, confidence intervals, medians, and covariance matrices of forecasts for dependent variables to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

ID Statement

ID TIME=variable < SECTION=variable > ;

The ID statement identifies observations in the input data table by specifying a variable for the time series data or two variables for the cross-sectional time series data.

You must specify the following option:

TIME=variable

specifies the temporal or sequential order of the observations. The *variable* cannot have missing values. If you omit the SECTION= option, the *variable* cannot have duplicate values. If you specify the SECTION= option, the *variable* cannot have duplicate values within a section.

You can also specify the following option:

SECTION=variable

identifies the section of each observation for modeling cross-sectional time series data. The *variable* cannot have missing values.

INITIAL Statement

INITIAL equation < , . . . , equation > ;

The INITIAL statement specifies the initial parameter values for nonlinear optimization when the maximum likelihood method or the maximum a posteriori method is applied to the estimation of the HMMs (that is, METHOD=ML or METHOD=MAP, respectively, in the MODEL statement). Only one INITIAL statement is allowed. If you specify more than one *equation*, separate them with commas. The INITIAL statement supports only equations; hence, inequality comparison operators (<, <=, >, and >=) and the distribution operator (~) are not supported in the INITIAL statement. If you omit this statement, the initial values of parameters are determined by the input data.

If you specify the NSTATE= n_1 : n_2 option or the YLAG= m_1 : m_2 option (or both) in the MODEL statement to estimate multiple models, the INITIAL statement is applied to the first model that corresponds to the specification of NSTATE= n_1 or YLAG= m_1 (or that corresponds to both NSTATE= n_1 and YLAG= m_1 if both options are specified).

To use the INITIAL statement, you need to know the form of the model: different sets of parameters are available for different types of HMMs. Nonlinear equations on parameters are not supported.

The *equation* is in the form of a matrix expression. For more information about the matrix expression, see the section “[Matrix Expression](#)” on page 1311.

The initial parameter values are values that solve the specified linear equations. If you do not specify initial values for all parameters, the default initial value for any parameter that is not specified in the INITIAL statement is 0, except for the following:

- The diagonal elements of the **SIGMA** or **COV** parameter matrix are set to ones if the **SIGMA** or **COV** parameter matrix is to be estimated.
- If a row of the **TPM** parameter matrix contains all zeros, each element in the row is set to $1/K$, where K is the number of states; if a row of the **TPM** parameter matrix does not add up to one, the row is normalized by dividing each element in the row by the sum of the row.
- If all elements of the **ISPV** parameter vector are zeros, each element of the **ISPV** parameter vector is set to $1/K$, where K is the number of states; if the sum of the **ISPV** parameter vector is not equal to one, it is normalized by dividing each element of the **ISPV** parameter vector by the sum of the **ISPV** parameter vector.
- The value of the **LAMBDA** parameter vector for each state is set to one if the **LAMBDA** parameter vector is to be estimated.

The following example uses the INITIAL statement for a bivariate three-state Gaussian HMM, which PROC HMM estimates by the maximum likelihood method by default:

```
proc hmm data=One;
  id time = t;
  model y1 y2 / type=gaussian nstate=3;
  initial TPM={0.8 0.1 0.1, 0.2 0.7 0.1, 0.9 0.05 0.05},
            MU={-1 -1, 0 0, 1 1}` ,
            SIGMA(1)=4*I(2),
            SIGMA(2)=I(2),
            SIGMA(3)=4*I(2)+2;
run;
```

MODEL Statement

MODEL *dependents* < = *regressors* > < / *options* > ;

The MODEL statement specifies dependent (endogenous) variables and regressors (the independent or exogenous variables) for the HMM model. Only one MODEL statement is allowed.

You can specify the following *options* after a forward slash (/):

ARMEAN=ADJUSTED | STANDARD

specifies the mean form in a regime-switching autoregression model. The ARMEAN= option applies only when you specify the TYPE=AR option in the MODEL statement. You can specify the following values:

ADJUSTED	specifies the mean-adjusted form of regime-switching autoregression model.
STANDARD	specifies the standard regime-switching autoregression model.

By default, METHOD=STANDARD.

ESTISPV

estimates the initial state probability vector (ISPV). By default, the ISPV is fixed to the stationary distribution of the Markov chain. For the finite-state-space and homogeneous HMM, the stationary distribution of the Markov chain is the π such that $\pi' A = \pi'$, where A is the transition probability matrix (TPM); that is, π is the leading left eigenvector of A . If you do not specify this option but you do specify the ISPV function in the INITIAL or PRIOR statement, the HMM procedure issues errors and then stops.

METHOD=MAP | ML

requests the type of estimates to be computed. You can specify the following values:

MAP	specifies the maximum a posteriori method.
ML	specifies the maximum likelihood method.

By default, METHOD=ML.

NCATEGORY=number

specifies the number of categories in a state for a finite HMM, where *number* must be a positive integer. This option applies only when you specify TYPE=FINITE. By default, NCATEGORY=2.

NCOMPONENT=number

specifies the number of components in a state for a Gaussian mixture HMM (GM HMM), where *number* must be a positive integer. This option applies only when you specify TYPE=GAUSSIANMIXTURE. By default, NCOMPONENT=1.

NOCURRENTX

suppresses the present values of the exogenous variables. This option applies only when you specify TYPE=AR or TYPE=REG.

NOINT

suppresses the constant (intercept) parameters. This option applies only when you specify TYPE=AR or TYPE=REG.

NSEASON=s

specifies the number of seasonal periods (where *s* must be a positive integer) and adds (*s*−1) seasonal dummies to the regressors. This option applies only when you specify TYPE=AR or TYPE=REG and do not specify the NOINT option.

NSTATE=number1 < : number2 >

specifies the number or the range of numbers of hidden states, where *number1* and *number2* must be positive integers. If you specify NSTATE=*number1*, the number of hidden states is *number1*; if you specify NSTATE=*number1:number2*, multiple models whose number of hidden states is inclusively between *number1* and *number2* are estimated and inferred. If you specify the same number for both *number1* and *number2*, *number2* is ignored. By default, NSTATE=2.

SCENTER

centers seasonal dummies that are implied by the NSEASON= option. The centered seasonal dummies are generated by $c - \frac{1}{s}$, where c is a seasonal dummy that the NSEASON= s option generates. This option applies only when you specify the NSEASON= option and TYPE=AR or TYPE=REG.

STATEINDEPENDENT=(name-list)

specifies the parameters to be estimated independent of state. The names of the independent parameters are entered in the *name-list* in parentheses. The names that can appear in the *name-list* are ISPV, TPM, MCP, CPM, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV. Table 20.4 gives the meanings of these names. This option does not support inputting the component index after the parameter names. For example, it supports COV but not COV(1,1,1). The name ISPV is ignored unless ESTISPV is also specified. Any parameters that are not in the model are ignored. Currently this option does not support ALGORITHM=SGD in the OPTIMIZE statement. The estimates are ordered by the names specified in the *name-list*.

To see how this option works, consider the following univariate bistate Gaussian HMM, which is described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, μ_1 and Σ_1 are the mean and covariance parameters for state 1, and μ_2 and Σ_2 are the mean and covariance parameters for state 2. STATEINDEPENDENT=(SIGMA) estimates the model with $\Sigma_1 = \Sigma_2$. The following statement specifies this option:

```
model y / stateindependent=(sigma);
```

For more details and an example, see “[Example 20.3: Analysis of the Business Cycle by Using the Regime-Switching Mean-Adjusted Autoregression Model](#)” on page 1395.

TREND=LINEAR | QUAD

specifies the degree of deterministic time trend that the model includes. You can specify the following values:

LINEAR	includes a linear time trend as a regressor.
QUAD	includes linear and quadratic time trends as regressors.

This option applies only when you specify TYPE=AR or TYPE=REG.

TYPE=AR | FINITE | GAUSSIAN | GAUSSIANMIXTURE | POISSON | REG

specifies the type of the model of interest to use. You can specify the following values:

AR	uses the regime-switching autoregression model (RS-AR, also known as the autoregressive hidden Markov model, AR HMM).
FINITE	uses the finite hidden Markov model (finite HMM).
GAUSSIAN	uses the Gaussian hidden Markov model (Gaussian HMM).
GAUSSIANMIXTURE	uses the Gaussian mixture hidden Markov model (GM HMM).
POISSON	uses the Poisson hidden Markov model (Poisson HMM).
REG	uses the regime-switching regression model (RS-REG, also known as the regression hidden Markov model, REG HMM).

By default, TYPE=GAUSSIAN.

XLAG=number

specifies the lags of exogenous (independent) variables. To exclude the present values of exogenous variables from the model, you must specify the NOCURRENTX option. This option applies only when you specify TYPE=AR or TYPE=REG. By default, XLAG=0.

YLAG=number1 < : number2 >

specifies the order or the range of orders of the autoregressive process, where *number1* and *number2* must be nonnegative integers. If you specify YLAG=*number1*, the order of the autoregressive process is *number1*; if you specify YLAG=*number1:number2*, multiple models whose order of the autoregressive process is inclusively between *number1* and *number2* are estimated and inferred. If you specify the same number for both *number1* and *number2*, *number2* is ignored. This option applies only when you specify TYPE=AR. By default, YLAG=0.

OPTIMIZE Statement

OPTIMIZE < *options* > ;

The OPTIMIZE statement optimizes the parameter estimation.

You can specify the following *options*:

ALGORITHM=type < (suboptions) >

specifies the optimization algorithm to use. You can specify the following *types*:

ACTIVESET < (suboptions) >

uses the active set algorithm. You can specify one or more of the following *suboptions* in parentheses:

FEASTOL=number

defines the feasible tolerance. The optimizer exits if the constraint violation is less than *number* and the scaled optimality conditions are less than the value of the OPTTOL= suboption. By default, FEASTOL= 10^{-6} .

MSRANGE=number

defines the range from which each variable can take values during the sampling process in multistart mode, where *number* is a positive number. This option is ignored if MULTI-START=0. This option affects only the sampling process that determines starting points for the local solver. It does not affect the bounds of the original nonlinear optimization problem. By default, MSRANGE=2.

MULTISTART=0 | 1 | (suboptions)

specifies whether to enable multistart mode or specifies *suboptions* to enable the distributed multistart mode. You can specify the following values:

- 0** disables multistart mode.
- 1** solves the problem from multiple starting points, possibly finding a better local optimum as a result. The computing cost in multistart mode might be huge.

By default, MULTISTART=0.

The following *suboptions*, which are specified in parentheses, are supported only in distributed (MPP) mode. If you specify empty parentheses, these *suboptions* are assigned their default values.

FESESSIONS=number

specifies the number of CAS subsessions that are dedicated to the function evaluations, where *number* is a positive integer. The default value is based on the setting of the CAS session.

FESESSNODES=number

specifies the number of computing nodes in each CAS subsession that are dedicated to the function evaluations, where *number* is a positive integer. The default value is based on the setting of the CAS session.

MAXSTARTS=number

specifies the maximum number of starting points to be used by the multistart algorithm, where *number* is a positive integer. By default, MAXSTARTS=100.

SAMPSELRATIO=number

specifies the ratio of the total number of sample points to be generated to the value of the MAXSTARTS= option, where *number* is a positive integer. By default, SAMPSELRATIO=20.

SESSNODES=number

specifies the number of computing nodes in the CAS session that are dedicated to the multistart algorithm, where *number* is a positive integer. The default value is based on the setting of the CAS session.

OBJLIMIT=number

specifies an upper limit on the magnitude of the objective value, where *number* is greater than or equal to 10^8 . If OBJLIMIT= M , the algorithm terminates when the objective value becomes less than $-M$ for a minimization problem or when the objective value exceeds M for a maximization problem. The termination of the algorithm implies that either the problem is unbounded or the algorithm diverges. If optimization were allowed to continue, numerical difficulty might be encountered. By default, OBJLIMIT= 10^{20} . If *number* is less than 10^8 , it is reset to the default value of 10^{20} .

OPTTOL= ϵ

specifies the measure by which the current iterate is determined to be an acceptable approximation of a local optimum, where ϵ is a positive real number. The optimizer determines that the current iterate is a local optimum when the norm of the scaled vector of the optimality conditions is less than ϵ and the true constraint violation is less than the value of the FEASTOL= option. By default, OPTTOL= 10^{-6} .

SOLTYPE=0 | 1

specifies the type of solution that the optimizer should return. You can specify the following values:

- 0** requests that the optimizer return a locally optimal solution, provided that it locates one.
- 1** requests that the optimizer return the best feasible solution that was found, provided that its objective value is better than that of the locally optimal solution that was found.

By default, SOLTYPE=1.

EM <(suboption)>

uses the expectation-maximization (EM) algorithm. This algorithm is supported only for the Gaussian HMM and the maximum likelihood method. You can specify the following *suboption* in parentheses:

MINITER=number

specifies the minimum number of iterations in the optimization process, where *number* must be a nonnegative integer. By default, MINITER=0.

SINGLE

executes the EM algorithm in a single thread on a single node. By default, the EM algorithm runs in parallel on multiple threads and nodes if you divide the data into sections and specify the SECTION= option in the ID statement.

INTERIORPOINT <(suboptions)>

uses the interior point algorithm. You can specify the same *suboptions* in parentheses that you can specify for ALGORITHM=ACTIVESET.

IPDIRECT <(suboptions)>

uses the interior point direct algorithm, in which the use of direct factorizations and other enhancements can reduce both the number of iterations and the CPU time for many problem types. You can specify the same *suboptions* in parentheses that you can specify for ALGORITHM=INTERIORPOINT.

SGD <(suboptions)>

uses the stochastic gradient descent algorithm. You can specify one or more of the following *suboptions* in parentheses:

ADAPTIVEDECAY=number

specifies the adaptive rate for the gradient descent algorithm when it is used, where the algorithm becomes ADAGRAD if ADAPTIVEDECAY=0 and becomes ADADELTA otherwise. By default, ADAPTIVEDECAY=0.95.

ANNEALINGRATE=number

specifies the annealing parameter, β , where *number* must be a nonnegative double. Annealing is a way to automatically reduce the learning rate as SGD progresses, causing smaller steps as SGD approaches a solution. Effectively, this option replaces the learning rate parameter, η , with

$$\eta' = \frac{\eta}{1 + \beta t}$$

where t is the number of iterations that SGD has performed. By default, ANNEALINGRATE= 10^{-5} .

CHAIN=(options)

specifies the options for the sequence data (which constitute a chain). You can specify the following *options* within parentheses:

LENGTH=number

specifies the length of a chain in a minibatch in optimization process, where *number* must be a positive integer. By default, LENGTH=1000.

BUFFER=number

specifies the buffer length on the left side of a chain in a minibatch in optimization process, where *number* must be a nonnegative integer. By default, BUFFER=20.

LEARNINGRATE=number

specifies the learning rate parameter, η , for SGD, where *number* must be a positive double. New iterates for SGD are found by using

$$\theta_{k+1} = \theta_k - \frac{\eta}{\|I_k\|} \sum_{(x_i, y_i) \in I_k} \nabla L(\theta_k; x_i, y_i)$$

where θ_k is the current weight vector, θ_{k+1} is the new weight vector, I_k is the minibatch that is used during iteration k , and $L(\theta_k; x_i, y_i)$ is the negative log likelihood that is associated with the i th chain of observation. If you see a huge objective value with SGD (especially with a small data table), it is likely that too large a value was specified for this option. By default, LEARNINGRATE=10.

MINIBATCHSIZE=number

specifies the size of the minibatches used in SGD, where *number* must be a positive integer. By default, MINIBATCHSIZE=1.

MOMENTUM=number

specifies the value for momentum parameter α , where *number* must be greater than or equal to 0 and less than 1. New iterates for SGD that uses the momentum method are found by using

$$\begin{aligned} \theta_{k+1} &= \theta_k + \nabla \theta_k \\ \nabla \theta_k &= \alpha \nabla \theta_k - \frac{\eta}{\|I_k\|} \sum_{(x_i, y_i) \in I_k} \nabla L(\theta_k; x_i, y_i) \end{aligned}$$

where θ_k is the current weight vector, θ_{k+1} is the new weight vector, I_k is the minibatch that is used during iteration k , and $L(\theta_k; x_i, y_i)$ is the negative log likelihood that is associated with the i th chain of observation. By default, MOMENTUM=0.

NONADUPDATE

suppresses the adaptive gradient descent algorithm.

USELOCKING

requests that computational threads share a common weight vector and that they update the weight vector without race conditions. By default, computational threads update a single weight vector simultaneously. This causes intentional race conditions and nondeterministic behavior, but it increases performance significantly.

By default, ALGORITHM=ACTIVESET.

COVOFPARM=YES | NO

specifies whether to calculate the covariance of parameters. You can specify the following values:

YES calculates the covariance of parameters.

NO suppresses the calculation of the covariance of parameters.

By default, COVOFPARM=YES. This option supports only the EM algorithm.

MAXITER=number

specifies the maximum number of iterations in the optimization process, where *number* must be a nonnegative integer. By default, MAXITER=128.

MAXTIME=t

specifies an upper limit of *t* seconds for the optimization process, including problem generation time and solution time, where *t* can be any positive number. The default value of *t* is the positive number that has the largest absolute value that your operating environment can represent. If you omit this option, the optimizer does not stop on the basis of the amount of time elapsed. This option is not supported for the EM algorithm.

PRINTITERFREQ=number

specifies how frequently to output the optimization process history, where *number* must be a nonnegative integer. If PRINTITERFREQ=0, no iterations are output. If *number*=*n* and *n* is positive, the optimization process history is output every *n* iterations. By default, PRINTITERFREQ=0.

PRINTLEVEL=0 | 1 | 2 | 3

specifies the print level. You can specify the following values:

0 suppresses output of the initial parameter values, the initial objective value, the algorithm information, the final parameter values, and the final objective value.

1 outputs the initial parameter values and the initial objective value.

2 outputs the initial parameter values, the initial objective value, and the algorithm information.

3 outputs the initial parameter values, the initial objective value, the algorithm information, the final parameter values, and the final objective value.

By default, PRINTLEVEL=0.

PRIOR Statement

PRIOR *distribution* < , . . . , *distribution* > ;

The PRIOR statement specifies the initial hyperparameter values in the prior distributions of the parameters when the maximum a posteriori method is applied to the estimation of the HMM (that is, when METHOD=MAP in the MODEL statement). If you specify the maximum likelihood (ML) method (METHOD=ML), the PRIOR statement is ignored. You can specify only one PRIOR statement. If you specify more than one *distribution*, separate them with commas.

If you specify the NSTATE= n_1 : n_2 option or the YLAG= m_1 : m_2 option (or both) in the MODEL statement to estimate multiple models, the PRIOR statement is applied to the first model that corresponds to the specification of NSTATE= n_1 or YLAG= m_1 (or that corresponds to both NSTATE= n_1 and YLAG= m_1 if both options are specified).

To use the PRIOR statement, you need to know the type of the model: different sets of prior distributions are available for different types of HMMs.

The *distribution* is in the following format:

parameter-name ~ *distribution-function*(*arguments*) ;

You can specify the following *parameter-names*:

CPM	indicates the category probability matrix in the finite HMM. The prior distribution must be the Dirichlet distribution.
ISPV	indicates the initial state probability vector. The prior distribution must be the Dirichlet distribution.
LAMBDA	indicates the mean vector in the Poisson HMM. The prior distribution must be the gamma distribution.
MCP	indicates the mixture component probabilities in the Gaussian mixture HMM (GM HMM). The prior distribution must be the Dirichlet distribution.
MUSIGMA	indicates the mean and covariance parameters for the observation distributions. For the Gaussian HMM, regime-switching regression model, or regime-switching autoregression model, the prior distribution of the mean and covariance parameters for each state must be the normal-inverse-Wishart (NIW) distribution. For the GM HMM, the prior distribution of the mean and covariance parameters for each component at each state must be the normal-inverse-Wishart (NIW) distribution.
TPM	indicates the transition probability matrix. The prior distribution must be the Dirichlet distribution.

You can specify the following *distribution-functions*:

DIR	specifies the Dirichlet distribution. There is one <i>argument</i> for the Dirichlet distribution, and it must be a valid matrix expression of the proper size. For example, if the Dirichlet distribution is the prior distribution of the ISPV, MCP, CPM or TPM, the size of the <i>argument</i> must be the same as the size of the ISPV, MCP, CPM or TPM. All elements of the <i>argument</i> should be positive. If you specify any nonpositive element in the <i>argument</i> ,
------------	---

the nonpositive element is changed to the machine precision. For information about how to define the matrix expressions, see the section “[Matrix Expression](#)” on page 1311. For more information about the Dirichlet distribution, see the section “[Maximum a Posteriori \(MAP\) Method](#)” on page 1304.

GAMMA

specifies the gamma distribution. There are two *arguments* (α, β) for the gamma distribution, and they must be a valid matrix expression of the proper size. The size of the *arguments* must be a $K \times 2$ matrix. All elements of the *arguments* should be positive. If you specify any nonpositive element in the *argument*, the nonpositive element is changed to the machine precision. For information about how to define the matrix expressions, see the section “[Matrix Expression](#)” on page 1311. For more information about the gamma distribution, see the section “[Maximum a Posteriori \(MAP\) Method](#)” on page 1304.

NIW

specifies the normal-inverse-Wishart (NIW) distribution. There are four *arguments* for the NIW distribution ($\tilde{\mu}$, $\tilde{\Sigma}$, $\tilde{\kappa}$, and $\tilde{\nu}$), and they must be valid matrix expressions of the proper size. If the NIW distribution is the prior distribution of the MUSIGMA for Gaussian HMM, then the following statements are true:

- $\tilde{\mu}$ is a $K \times k_y$ matrix, where K is the number of states, and k_y is the number of dependent variables.
- $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix.
- $\tilde{\kappa}$ is a $K \times 1$ vector.
- $\tilde{\nu}$ is a $K \times 1$ vector.

If the NIW distribution is the prior distribution of the MUSIGMA for GM HMM, then the following statements are true:

- $\tilde{\mu}$ is a $KM \times k_y$ matrix, where K is the number of states, M is the number of components at a state, and k_y is the number of dependent variables.
- $\tilde{\Sigma}$ is a $k_y KM \times k_y$ matrix.
- $\tilde{\kappa}$ is a $KM \times 1$ vector.
- $\tilde{\nu}$ is a $KM \times 1$ vector.

If the NIW distribution is the prior distribution of the MUSIGMA for regime-switching regression or regime-switching autoregression, then the following statements are true:

- $\tilde{\mu}$ is a $K \times k_y k_z$ matrix, where K is the number of states, k_y is the number of dependent variables, and k_z is the number of regressors in each equation. (For more information about how to calculate k_z , see the sections “[Regime-Switching Regression Model](#)” on page 1273 and “[Regime-Switching Autoregression Model](#)” on page 1274.)
- $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix.
- $\tilde{\kappa}$ is a $k_z K \times k_z$ matrix.
- $\tilde{\nu}$ is a $K \times 1$ vector.

For information about how to define the matrix expressions, see the section “[Matrix Expression](#)” on page 1311. For more information about the normal-inverse-Wishart (NIW) distribution, see the section “[Maximum a Posteriori \(MAP\) Method](#)” on page 1304.

The following statements specify the PRIOR statement for the ISPV, TPM, and MUSIGMA in a bivariate three-state Gaussian HMM:

```

proc hmm data=mylib.One;
  id time=t;
  model x y / type=gaussian nstate=3 estispv method=map;
  prior ISPV ~ dir(J(3,1,1)),
          TPM ~ dir(I(3)*100+J(3,3,1)),
          MUSIGMA ~ niw(J(1,2,-1) // J(1,2,0) // J(1,2,1),
                        J(3,1,1)@(I(2)*10),
                        5*J(3,1,1),
                        10*J(3,1,1));
run;

```

SCORE Statement

SCORE *options* ;

The SCORE statement specifies options that are related to scoring data. You can specify the following *options*:

INMODEL=*libref.data-table*

specifies the input data table for model information. The data table must be an output data table that is produced by the OUTMODEL= option in a previous call of the HMM procedure. When you specify this option, the IN= option in the ESTIMATE statement and the ID, INITIAL, MODEL, OPTIMIZE, and PRIOR statements are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

OUTMODEL=*libref.data-table*

writes the model information to the specified output data table. The output data table consists of binary large object columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

SMOOTH Statement

SMOOTH *option* ;

The SMOOTH statement specifies options that are related to the smoothing problem, which is to find the probability distribution of the state, conditional on all available observations. You can specify the following *option*:

OUT=*libref.data-table*

writes the smoothing results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

STORE Statement

STORE < **OUT=** >*libref.data-table* ;

The STORE statement saves the model information to the specified output data table that can be processed by the ASTORE procedure. An example of using analytic store technology is to perform a time-consuming model learning (that is, parameter estimation) and to store the learned model by using the STORE statement. Later you can perform the inference (namely, evaluating, filtering, smoothing, decoding, and forecasting) based on the saved model from the previous analysis, without having to learn the model again. For more information about the ASTORE procedure, see the chapter “The ASTORE Procedure” in *SAS Viya: Machine Learning Procedures*. The output data table consists of binary large object columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section “Using CAS Sessions and CAS Engine Librefs” on page 1186.

Details: HMM Procedure

Hidden Markov Model

This section is a brief review of the hidden Markov model. For more information, see Rabiner (1989), Krolzig (1997), Frühwirth-Schnatter (2006), and Cappé, Moulines, and Rydén (2010).

The hidden Markov model (HMM) is a bivariate discrete time process $\{\mathbf{S}_t, \mathbf{Y}_t\}$, where $\{\mathbf{S}_t\}$ is a (hidden) Markov chain; conditional on $\{\mathbf{S}_t\}$, the observable process $\{\mathbf{Y}_t\}$ is a sequence of independent random variables such that the conditional distribution of $\mathbf{Y}_t$ depends only on $\mathbf{S}_t$. The $\mathbf{S}_t$ is often called the state. In general, the HMM can be expressed in three equations:

1. Initialization equation: $\mathbf{S}_1 \sim h_\theta(\cdot)$
2. Transition equation: $\mathbf{S}_t \sim f_\theta(\cdot|\mathbf{S}_{t-1})$
3. Observation equation: $\mathbf{Y}_t \sim g_\theta(\cdot|\mathbf{S}_t)$

Here the initial probability distribution function $h_\theta(\cdot)$ is the probability distribution function for initial state $\mathbf{S}_1$; the transition probability distribution function $f_\theta(\cdot|\mathbf{S}_{t-1})$ is the probability distribution function of current state $\mathbf{S}_t$ conditional on the past state $\mathbf{S}_{t-1}$; the observation probability distribution function $g_\theta(\cdot|\mathbf{S}_t)$ is the probability distribution function of current observation $\mathbf{Y}_t$ conditional on the current state $\mathbf{S}_t$; and all probability distribution functions might depend on the parameter θ .

There is no restriction on the dimensionality of $\mathbf{S}_t$ and $\mathbf{Y}_t$; that is, they might be scalars, vectors, or matrices.

If the transition probability distribution function $f_\theta(\cdot|\cdot)$ is time-independent (that is, the Markov chain is time-homogeneous), then the HMM is called the (time-)homogeneous HMM; otherwise, the HMM is called the (time-)inhomogeneous HMM.

The state $\mathbf{S}_t$ could take discrete or continuous values or even a mix of discrete and continuous values.² In the simplest and most popular case, $\mathbf{S}_t$ is a scalar and takes finite discrete values; that is, $\mathbf{S}_t \in \{1, \dots, K\}$. In this case, the HMM could be called the finite-state-space HMM.

For the finite-state-space HMM, the initial state distribution function $h_\theta(\cdot)$ can be expressed as a $K \times 1$ vector π , which is called the initial state probability vector (ISPV); that is,

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

For the finite-state-space and homogeneous HMM, the transition probability distribution function $f_\theta(\cdot|\cdot)$ can be expressed as a $K \times K$ matrix $\mathbf{A}$, which is called the transition probability matrix (TPM), where an element a_{ij} of $\mathbf{A}$ denotes the probability transitioning from past state i to current state j ; that is,

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

There are six common problems to solve for an HMM:

²In some literature, the HMM is referred to only as the discrete-state-space HMM (that is, the state can take only discrete values). The name “state space model” is used for the continuous-state-space HMM (that is, the state can take only continuous values).

- Filtering problem: What is $p(\mathbf{S}_t | \mathbf{Y}_1, \dots, \mathbf{Y}_t)$, the probability distribution of state $\mathbf{S}_t$ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_t$?
- Smoothing problem: What is $p(\mathbf{S}_t | \mathbf{Y}_1, \dots, \mathbf{Y}_T)$, the probability distribution of state $\mathbf{S}_t$ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$, where T is the sample size?
- Forecasting problem: What is $p(\mathbf{S}_{t+h} | \mathbf{Y}_1, \dots, \mathbf{Y}_t)$, $h > 0$, the probability distribution of state $\mathbf{S}_{t+h}$ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_t$?
- Evaluating problem: What is $p(\mathbf{Y}_1, \dots, \mathbf{Y}_T)$, the probability (or the likelihood) of observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$?
- Decoding problem: What is $\arg \max_{\mathbf{S}_1, \dots, \mathbf{S}_T} p(\mathbf{S}_1, \dots, \mathbf{S}_T, \mathbf{Y}_1, \dots, \mathbf{Y}_T)$, the most likely sequence of hidden states $\mathbf{S}_1, \dots, \mathbf{S}_T$ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$, where T is the sample size?
- Learning problem: How do you estimate the unknown parameter θ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$, where T is the sample size?

In most cases, the learning problem is the most difficult one. You might apply the maximum likelihood (ML) method or the maximum a posteriori (MAP) method to get the point estimate of parameter θ . For more information, see the section “[Parameter Estimation Methods](#)” on page 1304.

Several algorithms are helpful in solving the problems for the HMMs. Three of them are the forward algorithm, the backward algorithm, and the Viterbi algorithm. They might be generalized to support many types of HMMs. The following sections focus on applying them to the finite-state-space and homogeneous HMM.

Forward Algorithm

The forward algorithm recursively calculates the joint probability distribution of a state at time t , $\mathbf{S}_t$, and all observations up to time t , $\mathbf{Y}_1, \dots, \mathbf{Y}_t$. First, define $\alpha_t(i)$, $i = 1, \dots, K$, as the joint probability of $(\mathbf{S}_t = i)$ and $\mathbf{Y}_1, \dots, \mathbf{Y}_t$:

$$\alpha_t(i) \equiv p(\mathbf{S}_t = i, \mathbf{Y}_1, \dots, \mathbf{Y}_t)$$

Then, $\alpha_t(i)$ can be recursively calculated by the following steps:

1. Calculate $\alpha_1(i)$, $i = 1, \dots, K$, by

$$\alpha_1(i) = \pi_i g_\theta(\mathbf{Y}_1 | \mathbf{S}_1 = i)$$

2. Calculate $\alpha_t(i)$, $i = 1, \dots, K$, $t = 2, \dots, T$, by

$$\alpha_t(i) = \left(\sum_{j=1}^K \alpha_{t-1}(j) a_{ji} \right) g_\theta(\mathbf{Y}_t | \mathbf{S}_t = i)$$

You can solve the evaluating problem by

$$L_t \equiv p(\mathbf{Y}_1, \dots, \mathbf{Y}_t) = \sum_{j=1}^K \alpha_t(j)$$

Because the likelihood of all observations, L_T , can be calculated, you can apply the maximum likelihood method to estimate the unknown parameter θ ; if the prior distribution of the parameter θ is given, you can also apply the MAP method. That is, you might use the forward algorithm in the learning problem.

You can solve the filtering problem by

$$p_t^{(f)}(i) \equiv p(\mathbf{S}_t = i | \mathbf{Y}_1, \dots, \mathbf{Y}_t) = \alpha_t(i) / L_t$$

When the filtering problem is solved, you can solve the forecast problem, because the h -step-ahead prediction of the state probability can be calculated by

$$p_{t+h|t}^{(p)} = (\mathbf{A}')^h p_t^{(f)}$$

where $p_t^{(f)}$ is the $K \times 1$ vector with the i th element as $p_t^{(f)}(i)$, and $p_{t+h|t}^{(p)}$ is the $K \times 1$ vector with the i th element as $p_{t+h|t}^{(p)}(i)$:

$$p_{t+h|t}^{(p)}(i) \equiv p(\mathbf{S}_{t+h} = i | \mathbf{Y}_1, \dots, \mathbf{Y}_t)$$

The distribution of $\mathbf{Y}_{t+h}$ conditional on observations $\mathbf{Y}_1, \dots, \mathbf{Y}_t$ is the following mixture distribution:

$$\mathbf{Y}_{t+h} | \mathbf{Y}_1, \dots, \mathbf{Y}_t \sim \sum_{i=1}^K p_{t+h|t}^{(p)}(i) g_{\theta}(\cdot | \mathbf{S}_{t+h} = i)$$

Backward Algorithm

To solve the smoothing problem, in addition to the forward algorithm, you also need the backward algorithm. Define $\beta_t(i)$ as the probability of observing the future observations $\mathbf{Y}_{t+1}, \dots, \mathbf{Y}_T$ conditional on the current state $\mathbf{S}_t = i$:

$$\beta_t(i) \equiv p(\mathbf{Y}_{t+1}, \dots, \mathbf{Y}_T | \mathbf{S}_t = i)$$

The backward algorithm recursively calculates $\beta_t(i)$ as follows:

1. Set $\beta_T(i) = 1, i = 1, \dots, K$.
2. Calculate $\beta_t(i), i = 1, \dots, K, t = T - 1, \dots, 1$, by

$$\beta_t(i) = \sum_{j=1}^K a_{ij} g_{\theta}(\mathbf{Y}_{t+1} | \mathbf{S}_{t+1} = j) \beta_{t+1}(j)$$

When you calculate both $\alpha_t(i)$ and $\beta_t(i)$, $i = 1, \dots, K$, $t = 1, \dots, T$, by using the forward algorithm and backward algorithm, the smoothing problem is solved, because you can calculate the probability of state $\mathbf{S}_t$ taking value i given all observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$ as follows:

$$\gamma_t(i) \equiv p(\mathbf{S}_t = i | \mathbf{Y}_1, \dots, \mathbf{Y}_T) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^K \alpha_t(j)\beta_t(j)}$$

Viterbi Algorithm

You use the Viterbi algorithm to solve the decoding problem. First, define the highest probability of a single state path ending in state $\mathbf{S}_t = i$ and observations $\mathbf{Y}_1, \dots, \mathbf{Y}_t$ as follows:

$$V_t(i) = \max_{\mathbf{S}_1, \dots, \mathbf{S}_{t-1}} p(\mathbf{S}_1, \dots, \mathbf{S}_{t-1}, \mathbf{S}_t = i, \mathbf{Y}_1, \dots, \mathbf{Y}_t)$$

To keep track of the best path, for $t = 2, \dots, T$, define

$$v_t(i) = \arg \max_j V_{t-1}(j) a_{ji}$$

Then, $V_t(i)$ and $v_t(i)$ can be recursively calculated as follows:

1. Calculate $V_1(i)$, $i = 1, \dots, K$, by

$$V_1(i) = \pi_i g_\theta(\mathbf{Y}_1 | \mathbf{S}_1 = i)$$

2. Calculate $V_t(i)$ and $v_t(i)$, $i = 1, \dots, K$, $t = 2, \dots, T$, by

$$\begin{aligned} V_t(i) &= \left(\max_j V_{t-1}(j) a_{ji} \right) g_\theta(\mathbf{Y}_t | \mathbf{S}_t = i) \\ v_t(i) &= \arg \max_j V_{t-1}(j) a_{ji} \end{aligned}$$

Then, the probability of the best path is

$$V^* = \max_i V_T(i)$$

Define $v_T^* = \arg \max_i V_T(i)$, and the best state path can be backtracked by

$$v_t^* = v_{t+1}(v_{t+1}^*), t = T-1, \dots, 1$$

That is, $\{v_t^*\}_{t=1, \dots, T}$ is the best state path. The decoding problem is solved.

Gaussian Hidden Markov Model

The Gaussian hidden Markov model (Gaussian HMM) is a type of finite-state-space and homogeneous HMM where the observation probability distribution is the normal distribution,

$$\mathbf{Y}_t | \mathbf{S}_t \sim N(\mu_{\mathbf{S}_t}, \Sigma_{\mathbf{S}_t})$$

where $\mu_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters at state $\mathbf{S}_t$, $\mathbf{S}_t = 1, \dots, K$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{\mu_i, \Sigma_i\}_{i=1, \dots, K}$, which consists of mean and covariance parameters) together specify the Gaussian HMM; that is, the parameter θ of the Gaussian HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the Gaussian HMM is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section “[Hidden Markov Model](#)” on page 1268. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Gaussian Mixture Hidden Markov Model

The Gaussian mixture hidden Markov model (GM HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the Gaussian mixture distribution,

$$\mathbf{Y}_t | \mathbf{S}_t \sim \text{GM}(\{w_{\mathbf{S}_t,1}, \dots, w_{\mathbf{S}_t,M}\}, \{\mu_{\mathbf{S}_t,1}, \dots, \mu_{\mathbf{S}_t,M}\}, \{\Sigma_{\mathbf{S}_t,1}, \dots, \Sigma_{\mathbf{S}_t,M}\})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\text{GM}(\dots)$ represents the Gaussian mixture distribution; M is the number of components at each state; and the $w_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_{\mathbf{S}_t,j} \geq 0$, $j = 1, \dots, M$, and $\sum_{j=1}^M w_{\mathbf{S}_t,j} = 1$). The $\mu_{\mathbf{S}_t,j}$ and $\Sigma_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are the mean and covariance parameters for the j th Gaussian component at state $\mathbf{S}_t$, and their values depend on the variable $\mathbf{S}_t$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{w_{ij}, \mu_{ij}, \Sigma_{ij}, i = 1, \dots, K, j = 1, \dots, M\}$, which consists of the mixture component probabilities, the mean and covariance parameters for all components) together specify the GM HMM; that is, the parameter θ of the GM HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the GM HMM is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms that are introduced in the section “[Hidden Markov Model](#)” on page 1268. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Regression Model

The regime-switching regression model (RS REG, also known as the regression hidden Markov model or REG HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the normal distribution conditional on the specified regressors,

$$\mathbf{Y}_t | z_t, \mathbf{S}_t \sim N(\mathbf{B}_{\mathbf{S}_t} z_t, \Sigma_{\mathbf{S}_t})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\mathbf{B}_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are the mean and covariance parameters, respectively, at state $\mathbf{S}_t$, $\mathbf{S}_t = 1, \dots, K$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{\mathbf{B}_i, \Sigma_i\}_{i=1, \dots, K}$, which consists of mean and covariance parameters) together specify the RS REG; that is, the parameter θ of the RS REG is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t , might contain the constant, seasonal dummies, linear time trend, quadratic time trend, and exogenous variables and their lagged values. That is,

$$z_t = \left(\left(z_t^{(\text{CONST})} \right)' \left(z_t^{(\text{SD})} \right)' \left(z_t^{(\text{LTREND})} \right)' \left(z_t^{(\text{QTREND})} \right)' \left(z_t^{(\text{XL})} \right)' \right)'$$

where

- $z_t^{(\text{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\text{CONST})} = (1)$.
- $z_t^{(\text{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\text{SD})} = (sd_t^{(1)} \dots sd_t^{(n_s-1)})'$, where $sd_t^{(i)}$, $i = 1, \dots, n_s - 1$, denote the values of seasonal dummies at time t and where n_s is the value specified in the NSEASON= n_s option.
- $z_t^{(\text{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\text{LTREND})} = (t)$.
- $z_t^{(\text{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\text{QTREND})} = (t^2)$.
- $z_t^{(\text{XL})}$ is empty if no exogenous variables are specified in the model, or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\text{XL})}$ is determined as follows, where XLAG= s and k_x is the number of exogenous variables:
 - If the NOCURRENTX option is specified,

$$z_t^{(\text{XL})} = (x_{1,t-1} \dots x_{k_x,t-1} \dots x_{1,t-s} \dots x_{k_x,t-s})'$$

- If the NOCURRENTX option is not specified,

$$z_t^{(\text{XL})} = (x_{1,t} \dots x_{k_x,t} \ x_{1,t-1} \dots x_{k_x,t-1} \dots x_{1,t-s} \dots x_{k_x,t-s})'$$

The statement

```
model y = x1 x2 x3 / type=reg nstate=2;
```

defines the regression for state $i, i = 1, \dots, K$ ($K = 2$ in this case), as

$$\begin{aligned} y_t &= b_{i,1} + b_{i,2}x1_t + b_{i,3}x2_t + b_{i,4}x3_t + \varepsilon_t \\ &= (b_{i,1} \ b_{i,2} \ b_{i,3} \ b_{i,4})z_t + \varepsilon_t \end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_i)$, $z_t = (1 \ x1_t \ x2_t \ x3_t)'$, and $k_z = 4$.

In the following statement, $z_t = (1 \ sd_t^{(1)} \ sd_t^{(2)} \ sd_t^{(3)} \ t \ t^2 \ x1_t \ x2_t \ x3_t \ x1_{t-1} \ x2_{t-1} \ x3_{t-1})'$, where $sd_t^{(i)}, i = 1, 2, 3$, denote the values of seasonal dummies at time t and where $k_z = 12$:

```
model y = x1 x2 x3 / trend=quad nseason=4 xlag=1 type=reg nstate=2;
```

In the following statement, $z_t = (x1_{t-1} \ x2_{t-1} \ x3_{t-1})'$ and $k_z = 3$:

```
model y = x1 x2 x3 / noint xlag=1 nocurrentx type=reg nstate=2;
```

The number of dependent variables, k_y , can be greater than 1; in that case, $\mathbf{B}_i, i = 1, \dots, K$, is the $k_y \times k_z$ matrix, and $\Sigma_i, i = 1, \dots, K$, is the $k_y \times k_y$ symmetric positive definite matrix.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, and **XL** functions refer to the mean parameters that correspond to $z_t^{(\text{CONST})}$, $z_t^{(\text{SD})}$, $z_t^{(\text{LTREND})}$, $z_t^{(\text{QTREND})}$, and $z_t^{(\text{XL})}$, respectively; the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1 \ \dots \ \Sigma_K)'$, where K is the number of states.

The Gaussian HMM can be regarded as a special case of the RS REG when there is only one regressor constant.

Because the RS REG is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section “[Hidden Markov Model](#)” on page 1268. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Autoregression Model

The regime-switching autoregression model (RS AR, also known as the autoregressive hidden Markov model or AR HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the normal distribution conditional on the specified regressors. For a standard form of RS AR,

$$\mathbf{Y}_t | z_t, \mathbf{S}_t \sim N(\mathbf{B}_{\mathbf{S}_t} z_t, \Sigma_{\mathbf{S}_t})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\mathbf{B}_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters at state $\mathbf{S}_t, \mathbf{S}_t = 1, \dots, K$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the parameter $\mathbf{B} (\equiv \{\mathbf{B}_i, \Sigma_i\}_{i=1, \dots, K})$, which consists of mean and covariance parameters) together specify the RS AR; that is, the parameter θ of the RS AR is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t , might contain the constant, seasonal dummies, linear time trend, quadratic time trend, exogenous variables and their lagged values, and the lagged values of the dependent variables. That is,

$$z_t = \left(\left(z_t^{(\text{CONST})} \right)' \left(z_t^{(\text{SD})} \right)' \left(z_t^{(\text{LTREND})} \right)' \left(z_t^{(\text{QTREND})} \right)' \left(z_t^{(\text{XL})} \right)' \left(z_t^{(\text{AR})} \right)' \right)'$$

where

- $z_t^{(\text{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\text{CONST})} = (1)$.
- $z_t^{(\text{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\text{SD})} = (sd_t^{(1)} \dots sd_t^{(n_s-1)})'$, where $sd_t^{(i)}, i = 1, \dots, n_s - 1$, denote the values of seasonal dummies at time t and where n_s is the value specified in the NSEASON= n_s option.
- $z_t^{(\text{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\text{LTREND})} = (t)$.
- $z_t^{(\text{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\text{QTREND})} = (t^2)$.
- $z_t^{(\text{XL})}$ is empty if no exogenous variables are specified in the model, or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\text{XL})}$ is determined as follows, where XLAG= s and k_x is the number of exogenous variables:
 - If the NOCURRENTX option is specified,

$$z_t^{(\text{XL})} = (x_{1,t-1} \dots x_{k_x,t-1} \dots x_{1,t-s} \dots x_{k_x,t-s})'$$

- If the NOCURRENTX option is not specified,

$$z_t^{(\text{XL})} = (x_{1,t} \dots x_{k_x,t} x_{1,t-1} \dots x_{k_x,t-1} \dots x_{1,t-s} \dots x_{k_x,t-s})'$$

- $z_t^{(\text{AR})}$ is empty if the YLAG= option is not specified or YLAG=0 is specified; otherwise, $z_t^{(\text{AR})}$ is determined as follows, where YLAG= p and k_y is the number of dependent variables:

$$z_t^{(\text{AR})} = (y_{1,t-1} \dots y_{k_y,t-1} \dots y_{1,t-p} \dots y_{k_y,t-p})'$$

The statement

```
model y = x1 x2 x3 / ylag=2 type=ar nstate=2;
```

defines the autoregression for state $i, i = 1, \dots, K$ ($K = 2$ in this case), as

$$\begin{aligned} y_t &= b_{i,1} + b_{i,2}x_{1t} + b_{i,3}x_{2t} + b_{i,4}x_{3t} + b_{i,5}y_{t-1} + b_{i,6}y_{t-2} + \varepsilon_t \\ &= (b_{i,1} \ b_{i,2} \ b_{i,3} \ b_{i,4} \ b_{i,5} \ b_{i,6})z_t + \varepsilon_t \end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_i)$, $z_t = (1 \ x_{1t} \ x_{2t} \ x_{3t} \ y_{t-1} \ y_{t-2})'$, and $k_z = 6$.

In the following statement, $z_t = (1 \ sd_t^{(1)} \ sd_t^{(2)} \ sd_t^{(3)} \ t \ t^2 \ x_{1t} \ x_{2t} \ x_{3t} \ x_{1,t-1} \ x_{2,t-1} \ x_{3,t-1} \ y_{t-1} \ y_{t-2})'$, where $sd_t^{(i)}, i = 1, 2, 3$, denote the values of seasonal dummies at time t , and $k_z = 14$:

```
model y = x1 x2 x3 / ylag=2 trend=quad nseason=4 xlag=1 type=reg nstate=2;
```

In the following statement, $z_t = (x1_{t-1} \ x2_{t-1} \ x3_{t-1} \ y1_{t-1} \ y2_{t-1} \ y1_{t-2} \ y2_{t-2})'$ and $k_z = 7$:

```
model y1 y2 = x1 x2 x3 / ylag=2 noint xlag=1 nocurrentx type=reg nstate=2;
```

The number of dependent variables, k_y , can be greater than 1 (for example, $k_y = 2$ in the previous statement); in that case, $\mathbf{B}_i, i = 1, \dots, K$, is the $k_y \times k_z$ matrix, and $\Sigma_i, i = 1, \dots, K$, is the $k_y \times k_y$ symmetric positive definite matrix. When $k_y > 1$, the model is called the vector autoregressive hidden Markov model (VAR HMM) in some literature.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, **XL**, and **AR** functions refer to the mean parameters that correspond to $z_t^{(\text{CONST})}$, $z_t^{(\text{SD})}$, $z_t^{(\text{LTREND})}$, $z_t^{(\text{QTREND})}$, $z_t^{(\text{XL})}$, and $z_t^{(\text{AR})}$, respectively; the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1 \ \dots \ \Sigma_K)'$, where K is the number of states.

The RS REG can be regarded as a special case of the RS AR when YLAG=0 is specified.

Because the RS AR is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section “[Hidden Markov Model](#)” on page 1268. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Mean-Adjusted Autoregression Model

The regime-switching mean-adjusted autoregression model (also known as the mean-adjusted autoregressive hidden Markov model) is a type of finite-state-space and homogeneous HMM. For a regime-switching mean-adjusted autoregression model, the observation $\mathbf{Y}_t$ follows the normal distribution as

$$\mathbf{Y}_t | \mathbf{Y}_t, \dots, \mathbf{Y}_{t-p}, z_t, \dots, z_{t-p}, \mathbf{S}_t, \mathbf{S}_{t-1}, \dots, \mathbf{S}_{t-p} \sim N(\mathbf{B}_{\mathbf{S}_t}^{(R)} z_t + \sum_{\tau=1}^p \mathbf{B}_{\mathbf{S}_{t-\tau}}^{(A)} (\mathbf{Y}_{t-\tau} - \mathbf{B}_{\mathbf{S}_{t-\tau}}^{(R)} z_{t-p}), \Sigma_{\mathbf{S}_t})$$

where p is the autoregressive order, the variable $\mathbf{S}_t$ is the (hidden) state, $\mathbf{B}_{\mathbf{S}_{t-\tau}}^{(A)}$ is the autoregressive part of mean parameters at state $\mathbf{S}_t$, $\mathbf{B}_{\mathbf{S}_t}^{(R)}$ is the remaining mean parameters at state $\mathbf{S}_t$, and $\Sigma_{\mathbf{S}_t}$ is the covariance parameters at state $\mathbf{S}_t$ for $\mathbf{S}_t = 1, \dots, K$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the parameter $\mathbf{B} (\equiv \{\mathbf{B}_i^{(A)}, \mathbf{B}_i^{(R)}, \Sigma_i\}_{i=1, \dots, K})$ together specify the regime-switching mean-adjusted autoregression model; that is, the parameter θ of the regime-switching mean-adjusted autoregression model is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t , might contain the constant, seasonal dummies, linear time trend, quadratic time trend, exogenous variables. That is,

$$z_t = \left(\left(z_t^{(\text{CONST})} \right)' \left(z_t^{(\text{SD})} \right)' \left(z_t^{(\text{LTREND})} \right)' \left(z_t^{(\text{QTREND})} \right)' \left(z_t^{(\text{XL})} \right)' \right)'$$

where

- $z_t^{(\text{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\text{CONST})} = (1)$.
- $z_t^{(\text{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\text{SD})} = (sd_t^{(1)} \cdots sd_t^{(n_s-1)})'$, where $sd_t^{(i)}, i = 1, \dots, n_s - 1$, denote the values of the seasonal dummies at time t and where n_s is the value specified in the NSEASON= n_s option.
- $z_t^{(\text{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\text{LTREND})} = (t)$.
- $z_t^{(\text{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\text{QTREND})} = (t^2)$.
- $z_t^{(\text{XL})}$ is empty if no exogenous variables are specified in the model or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\text{XL})}$ is determined as follows, where XLAG= s and k_x is the number of exogenous variables:

– If the NOCURRENTX option is specified,

$$z_t^{(\text{XL})} = (x_{1,t-1} \cdots x_{k_x,t-1} \cdots x_{1,t-s} \cdots x_{k_x,t-s})'$$

– If the NOCURRENTX option is not specified,

$$z_t^{(\text{XL})} = (x_{1,t} \cdots x_{k_x,t} \ x_{1,t-1} \cdots x_{k_x,t-1} \cdots x_{1,t-s} \cdots x_{k_x,t-s})'$$

The ARMEAN=ADJUSTED option in the following statement specifies the following regime-switching mean-adjusted autoregression model:

```
model y = x / armean=adjusted ylag=1 type=ar nstate=2;
```

$$\begin{aligned} y_t &= c\mathbf{S}_t + b\mathbf{S}_t x_t + a\mathbf{S}_t (y_{t-1} - c\mathbf{S}_{t-1} - b\mathbf{S}_{t-1} x_{t-1}) + \varepsilon_t \\ &= (c\mathbf{S}_t \ b\mathbf{S}_t) z_t + a\mathbf{S}_t (y_{t-1} - (c\mathbf{S}_{t-1} \ b\mathbf{S}_{t-1}) z_{t-1}) + \varepsilon_t \\ &= \begin{cases} c_1 + b_1 x_t + a_1 (y_{t-1} - c_1 - b_1 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 1, \mathbf{S}_{t-1} = 1, \\ c_2 + b_2 x_t + a_2 (y_{t-1} - c_1 - b_1 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 2, \mathbf{S}_{t-1} = 1, \\ c_1 + b_1 x_t + a_1 (y_{t-1} - c_2 - b_2 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 1, \mathbf{S}_{t-1} = 2, \\ c_2 + b_2 x_t + a_2 (y_{t-1} - c_2 - b_2 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 2, \mathbf{S}_{t-1} = 2. \end{cases} \end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, $z_t = (1 \ x_t)'$, and $k_z = 2$.

The number of dependent variables, k_y , can be greater than 1; in that case, $\Sigma_i, i = 1, \dots, K$, is the $k_y \times k_y$ symmetric positive definite matrix. When $k_y > 1$, the model is called the vector autoregressive hidden Markov model (VAR HMM) in some literature.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, and **XL** functions refer to the mean parameters that correspond to $z_t^{(\text{CONST})}$, $z_t^{(\text{SD})}$, $z_t^{(\text{LTREND})}$, $z_t^{(\text{QTREND})}$, $z_t^{(\text{XL})}$, respectively; the **AR** function refers to the mean parameters that correspond to demeaned lagged dependent variables $(\mathbf{Y}_{t-\tau} - \mathbf{B}_{\mathbf{S}_{t-\tau}}^{(R)} z_{t-\tau})$, $\tau = 1, \dots, p$; and the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1 \cdots \Sigma_K)'$, where K is the number of states.

Because the regime-switching mean-adjusted autoregression model is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms that are introduced in the section “[Hidden Markov Model](#)” on page 1268. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Finite Hidden Markov Model

The finite hidden Markov model (finite HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the categorical distribution whose support is finite,

$$Y_t | \mathbf{S}_t \sim C(b_{\mathbf{S}_t,1}, \dots, b_{\mathbf{S}_t,M})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $C(\dots)$ represents the categorical distribution; M is the number of categories at each state; and the $b_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are category probabilities, satisfying ($b_{\mathbf{S}_t,j} \geq 0$, $j = 1, \dots, M$, and $\sum_{j=1}^M b_{\mathbf{S}_t,j} = 1$).

Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the probability of observational outcomes $\mathbf{B} (\equiv \{b_{ij}, i = 1, \dots, K, j = 1, \dots, M\})$ together specify the finite HMM; that is, the parameter θ of the finite HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the finite HMM is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms that are introduced in the section “[Hidden Markov Model](#)” on page 1268. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood. The dependent variable for the finite HMM must be an integer greater than or equal to 1 and less than or equal to the number of the category. The number of dependent variables for finite HMM must be 1. It is a trivial case when NSTATE=1 and NCATEGORY=1 for the finite HMM, and the program will stop processing.

The %FiniteHMM Macro

Although in theory, finite HMM assumes that the response variable is discrete (whether it is numeric or categorical), PROC HMM assumes that the response variable is composed of consecutive natural numbers that start with 1. However, for most real-world business data, the variable of interest could be recorded in other formats, such as categorical levels, decimals, negative numbers, and so on. In other cases, a response variable could be recorded as nonconsecutive natural numbers. Therefore it is likely that you will need to preprocess the data before using PROC HMM to perform finite HMM.

For example, if you are interested in modeling the dynamic feature of receiving promotion channels for a buyer, the response variable y_1 could have values such as ‘aa email’, ‘bb mail’, ‘cc phone’, ‘dd flyer’, ‘ee

seminar', and 'ff forum'. To use PROC HMM for finite HMM on y_1 , you would need to preprocess the data by creating a natural-number-valued variable y and model on y instead. If you assume that y_1 is in the data set `finite1`, the following SAS program does the preprocessing:

```
data finite1;
  set finite1;
  if y1='aa email'  ' then y=1;
  if y1='bb mail'   ' then y=2;
  if y1='cc phone'  ' then y=3;
  if y1='dd flyer'  ' then y=4;
  if y1='ee seminar' then y=5;
  if y1='ff forum'  ' then y=6;
run;
```

Then you can use PROC HMM for finite HMM on y as follows after you upload the data set to a SAS data library named `mylib`. Here the SAS data library `mylib` must already be defined.

```
ods output CPM=my_cpm_0
ParameterEstimates=my_est_0;
proc hmm data=mylib.finite1;
  id time=t section=section;
  model y / type=finite nstate=3 nCategory=6 method=MAP;
  estimate out=mylib.est_0 outall=mylib.estall_0;
  forecast out=mylib.fcst_0 lead=4;
run;
```

Because y has values 1 to 6, the output tables have information only about the levels of y , not the levels of y_1 . To understand and analyze the results for the original y_1 , you need to do postprocessing by manually matching the y levels to the levels of y_1 . This can be tedious if y_1 has too many levels. The SAS macro %FiniteHMM helps you do the preprocessing and postprocessing automatically in such cases. The %FiniteHMM macro is a wrapper for finite HMM with PROC HMM. It treats the response variable as a categorical variable, and it automatically preprocesses the data and postprocesses the results. This macro is useful in the following three cases:

- The response variable contains values other than natural numbers.
- The response variable is composed of nonconsecutive natural numbers.
- The response variable is composed of consecutive natural numbers but does not start with 1.

Be aware that there are limits to using the %FiniteHMM macro. In particular, it does not currently support n -literal response variable names and does not support interior quotation marks in response variable values.

Specifically, the %FiniteHMM macro does the following:

- treats the response variable for finite HMM as a categorical variable
- creates the internal response variable `_mody_` with consecutive natural numbers as its values, and links each natural number to the corresponding categorical level of the original response variable
- updates the `NCATEGORY=` option value to the actual number of levels of the original response variable

- applies PROC HMM to the internally created response variable `_mody_` for finite HMM
- for the ODS CPM output table and the output forecast table, updates the label of categorical columns with their corresponding levels in the original response variable
- for both the ODS table and the output parameter estimates table, adds a column named `RespVarLevel` that indicates the corresponding levels in the original response variable

These updated tables from the `%FiniteHMM` macro are ready for analysis on the original response variable because the parameters are linked to the original response variable levels through labels or through the additional column `RespVarLevel`. The ODS CPM output table and output forecast table will have labels of the variables `Category1` to `Categoryk` that are linked to the levels of the original variable. The parameter estimates table will have an additional column called `RespVarLevel` to indicate the levels of the original variable.

The parameters of the `%FiniteHMM` macro are defined to correspond to the statements in PROC HMM. The options in each statement of PROC HMM become the parameter values in the macro. Two additional parameters are added to the macro:

- `DATA=%str()`, which contains the name of the two-level input data set for PROC HMM
- `ODSOUT=%str()`, which contains the ODS table specification in the ODS OUTPUT statement of PROC HMM

Because the value of the `DATA=` option in PROC HMM is defined as a separate parameter in the `%FiniteHMM` macro, the PROC parameter in the macro contains only options other than the `DATA=` option of PROC HMM. To prevent ambiguity with the special symbols, it is suggested that the value of each parameter be quoted in `%str( )`. The following parameters are defined in the `%FiniteHMM` macro:

DATA

specifies the two-level input table name.

DECODE

specifies options for the DECODE statement.

ESTIMATE

specifies options for the ESTIMATE statement.

EVALUATE

specifies options for the EVALUATE statement.

FILTER

specifies options for the FILTER statement.

FORECAST

specifies options for the FORECAST statement.

ID

specifies options for the ID statement.

INITIAL

specifies options for the INITIAL statement.

MODEL

specifies options for the MODEL statement.

ODSOUT

specifies options for the ODS OUTPUT statement.

OPTIMIZE

specifies options for the OPTIMIZE statement.

PRIOR

specifies options for the PRIOR statement.

PROC

specifies options in the PROC HMM statement, excluding the DATA= option.

SCORE

specifies options for the SCORE statement.

SMOOTH

specifies options for the SMOOTH statement.

Among the macro parameters, only DATA, ID, and MODEL are required. All others are optional. For example, say that you run finite HMM by using the following PROC HMM specification:

```
ods output CPM=my_cpm_1
  ParameterEstimates=my_est_1;
proc hmm data=mylib.finitel;
  id time=t section=section;
  model y / type=finite nstate=3 nCategory=6 method=MAP;
  estimate out=mylib.est_1 outall=mylib.estall_1;
  forecast out=mylib.fcst_1 lead=4;
run;
```

The corresponding %FiniteHMM macro syntax would be as follows:

```
%FiniteHMM(
  data = %str(mylib.finitel),
  id = %str(time=t section=section),
  model = %str(y / type=finite nstate=3 nCategory=6 method=MAP),
  estimate = %str(out=mylib.est_1 outall=mylib.estall_1),
  forecast = %str(out=mylib.fcst_1 lead=4),
  odsout = %str(CPM=my_cpm_1
    ParameterEstimates=my_est_1));
```

Note that in the preceding example, because DATA=MYLIB.FINITE1 is the only option in the PROC HMM statement, the PROC parameter is not needed in the %FiniteHMM macro syntax. The NCATEGORY=6 option is not required in this macro because this value is determined by input data; hence, even if you specify it as shown here, it will be overwritten according to the data.

Now say that you have a more complicated specification of PROC HMM, such as the following:

```

ods output CPM=my_cpm_0
      ParameterEstimates=my_est_0;
proc hmm data=mylib.finitel labelSwitch=(sort=asc(tpm));
  id time=t section=section;
  model y1 / type=finitel nstate=3 nCategory=6 method=MAP;
  score outmodel=mylib.myScore_0;
  estimate out=mylib.est_0 outall=mylib.estall_0;
  forecast out=mylib.fcst_0 lead=4;
  decode out=mylib.decode_0;
  evaluate out=mylib.eval_0;
  filter out=mylib.flt_0;
  smooth out=mylib.smth_0;
  optimize printlevel=3 printiterfreq=2 maxiter=10;
  prior CPM ~ dir({1 1 1 2 2 2,
                  2 2 2 3 3 3,
                  2 2 2 4 4 4});
  initial TPM = {0.97 0.02 0.01,
                0.015 0.98 0.005,
                0.01 0.03 0.96},
  CPM = {0.09 0.32 0.57 0.01 0.01 0.00,
        0.01 0.08 0.28 0.61 0.01 0.01,
        0.01 0.02 0.01 0.08 0.29 0.59};

run;

```

The corresponding %FiniteHMM macro syntax would look like this:

```

%FiniteHMM(
  data = %str(mylib.finitel),
  proc = %str(labelSwitch=(sort=asc(tpm))),
  id = %str(time=t section=section),
  model = %str(y1 / type=finitel nstate=3 nCategory=6 method=MAP),
  score = %str(outmodel=mylib.myScore_0),
  estimate = %str(out=mylib.est_0 outall=mylib.estall_0),
  forecast = %str(out=mylib.fcst_0 lead=4),
  decode = %str(out=mylib.decode_0),
  evaluate = %str(out=mylib.eval_0),
  filter = %str(out=mylib.flt_0),
  smooth = %str(out=mylib.smth_0),
  optimize = %str(printlevel=3 printiterfreq=2 maxiter=10),
  prior = %str(CPM ~ dir({1 1 1 2 2 2,
                        2 2 2 3 3 3,
                        2 2 2 4 4 4})),
  initial = %str(TPM = {0.97 0.02 0.01,
                      0.01 0.98 0.01,
                      0.01 0.03 0.96},
  CPM = {0.09 0.32 0.57 0.01 0.01 0.00,
        0.01 0.08 0.28 0.61 0.01 0.01,
        0.01 0.02 0.01 0.08 0.29 0.59}),
  odsout = %str(CPM=my_cpm_0
  ParameterEstimates=my_est_0));

```

Note that in the preceding example, PROC HMM has the LABELSWITCH=(SORT=ASC(TPM)) option in addition to the DATA= option, so a PROC parameter is included in the macro call to contain this option.

The sections that follow present several examples to demonstrate how to use the %FiniteHMM macro. First, a data set needs to be simulated for these macro examples. The following PROC IML and DATA step code simulates a data set on the basis of a finite HMM model with the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$:

$$\pi = \begin{pmatrix} 0.45 \\ 0.25 \\ 0.30 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 0.98 & 0.015 & 0.005 \\ 0.02 & 0.97 & 0.01 \\ 0.01 & 0.01 & 0.98 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{pmatrix}$$

where π is the initial state probability vector, or ISPV; $\mathbf{A}$ is the transition probability matrix, or TPM; and $\mathbf{B}$ is the category probability matrix, or CPM. The simulated data set, finite1, has 1,500 observations in three sections. Each section has 500 observations. The data set has three variables: y, t, and section.

```
proc iml;
*** Defining model parameters: A and PI *****;
a={0.98 0.015 0.005, 0.02 0.97 0.01, 0.01 0.01 0.98};
pi={0.45 0.25 0.30};
*** Simulating a Markov process *****;
seed=54321;
call streaminit(seed);
free st state;
do i=1 to 500;
  tmp_st={0 0 0};
  if i=1 then do;
    check_p=pi;
  end;
  else do;
    lag_st=st[i-1,];
    check_p=lag_st*a;
  end;
  u=uniform(seed);
  if u<=check_p[1] then tmp_st[1]=1;
  else do;
    tmp=check_p[1] + check_p[2];
    if u<=tmp then tmp_st[2]=1;
    else tmp_st[3]=1;
  end;
  tmp_state=loc(tmp_st=1);
  st=st//tmp_st;
  state=state//tmp_state;
  seed=seed+1;
end;
create sim_stat from st[colname={"state1" "state2" "state3"}];
append from st;
create sim_state from state[colname={"state1"}];
append from state;
quit;
*** Simulating finite HMM data *****;
%let simseed=4321;
data temnm(keep = section t y);
  call streaminit(&simseed.);
  retain section 1;
  set sim_state;
```

```

t=_n_;
if(statel=1) then do;
  u = uniform(&simseed.);
  if (u <= 1/2) then y=1;
  else if( 1/2 <u <= 2/2) then y=2;
  else if( 2/2 <u <= 1) then y=3;
  else if( 1 <u <= 1) then y=4;
  else if( 1 <u <= 1) then y=5;
  else y = 6;
end;
else if statel=2 then do;
  u = uniform(&simseed.);
  if (u <= 0) then y=1;
  else if( 0 <u <= 0) then y=2;
  else if( 0 <u <= 1/2) then y=3;
  else if( 1/2 <u <= 2/2) then y=4;
  else if( 2/2 <u <= 1) then y=5;
  else y = 6;
end;
else do;
  u = uniform(&simseed.);
  if (u <= 0) then y=1;
  else if( 0 <u <= 0) then y=2;
  else if( 0 <u <= 0) then y=3;
  else if( 0 <u <= 0) then y=4;
  else if( 0 <u <= 1/2) then y=5;
  else y = 6;
end;
output;
run;
data finitel;
  set temnm;
run;
%let simseed1=1234;
data temnm(keep = section t y);
  call streaminit(&simseed1.);
  retain section 2;
  set sim_state;
  t=_n_;
  if(statel=1) then do;
    u = uniform(&simseed1.);
    if (u <= 1/2) then y=1;
    else if( 1/2 <u <= 2/2) then y=2;
    else if( 2/2 <u <= 1) then y=3;
    else if( 1 <u <= 1) then y=4;
    else if( 1 <u <= 1) then y=5;
    else y = 6;
  end;
  else if statel=2 then do;
    u = uniform(&simseed1.);
    if (u <= 0) then y=1;
    else if( 0 <u <= 0) then y=2;
    else if( 0 <u <= 1/2) then y=3;
    else if( 1/2 <u <= 2/2) then y=4;

```

```

        else if( 2/2 <u <= 1) then y=5;
        else y = 6;
    end;
else do;
    u = uniform(&simseed1.);
    if (u <= 0) then y=1;
    else if( 0 <u <= 0) then y=2;
    else if( 0 <u <= 0) then y=3;
    else if( 0 <u <= 0) then y=4;
    else if( 0 <u <= 1/2) then y=5;
    else y = 6;
end;
output;
run;
data finitel;
    set finitel temnm;
run;
%let simseed2=5432;
data temnm(keep = section t y);
    call streaminit(&simseed2.);
    retain section 3;
    set sim_state;
    t=_n_;
    if(statel=1) then do;
        u = uniform(&simseed2.);
        if (u <= 1/2) then y=1;
        else if( 1/2 <u <= 2/2) then y=2;
        else if( 2/2 <u <= 1) then y=3;
        else if( 1 <u <= 1) then y=4;
        else if( 1 <u <= 1) then y=5;
        else y = 6;
    end;
    else if statel=2 then do;
        u = uniform(&simseed2.);
        if (u <= 0) then y=1;
        else if( 0 <u <= 0) then y=2;
        else if( 0 <u <= 1/2) then y=3;
        else if( 1/2 <u <= 2/2) then y=4;
        else if( 2/2 <u <= 1) then y=5;
        else y = 6;
    end;
    else do;
        u = uniform(&simseed2.);
        if (u <= 0) then y=1;
        else if( 0 <u <= 0) then y=2;
        else if( 0 <u <= 0) then y=3;
        else if( 0 <u <= 0) then y=4;
        else if( 0 <u <= 1/2) then y=5;
        else y = 6;
    end;
    output;
run;
data finitel;
    set finitel temnm;

```

```
run;
```

The variable `y` in the `finite1` data set has values of the consecutive natural numbers 1 to 6. In this case, you can use either PROC HMM or the `%FiniteHMM` macro on `y` for a finite HMM. The results should be the same.

The following SAS code saves the `finite1` data set to the `mylib` library:

```
data mylib.finite1;
    set finite1;
run;
```

Macro Example 1: Finite HMM with Consecutive Natural Number Responses

In this example, both PROC HMM and the `%FiniteHMM` macro should give you the same numeric results. Both the ODS table and the output parameter estimates table that this macro produces have an additional column named `RespVarLevel` to indicate the levels of the response variable. In practice, if a response variable has values of consecutive natural numbers, you can use PROC HMM directly.

The first part of the following SAS code uses PROC HMM on `y`. The ODS output tables for CPM and the parameter estimates are stored in `my_cpm_0` and `my_est_0`, respectively. Then `%FiniteHMM` is used on `y` for the same model specification. The ODS output tables for CPM and the parameter estimates are stored in `my_cpm_1` and `my_est_1`, respectively.

```
*** Using PROC HMM on y *****;
ods listing close;
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite1 ;
    id time=t section=section;
    model y / type=finite nstate=3 nCategory=6 method=MAP;
    estimate out=mylib.est_0 outall=mylib.estall_0;
    forecast out=mylib.fcst_0 lead=4;
run;
*** Using %FiniteHMM on y *****;
%FiniteHMM(
    data=%str(mylib.finite1),
    id = %str(time=t section=section),
    model = %str(y / type=finite nstate=3 nCategory=6 method=MAP),
    estimate = %str(out=mylib.est_1 outall=mylib.estall_1),
    forecast = %str(out=mylib.fcst_1 lead=4),
    odsout = %str(CPM=my_cpm_1
                  ParameterEstimates=my_est_1)
);
```

The following SAS code prints the ODS CPM output table from PROC HMM. The table is shown in Figure 20.68.

```
proc print data=my_cpm_0 label noobs;
    format category: 8.5;
run;
```


Figure 20.68 ODS CPM Output Table from PROC HMM

Group	State	1	2	3	4	5	6
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.51608	0.48392
ParameterMatrices	2	0.00000	0.00000	0.48113	0.51887	0.00000	0.00000
ParameterMatrices	3	0.52209	0.47791	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS CPM output table from the %FiniteHMM macro. The table is shown in Figure 20.69.

```
proc print data=my_cpm_1 label noobs;
  format category: 8.5;
run;
```

Figure 20.69 ODS CPM Output Table from the %FiniteHMM Macro

Group	State	1	2	3	4	5	6
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.51608	0.48392
ParameterMatrices	2	0.00000	0.00000	0.48113	0.51887	0.00000	0.00000
ParameterMatrices	3	0.52209	0.47791	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS parameter estimates table from PROC HMM. The table is shown in Figure 20.70.

```
proc print data=my_est_0 noobs;
  format Estimate StdErr 8.5;
run;
```

Figure 20.70 ODS Parameter Estimates Table from PROC HMM

Parameter	Estimate	StdErr	tValue	Probt
TPM1_1	0.98912	0.00380	260.10	<.0001
TPM1_2	0.00000	0.00000	0.00	0.9986
TPM1_3	0.01088	0.00380	2.86	0.0043
TPM2_1	0.00941	0.00543	1.73	0.0837
TPM2_2	0.97953	0.00771	127.07	<.0001
TPM2_3	0.01107	0.00595	1.86	0.0629
TPM3_1	0.01012	0.00425	2.38	0.0173
TPM3_2	0.01014	0.00425	2.39	0.0172
TPM3_3	0.97974	0.00614	159.65	<.0001
CPM1_1	0.00000	0.00000	0.00	0.9974
CPM1_2	0.00000	0.00001	0.00	0.9973
CPM1_3	0.00000	0.00000	0.00	0.9984
CPM1_4	0.00000	0.00000	0.00	0.9986
CPM1_5	0.51608	0.01911	27.01	<.0001
CPM1_6	0.48392	0.01911	25.33	<.0001
CPM2_1	0.00000	0.00001	0.00	0.9972
CPM2_2	0.00000	0.00001	0.00	0.9987
CPM2_3	0.48113	0.02802	17.17	<.0001
CPM2_4	0.51887	0.02802	18.52	<.0001
CPM2_5	0.00000	0.00009	0.00	0.9999
CPM2_6	0.00000	0.00009	0.00	0.9999
CPM3_1	0.52209	0.02238	23.32	<.0001
CPM3_2	0.47791	0.02238	21.35	<.0001
CPM3_3	0.00000	0.00002	0.00	0.9999
CPM3_4	0.00000	0.00001	0.00	0.9997
CPM3_5	0.00000	0.00000	0.00	1.0000
CPM3_6	0.00000	0.00001	0.00	0.9997

The following SAS code prints the ODS parameter estimates table from the %FiniteHMM macro. The table is shown in [Figure 20.71](#).

```
proc print data=my_est_1 noobs;
    format Estimate StdErr 8.5;
run;
```

Figure 20.71 ODS Parameter Estimates Table from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	StdErr	tValue	Probt
	TPM1_1	0.98912	0.00380	260.10	<.0001
	TPM1_2	0.00000	0.00000	0.00	0.9986
	TPM1_3	0.01088	0.00380	2.86	0.0043
	TPM2_1	0.00941	0.00543	1.73	0.0837
	TPM2_2	0.97953	0.00771	127.07	<.0001
	TPM2_3	0.01107	0.00595	1.86	0.0629
	TPM3_1	0.01012	0.00425	2.38	0.0173
	TPM3_2	0.01014	0.00425	2.39	0.0172
	TPM3_3	0.97974	0.00614	159.65	<.0001
1	CPM1_1	0.00000	0.00000	0.00	0.9974
2	CPM1_2	0.00000	0.00001	0.00	0.9973
3	CPM1_3	0.00000	0.00000	0.00	0.9984
4	CPM1_4	0.00000	0.00000	0.00	0.9986
5	CPM1_5	0.51608	0.01911	27.01	<.0001
6	CPM1_6	0.48392	0.01911	25.33	<.0001
1	CPM2_1	0.00000	0.00001	0.00	0.9972
2	CPM2_2	0.00000	0.00001	0.00	0.9987
3	CPM2_3	0.48113	0.02802	17.17	<.0001
4	CPM2_4	0.51887	0.02802	18.52	<.0001
5	CPM2_5	0.00000	0.00009	0.00	0.9999
6	CPM2_6	0.00000	0.00009	0.00	0.9999
1	CPM3_1	0.52209	0.02238	23.32	<.0001
2	CPM3_2	0.47791	0.02238	21.35	<.0001
3	CPM3_3	0.00000	0.00002	0.00	0.9999
4	CPM3_4	0.00000	0.00001	0.00	0.9997
5	CPM3_5	0.00000	0.00000	0.00	1.0000
6	CPM3_6	0.00000	0.00001	0.00	0.9997

Figure 20.71 shows that the parameter estimates table that the %FiniteHMM macro produces has an additional column named RespVarLevel to indicate the levels of the original response variable.

Macro Example 2: Finite HMM with Categorical Responses

In this example, the variable y_1 has the categorical values ‘aa email’, ‘bb mail’, ‘cc phone’, ‘dd flyer’, ‘ee seminar’, and ‘ff forum’. In this case, directly applying PROC HMM to y_1 generates error messages, and PROC HMM quits. Using the %FiniteHMM macro on y_1 gives you the expected results.

The following SAS code creates the finite2 data set:

```
data finite2;
  set finitel;
  if y=1 then y1='aa email  ';
  if y=2 then y1='bb mail   ';
  if y=3 then y1='cc phone  ';
  if y=4 then y1='dd flyer  ';
  if y=5 then y1='ee seminar';
  if y=6 then y1='ff forum  ';
```

```
run;
data mylib.finite2;
  set finite2;
run;
```

The following SAS code uses PROC HMM:

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite2;
  id time=t section=section;
  model y1 / type=finite nstate=3 nCategory=6 method=MAP;
  estimate out=mylib.est_0 outall=mylib.estall_0;
  forecast out=mylib.fcst_0 lead=4;
run;
ERROR: The data type of dependent variables must be numeric.
```

The following SAS code applies the %FiniteHMM macro:

```
%FiniteHMM(
  data = %str(mylib.finite2),
  proc = %str(labelSwitch=(sort=asc(cpm))),
  id = %str(time=t section=section),
  model = %str(y1 / type=finite nstate=3 nCategory=6 method=MAP),
  estimate = %str(out=mylib.est_1 outall=mylib.estall_1),
  forecast = %str(out=mylib.fcst_1 lead=4),
  odsout = %str(CPM=my_cpm_1
                ParameterEstimates=my_est_1));
```

The following SAS code prints the ODS CPM output table from the %FiniteHMM macro. The table is shown in [Figure 20.72](#).

```
proc print data=my_cpm_1 label noobs;
  format category: 8.5;
run;
```

Figure 20.72 ODS CPM Output Table from the %FiniteHMM Macro

Group	State	ee					
		aa email	bb mail	cc phone	dd flyer	seminar	ff forum
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.51608	0.48392
ParameterMatrices	2	0.00000	0.00000	0.48113	0.51887	0.00000	0.00000
ParameterMatrices	3	0.52209	0.47791	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS parameter estimates table from the %FiniteHMM macro. The table is shown in [Figure 20.73](#).

```
proc print data=my_est_1 noobs;
  format Estimate StdErr 8.5;
run;
```

Figure 20.73 ODS Parameter Estimates Table from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	StdErr	tValue	Probt
	TPM1_1	0.98912	0.00380	260.10	<.0001
	TPM1_2	0.00000	0.00000	0.00	0.9986
	TPM1_3	0.01088	0.00380	2.86	0.0043
	TPM2_1	0.00941	0.00543	1.73	0.0837
	TPM2_2	0.97953	0.00771	127.07	<.0001
	TPM2_3	0.01107	0.00595	1.86	0.0629
	TPM3_1	0.01012	0.00425	2.38	0.0173
	TPM3_2	0.01014	0.00425	2.39	0.0172
	TPM3_3	0.97974	0.00614	159.65	<.0001
aa email	CPM1_1	0.00000	0.00000	0.00	0.9974
bb mail	CPM1_2	0.00000	0.00001	0.00	0.9973
cc phone	CPM1_3	0.00000	0.00000	0.00	0.9984
dd flyer	CPM1_4	0.00000	0.00000	0.00	0.9986
ee seminar	CPM1_5	0.51608	0.01911	27.01	<.0001
ff forum	CPM1_6	0.48392	0.01911	25.33	<.0001
aa email	CPM2_1	0.00000	0.00001	0.00	0.9972
bb mail	CPM2_2	0.00000	0.00001	0.00	0.9987
cc phone	CPM2_3	0.48113	0.02802	17.17	<.0001
dd flyer	CPM2_4	0.51887	0.02802	18.52	<.0001
ee seminar	CPM2_5	0.00000	0.00009	0.00	0.9999
ff forum	CPM2_6	0.00000	0.00009	0.00	0.9999
aa email	CPM3_1	0.52209	0.02238	23.32	<.0001
bb mail	CPM3_2	0.47791	0.02238	21.35	<.0001
cc phone	CPM3_3	0.00000	0.00002	0.00	0.9999
dd flyer	CPM3_4	0.00000	0.00001	0.00	0.9997
ee seminar	CPM3_5	0.00000	0.00000	0.00	1.0000
ff forum	CPM3_6	0.00000	0.00001	0.00	0.9997

The following SAS code prints the output forecast table from the %FiniteHMM macro. The table is shown in Figure 20.74.

```
proc print data=mylib.fcst_1 label noobs;
    format State: Category: 8.5;
run;
```

Figure 20.74 Output Forecast Table from the %FiniteHMM Macro

section	t	Step	State1	State2	State3	aa email	bb mail	cc phone	dd flyer	ee	
										seminar	ff forum
1	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
1	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
1	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
1	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350
2	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
2	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
2	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
2	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350
3	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
3	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
3	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
3	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350

The RespVarLevel column in Figure 20.73 provides information about the levels of response variables for the CPM parameters. The labels in Figure 20.74 clearly show the categorical levels of the response variable y_1 .

Macro Example 3: Finite HMM for Responses with Nonconsecutive Natural Number Values

In this example, although y_1 has values of 2 and 6, PROC HMM still assumes that it has consecutive values of 1, 2, 3, 4, 5, and 6, and the output tables for CPM and the forecast will contain extra columns for the values 1, 3, 4, and 5. The %FiniteHMM macro will consider 2 and 6 as two categorical values, so it will give you the desired results.

The following SAS code creates the finite3 data set:

```
data finite3;
    set finitel;
    if y<=3 then y1=2; else y1=6;
run;
data mylib.finite3;
    set finite3;
run;
```

The following SAS code uses PROC HMM on y_1 :

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite3 labelSwitch=(sort=asc(cpm));
    id time=t section=section;
    model y1 / type=finite nstate=3 nCategory=6 method=MAP;
    estimate out=mylib.est_0 outall=mylib.estall_0;
    forecast out=mylib.fcst_0 lead=4;
run;
```

The following SAS code prints the ODS CPM output table, parameter estimates table, and output forecast table, respectively, from PROC HMM. The results are shown in Figure 20.75, Figure 20.76, and Figure 20.77.

```

proc print data=my_cpm_0 label noobs;
  format Category: 8.5;
run;

proc print data=my_est_0 label noobs;
  format Estimate StdErr 8.5;
run;

proc print data=mylib.fcst_0 label noobs;
  format State: Category: 8.5;
run;

```

Figure 20.75 ODS CPM Output Table from PROC HMM

Group	State	1	2	3	4	5	6
ParameterMatrices	1	0.00000	0.48869	0.00000	0.00000	0.00000	0.51131
ParameterMatrices	2	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
ParameterMatrices	3	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000

Figure 20.76 ODS Parameter Estimates Table from PROC HMM

Parameter	Estimate	Standard Error	t value	Pr > t
TPM1_1	0.97835	0.00819	119.47	<.0001
TPM1_2	0.01251	0.00727	1.72	0.0857
TPM1_3	0.00915	0.00634	1.44	0.1496
TPM2_1	0.01144	0.00482	2.37	0.0177
TPM2_2	0.97817	0.00666	146.93	<.0001
TPM2_3	0.01038	0.00465	2.23	0.0256
TPM3_1	0.00000	0.00001	0.00	0.9968
TPM3_2	0.01120	0.00393	2.85	0.0045
TPM3_3	0.98880	0.00393	251.34	<.0001
CPM1_1	0.00000	0.00000	0.00	0.9998
CPM1_2	0.48869	0.02941	16.62	<.0001
CPM1_3	0.00000	0.00000	0.00	0.9998
CPM1_4	0.00000	0.00000	0.00	0.9998
CPM1_5	0.00000	0.00000	0.00	0.9998
CPM1_6	0.51131	0.02941	17.39	<.0001
CPM2_1	0.00000	0.00000	0.00	0.9998
CPM2_2	1.00000	0.00000	1735239	<.0001
CPM2_3	0.00000	0.00000	0.00	1.0000
CPM2_4	0.00000	0.00000	0.00	1.0000
CPM2_5	0.00000	0.00000	0.00	1.0000
CPM2_6	0.00000	0.00000	0.00	1.0000
CPM3_1	0.00000	0.00000	0.00	0.9997
CPM3_2	0.00000	0.00000	0.00	0.9995
CPM3_3	0.00000	0.00000	0.00	0.9997
CPM3_4	0.00000	0.00000	0.00	0.9996
CPM3_5	0.00000	0.00000	0.00	0.9996
CPM3_6	1.00000	0.00000	636590	<.0001

Figure 20.77 Output Forecast Table from PROC HMM

section	t	Step	State1	State2	State3	Category1	Category2	Category3	Category4	Category5	Category6
1	500	1	0.06414	0.01129	0.92457	0.00000	0.04264	0.00000	0.00000	0.00000	0.95736
1	500	2	0.06288	0.02221	0.91491	0.00000	0.05294	0.00000	0.00000	0.00000	0.94706
1	500	3	0.06178	0.03276	0.90547	0.00000	0.06295	0.00000	0.00000	0.00000	0.93705
1	500	4	0.06081	0.04296	0.89623	0.00000	0.07268	0.00000	0.00000	0.00000	0.92732
2	500	1	0.06414	0.01129	0.92457	0.00000	0.04264	0.00000	0.00000	0.00000	0.95736
2	500	2	0.06288	0.02221	0.91491	0.00000	0.05294	0.00000	0.00000	0.00000	0.94706
2	500	3	0.06178	0.03276	0.90547	0.00000	0.06295	0.00000	0.00000	0.00000	0.93705
2	500	4	0.06081	0.04296	0.89623	0.00000	0.07268	0.00000	0.00000	0.00000	0.92732
3	500	1	0.06414	0.01129	0.92457	0.00000	0.04264	0.00000	0.00000	0.00000	0.95736
3	500	2	0.06288	0.02221	0.91491	0.00000	0.05294	0.00000	0.00000	0.00000	0.94706
3	500	3	0.06178	0.03276	0.90547	0.00000	0.06295	0.00000	0.00000	0.00000	0.93705
3	500	4	0.06081	0.04296	0.89623	0.00000	0.07268	0.00000	0.00000	0.00000	0.92732

If you use the %FiniteHMM macro on y_1 , as in the following code, the results become concise and correct. Be aware that because the number of levels of the response variable differs in these three instances, the output state orders could be different even if you use the same LABELSWITCH= option value in both PROC HMM and the %FiniteHMM macro.

The following SAS program uses %FiniteHMM on y_1 :

```
%FiniteHMM(
  data = %str(mylib.finite3),
  proc = %str(labelSwitch=(sort=asc(cpm))),
  id = %str(time=t section=section),
  model = %str(y1 / type=finit nstate=3 nCategory=6 method=MAP),
  estimate = %str(out=mylib.est_1 outall=mylib.estall_1),
  forecast = %str(out=mylib.fcst_1 lead=4),
  odsout = %str(CPM=my_cpm_1
    ParameterEstimates=my_est_1));
```

The following three SAS programs print the ODS CPM output table, parameter estimates table, and output forecast table, respectively, from the %FiniteHMM macro. The results are shown in [Figure 20.78](#), [Figure 20.79](#), and [Figure 20.80](#).

```
proc print data=my_cpm_1 label noobs;
  format Category: 8.5;
run;

proc print data=my_est_1 label noobs;
  format Estimate StdErr 8.5;
run;

proc print data=mylib.fcst_1 label noobs;
  format State: Category: 8.5;
run;
```


Figure 20.78 ODS CPM Output Table from the %FiniteHMM Macro

Group	State	2	6
ParameterMatrices	1	0.00000	1.00000
ParameterMatrices	2	0.48869	0.51131
ParameterMatrices	3	1.00000	0.00000

Figure 20.79 ODS Parameter Estimates Table from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	Standard Error	t value	Pr > t
	TPM1_1	0.98880	0.00393	251.34	<.0001
	TPM1_2	0.00000	0.00000	0.00	0.9989
	TPM1_3	0.01120	0.00393	2.85	0.0045
	TPM2_1	0.00915	0.00634	1.44	0.1496
	TPM2_2	0.97835	0.00819	119.47	<.0001
	TPM2_3	0.01251	0.00727	1.72	0.0857
	TPM3_1	0.01038	0.00465	2.23	0.0256
	TPM3_2	0.01144	0.00482	2.37	0.0177
	TPM3_3	0.97817	0.00666	146.93	<.0001
2	CPM1_1	0.00000	0.00000	0.00	0.9990
6	CPM1_2	1.00000	0.00000	548383	<.0001
2	CPM2_1	0.48869	0.02941	16.62	<.0001
6	CPM2_2	0.51131	0.02941	17.39	<.0001
2	CPM3_1	1.00000	0.00001	113383	<.0001
6	CPM3_2	0.00000	0.00001	0.00	0.9967

Figure 20.80 Output Forecast Table from the %FiniteHMM Macro

section	t	Step	State1	State2	State3	2	6
1	500	1	0.92457	0.06414	0.01129	0.04264	0.95736
1	500	2	0.91491	0.06288	0.02221	0.05294	0.94706
1	500	3	0.90547	0.06178	0.03276	0.06295	0.93705
1	500	4	0.89623	0.06081	0.04296	0.07268	0.92732
2	500	1	0.92457	0.06414	0.01129	0.04264	0.95736
2	500	2	0.91491	0.06288	0.02221	0.05294	0.94706
2	500	3	0.90547	0.06178	0.03276	0.06295	0.93705
2	500	4	0.89623	0.06081	0.04296	0.07268	0.92732
3	500	1	0.92457	0.06414	0.01129	0.04264	0.95736
3	500	2	0.91491	0.06288	0.02221	0.05294	0.94706
3	500	3	0.90547	0.06178	0.03276	0.06295	0.93705
3	500	4	0.89623	0.06081	0.04296	0.07268	0.92732

Figure 20.75 shows that the CPM output table from PROC HMM has six columns, but only two columns contain useful information. Similarly, the parameter estimates table from PROC HMM also contains extra parameters. On the other hand, results from the %FiniteHMM macro contain information for two levels of the values 2 and 6 only. On a large scale, if y_1 has values of 1 and 100, then PROC HMM assumes that it has the consecutive values 1 through 100, and the CPM output table for y_1 has 100 columns, of which only two

columns contain useful information. The %FiniteHMM macro gives you results on two categories, 1 and 100, and is much more efficient. The RespVarLevel column in Figure 20.79 provides information about the levels of response variables for the CPM parameters. The labels in Figure 20.80 clearly show the categorical levels of the response variable y_1 .

Macro Example 4: Automatic Handling of the NCATEGORY= Option in the %FiniteHMM Macro

In this example, y_1 has integer values of 1, 3, 4, and 6. In PROC HMM, you get an error message if the value of the NCATEGORY= option is less than the maximum value of y_1 , which is 6. On the other hand, if the NCATEGORY= option value is greater than the maximum value of y_1 , PROC HMM assumes that y_1 has consecutive values from 1 to the NCATEGORY= option value. Hence the results will contain extra columns or parameters similar to those in Macro Example 3.

If you apply the %FiniteHMM macro, the NCATEGORY= option value is automatically updated to the number of levels of the response variable. There will be no error message if the NCATEGORY= option value is less than the maximum value of the response variable, and the results are more desirable.

The following SAS code creates the finite4 data set:

```
data finite4;
    set finitel;
    y1 = y;
    if y=2 then y1=1;
    if y=5 then y1=4;
run;
data mylib.finite4;
    set finite4;
run;
```

The following SAS code specifies NCATEGORY=4. PROC HMM generates an error message.

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite4 labelSwitch=(sort=asc(cpm));
    id time=t section=section;
    model y1 / type=finite nstate=3 nCategory=4 method=MAP;
    estimate out=mylib.est_0 outall=mylib.estall_0;
    forecast out=mylib.fcst_0 lead=4;
run;
ERROR: The dependent variable for the finite HMM must be an integer between 1
       and the number of the categories which is specified in the NCATEGORY= option
       in the MODEL statement. To correct this, you can either preprocess your data
       or specify a larger value in the NCATEGORY= option.
```

The following SAS code specifies NCATEGORY=8. PROC HMM assumes that y_1 has the consecutive integer values 1 through 8.

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite4 labelSwitch=(sort=asc(cpm));
    id time=t section=section;
    model y1 / type=finite nstate=3 nCategory=8 method=MAP;
    estimate out=mylib.est_0 outall=mylib.estall_0;
    forecast out=mylib.fcst_0 lead=4;
```

```
run;
```

The following SAS code prints the ODS CPM output table from PROC HMM when NCATEGORY=8. The table is shown in [Figure 20.81](#).

```
proc print data=my_cpm_0 label noobs;
format Category: 8.5;
run;
```

Figure 20.81 ODS CPM Output Table from PROC HMM When NCATEGORY=8

Group	State	1	2	3	4	5	6	7	8
ParameterMatrices	1	0.00000	0.00000	0.00000	0.51379	0.00000	0.48621	0.00000	0.00000
ParameterMatrices	2	0.00000	0.00000	0.47630	0.52370	0.00000	0.00000	0.00000	0.00000
ParameterMatrices	3	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

If you use the %FiniteHMM macro as in the following program, the NCATEGORY= option value is always updated to the actual number of levels of the response variable and a message is displayed. The ODS CPM output tables when NCATEGORY=4 and 8 are shown in [Figure 20.82](#) and [Figure 20.83](#), respectively.

```
%FiniteHMM(
  data = %str(mylib.finite4),
  proc = %str(labelSwitch=(sort=asc(cpm))),
  id = %str(time=t section=section),
  model = %str(y1 / type=finite nstate=3 nCategory=4 method=MAP),
  estimate = %str(out=mylib.est_1 outall=mylib.estall_1),
  forecast = %str(out=mylib.fcst_1 lead=4),
  odsout = %str(CPM=my_cpm_1
    ParameterEstimates=my_est_1));
```

NOTE: The NCATEGORY= option value is always updated to the number of levels of the response variable.

```
%FiniteHMM(
  data = %str(mylib.finite4),
  proc = %str(labelSwitch=(sort=asc(cpm))),
  id = %str(time=t section=section),
  model = %str(y1 / type=finite nstate=3 nCategory=8 method=MAP),
  estimate = %str(out=mylib.est_2 outall=mylib.estall_2),
  forecast = %str(out=mylib.fcst_2 lead=4),
  odsout = %str(CPM=my_cpm_2
    ParameterEstimates=my_est_2));
```

NOTE: The NCATEGORY= option value is always updated to the number of levels of the response variable.

The following SAS code prints the ODS CPM output table from the %FiniteHMM macro when NCATEGORY=4. The table is shown in [Figure 20.82](#).

```
proc print data=my_cpm_1 label noobs;
format Category: 8.5;
run;
```

Figure 20.82 ODS CPM Table from the %FiniteHMM Macro When NCATEGORY=4

Group	State	1	3	4	6
ParameterMatrices	1	0.00000	0.47630	0.52370	0.00000
ParameterMatrices	2	0.00000	0.00000	0.51379	0.48621
ParameterMatrices	3	1.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS CPM output table from the %FiniteHMM macro when NCATEGORY=8. The table is shown in Figure 20.83.

```
proc print data=my_cpm_2 label noobs;
format Category: 8.5;
run;
```

Figure 20.83 ODS CPM Table from the %FiniteHMM Macro When NCATEGORY=8

Group	State	1	3	4	6
ParameterMatrices	1	0.00000	0.47630	0.52370	0.00000
ParameterMatrices	2	0.00000	0.00000	0.51379	0.48621
ParameterMatrices	3	1.00000	0.00000	0.00000	0.00000

Macro Example 5: Finite HMM with Decimal-Valued or Negative-Valued Responses

In this example, the response variable y_1 has decimal values, and the response variable y_2 has values of negative integers. PROC HMM generates an error message for both y_1 and y_2 . But using the %FiniteHMM macro gives you correct and complete results.

The following SAS code creates the finite5 data set:

```
data finite5;
  set finitel;
  y1 = y/100;
  y2 = -y;
run;
data mylib.finite5;
  set finite5;
run;
```

Using PROC HMM on y_1 and y_2 as follows generates error messages:

```
ods output CPM=my_cpm_1
           ParameterEstimates=my_est_1;
proc hmm data=mylib.finite5 labelSwitch=(sort=asc(cpm));
  id time=t section=section;
  model y1 / type=finite nstate=3 nCategory=6 method=MAP;
  estimate out=mylib.est_1 outall=mylib.estall_1;
  forecast out=mylib.fcst_1 lead=4;
run;
ERROR: For finite HMM, the series variable must be integer-valued.

ods output CPM=my_cpm_2
           ParameterEstimates=my_est_2;
```

```
proc hmm data=mylib.finite5 labelSwitch=(sort=asc(cpm));
  id time=t section=section;
  model y2 / type=finite nstate=3 nCategory=6 method=MAP;
  estimate out=mylib.est_2 outall=mylib.estall_2;
  forecast out=mylib.fcst_2 lead=4;
run;
ERROR: The dependent variable for the finite HMM must be an integer between 1
       and the number of the categories which is specified in the NCATEGORY=
       option in the MODEL statement. To correct this, you can either
       preprocess your data or specify a larger value in the NCATEGORY= option.
```

Using the %FiniteHMM macro on y_1 and y_2 as follows provides the expected results:

```
%FiniteHMM(
  data = %str(mylib.finite5),
  proc = %str(labelSwitch=(sort=asc(cpm))),
  id = %str(time=t section=section),
  model = %str(y1 / type=finite nstate=3 nCategory=6 method=MAP),
  estimate = %str(out=mylib.est_1 outall=mylib.estall_1),
  forecast = %str(out=mylib.fcst_1 lead=4),
  odsout = %str(CPM=my_cpm_1
                ParameterEstimates=my_est_1));

%FiniteHMM(
  data = %str(mylib.finite5),
  proc = %str(labelSwitch=(sort=asc(cpm))),
  id = %str(time=t section=section),
  model = %str(y2 / type=finite nstate=3 nCategory=6 method=MAP),
  estimate = %str(out=mylib.est_2 outall=mylib.estall_2),
  forecast = %str(out=mylib.fcst_2 lead=4),
  odsout = %str(CPM=my_cpm_2
                ParameterEstimates=my_est_2));
```

The following SAS code prints the ODS CPM output table of y_1 from the %FiniteHMM macro. The table is shown in [Figure 20.84](#).

```
proc print data=my_cpm_1 label noobs;
  format Category: 8.5;
run;
```

Figure 20.84 ODS CPM Output Table of y_1 from the %FiniteHMM Macro

Group	State	0.01	0.02	0.03	0.04	0.05	0.06
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.51608	0.48392
ParameterMatrices	2	0.00000	0.00000	0.48113	0.51887	0.00000	0.00000
ParameterMatrices	3	0.52209	0.47791	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS parameter estimates table of y_1 from the %FiniteHMM macro. The table is shown in [Figure 20.85](#).

```
proc print data=my_est_1 label noobs;
  format Estimate StdErr 8.5;
run;
```

Figure 20.85 ODS Parameter Estimates Table of y_1 from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	Standard Error	t value	Pr > t
	TPM1_1	0.98912	0.00380	260.10	<.0001
	TPM1_2	0.00000	0.00000	0.00	0.9986
	TPM1_3	0.01088	0.00380	2.86	0.0043
	TPM2_1	0.00941	0.00543	1.73	0.0837
	TPM2_2	0.97953	0.00771	127.07	<.0001
	TPM2_3	0.01107	0.00595	1.86	0.0629
	TPM3_1	0.01012	0.00425	2.38	0.0173
	TPM3_2	0.01014	0.00425	2.39	0.0172
	TPM3_3	0.97974	0.00614	159.65	<.0001
0.01	CPM1_1	0.00000	0.00000	0.00	0.9974
0.02	CPM1_2	0.00000	0.00001	0.00	0.9973
0.03	CPM1_3	0.00000	0.00000	0.00	0.9984
0.04	CPM1_4	0.00000	0.00000	0.00	0.9986
0.05	CPM1_5	0.51608	0.01911	27.01	<.0001
0.06	CPM1_6	0.48392	0.01911	25.33	<.0001
0.01	CPM2_1	0.00000	0.00001	0.00	0.9972
0.02	CPM2_2	0.00000	0.00001	0.00	0.9987
0.03	CPM2_3	0.48113	0.02802	17.17	<.0001
0.04	CPM2_4	0.51887	0.02802	18.52	<.0001
0.05	CPM2_5	0.00000	0.00009	0.00	0.9999
0.06	CPM2_6	0.00000	0.00009	0.00	0.9999
0.01	CPM3_1	0.52209	0.02238	23.32	<.0001
0.02	CPM3_2	0.47791	0.02238	21.35	<.0001
0.03	CPM3_3	0.00000	0.00002	0.00	0.9999
0.04	CPM3_4	0.00000	0.00001	0.00	0.9997
0.05	CPM3_5	0.00000	0.00000	0.00	1.0000
0.06	CPM3_6	0.00000	0.00001	0.00	0.9997

The following SAS code prints the output forecast table of y_1 from the %FiniteHMM macro. The table is shown in [Figure 20.86](#).

```
proc print data=mylib.fcst_1 label noobs;
  format State: Category: 8.5;
run;
```

Figure 20.86 Output Forecast Table of y_1 from the %FiniteHMM Macro

section	t	Step	State1	State2	State3	0.01	0.02	0.03	0.04	0.05	0.06
1	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
1	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
1	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
1	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350
2	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
2	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
2	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
2	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350
3	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
3	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
3	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
3	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350

The following SAS code prints the ODS CPM output table of y_2 from the %FiniteHMM macro. The table is shown in [Figure 20.87](#).

```
proc print data=my_cpm_2 label noobs;
  format Category: 8.5;
run;
```

Figure 20.87 ODS CPM Output Table of y_2 from the %FiniteHMM Macro

Group	State	-6	-5	-4	-3	-2	-1
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.47791	0.52209
ParameterMatrices	2	0.00000	0.00000	0.51887	0.48113	0.00000	0.00000
ParameterMatrices	3	0.48392	0.51608	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS parameter estimates table of y_2 from the %FiniteHMM macro. The table is shown in [Figure 20.88](#).

```
proc print data=my_est_2 label noobs;
  format Estimate StdErr 8.5;
run;
```

Figure 20.88 ODS Parameter Estimates Table of y_2 from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	Standard Error	t value	Pr > t
	TPM1_1	0.97974	0.00614	159.65	<.0001
	TPM1_2	0.01014	0.00425	2.39	0.0172
	TPM1_3	0.01012	0.00425	2.38	0.0173
	TPM2_1	0.01107	0.00595	1.86	0.0629
	TPM2_2	0.97953	0.00771	127.07	<.0001
	TPM2_3	0.00941	0.00543	1.73	0.0837
	TPM3_1	0.01088	0.00380	2.86	0.0043
	TPM3_2	0.00000	0.00000	0.00	0.9996
	TPM3_3	0.98912	0.00380	260.10	<.0001
-6	CPM1_1	0.00000	0.00000	0.00	0.9996
-5	CPM1_2	0.00000	0.00000	0.00	0.9996
-4	CPM1_3	0.00000	0.00001	0.00	0.9973
-3	CPM1_4	0.00000	0.00000	0.00	0.9985
-2	CPM1_5	0.47791	0.02238	21.35	<.0001
-1	CPM1_6	0.52209	0.02238	23.32	<.0001
-6	CPM2_1	0.00000	0.00000	0.00	0.9989
-5	CPM2_2	0.00000	0.00000	0.00	0.9991
-4	CPM2_3	0.51887	0.02802	18.52	<.0001
-3	CPM2_4	0.48113	0.02802	17.17	<.0001
-2	CPM2_5	0.00000	0.00000	0.00	1.0000
-1	CPM2_6	0.00000	0.00000	0.00	1.0000
-6	CPM3_1	0.48392	0.01911	25.33	<.0001
-5	CPM3_2	0.51608	0.01911	27.01	<.0001
-4	CPM3_3	0.00000	0.00000	0.00	0.9994
-3	CPM3_4	0.00000	0.00004	0.00	1.0000
-2	CPM3_5	0.00000	0.00000	0.00	1.0000
-1	CPM3_6	0.00000	0.00004	0.00	1.0000

The following SAS code prints the output forecast table of y_2 from the %FiniteHMM macro. The table is shown in Figure 20.89.

```
proc print data=mylib.fcst_2 label noobs;
  format State: Category: 8.5;
run;
```


Figure 20.89 Output Forecast Table of y2 from the %FiniteHMM Macro

section	t	Step	State1	State2	State3	-6	-5	-4	-3	-2	-1
1	500	1	0.01088	0.00000	0.98912	0.47865	0.51046	0.00000	0.00000	0.00520	0.00568
1	500	2	0.02143	0.00011	0.97846	0.47349	0.50497	0.00006	0.00005	0.01024	0.01119
1	500	3	0.03165	0.00033	0.96803	0.46845	0.49958	0.00017	0.00016	0.01512	0.01652
1	500	4	0.04155	0.00064	0.95781	0.46350	0.49431	0.00033	0.00031	0.01986	0.02169
2	500	1	0.01088	0.00000	0.98912	0.47865	0.51046	0.00000	0.00000	0.00520	0.00568
2	500	2	0.02143	0.00011	0.97846	0.47349	0.50497	0.00006	0.00005	0.01024	0.01119
2	500	3	0.03165	0.00033	0.96803	0.46845	0.49958	0.00017	0.00016	0.01512	0.01652
2	500	4	0.04155	0.00064	0.95781	0.46350	0.49431	0.00033	0.00031	0.01986	0.02169
3	500	1	0.01088	0.00000	0.98912	0.47865	0.51046	0.00000	0.00000	0.00520	0.00568
3	500	2	0.02143	0.00011	0.97846	0.47349	0.50497	0.00006	0.00005	0.01024	0.01119
3	500	3	0.03165	0.00033	0.96803	0.46845	0.49958	0.00017	0.00016	0.01512	0.01652
3	500	4	0.04155	0.00064	0.95781	0.46350	0.49431	0.00033	0.00031	0.01986	0.02169

Poisson Hidden Markov Model

The Poisson hidden Markov model (Poisson HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the Poisson distribution whose support is nonnegative integers. The observation y_t is said to have a Poisson distribution, with the parameter $\lambda_{\mathbf{S}_t} > 0$, if it has a probability mass function given by

$$p(y_t = k | \mathbf{S}_t) = \frac{e^{-\lambda_{\mathbf{S}_t}} \lambda_{\mathbf{S}_t}^k}{k!}$$

where the variable $\mathbf{S}_t$ is the (hidden) state, k is the number of occurrences, and $!$ is the factorial function. The positive real number $\lambda_{\mathbf{S}_t}$ is equal to the expected value of y_t and also to its variance in its hidden state.

Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the probability of observational outcomes $\mathbf{B} (\equiv \{b_{ij}, i = 1, \dots, K, j = 1, \dots, M\})$ together specify the Poisson HMM; that is, the parameter θ of the Poisson HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the Poisson HMM is a type of finite-state-space and homogeneous HMM, you can solve the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—by using the three algorithms that are introduced in the section “[Hidden Markov Model](#)” on page 1268. That is, you can solve the filtering, forecasting, and evaluating problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood. The dependent variable for the finite HMM must be a nonnegative integer. The number of dependent variables for the finite HMM must be one.

Parameter Estimation Methods

For the parameter estimation (the learning problem) of the HMM, the HMM procedure supports two methods: the maximum likelihood (ML) method and the maximum a posteriori (MAP) method.

Maximum Likelihood (ML) Method

For the ML method, the parameter estimator is obtained as

$$\hat{\theta} = \arg \max_{\theta} \log (L_T(\mathbf{Y}_1, \dots, \mathbf{Y}_T; \theta))$$

where $L_T(\cdot; \theta)$ is the likelihood of all observations for a particular parameter θ . The $\hat{\theta}$ is obtained through gradient-based optimization; the covariance matrix estimator of the $\hat{\theta}$ is obtained through the inverse of the negative Hessian.

Maximum a Posteriori (MAP) Method

For the MAP method, the parameter estimator is obtained as

$$\hat{\theta} = \arg \max_{\theta} \log (L_T(\mathbf{Y}_1, \dots, \mathbf{Y}_T; \theta)) + \log (p(\theta))$$

where $p(\theta)$ is the prior distribution function of parameter θ . The prior distribution can be specified in the PRIOR statement. The $\hat{\theta}$ is obtained through gradient-based optimization; the covariance matrix estimator of the $\hat{\theta}$ is obtained through the inverse of the negative Hessian.

In theory, the likelihood of the hidden Markov model (HMM) is unbounded and “the ML estimator as a global maximizer of the likelihood function does not exist” (Frühwirth-Schnatter 2006, page 173). The introduction of the proper prior distribution of the parameters in the MAP method can solve the unboundedness problem.

Prior Distribution of the Initial State Probability Vector

The prior distribution of the initial state probability vector (ISPV), π , is the Dirichlet distribution,

$$\pi \sim \text{Dir}(\alpha)$$

where the hyperparameter α is the $K \times 1$ vector and K is the number of states. The probability density function (PDF) of the Dirichlet distribution is

$$f(x, \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^K x_i^{\alpha_i - 1}$$

where $x = (x_1 \cdots x_K)'$, $x_i > 0, i = 1, \dots, K$, and $\sum_{i=1}^K x_i = 1$; $\alpha = (\alpha_1 \cdots \alpha_K)'$ and $\alpha_i > 0, i = 1, \dots, K$; $B(\alpha)$ is the beta function,

$$B(\alpha) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}$$

Prior Distribution of the Transition Probability Matrix

The prior distribution of each row of the transition probability matrix (TPM), $\mathbf{A}$, is the Dirichlet distribution,

$$a'_i \sim \text{Dir}(\alpha^{(i)}), i = 1, \dots, K$$

where a_i is the i th row of the TPM $\mathbf{A}$, $\alpha^{(i)}$ is the $K \times 1$ vector, and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{A} \sim \text{Dir}(\tilde{\mathbf{A}})$$

where the hyperparameter $\tilde{\mathbf{A}} = (\alpha^{(1)} \dots \alpha^{(K)})'$, which is the $K \times K$ matrix that results from stacking the transpose of $\alpha^{(i)}$, $i = 1, \dots, K$.

Prior Distribution of the Category Probability Matrix

The prior distribution of each row of the category probability matrix (CPM), $\mathbf{B}$, is the Dirichlet distribution,

$$b'_i \sim \text{Dir}(b^{(i)}), i = 1, \dots, K$$

where b_i is the i th row of the CPM, $\mathbf{B}$; $b^{(i)}$ is the $M \times 1$ vector; M is the number of categories at a state; and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{B} \sim \text{Dir}(\tilde{\mathbf{B}})$$

where the hyperparameter $\tilde{\mathbf{B}} = (b^{(1)} \dots b^{(K)})'$, which is the $K \times M$ matrix that results from stacking the transpose of $b^{(i)}$, $i = 1, \dots, K$.

Prior Distribution of the Mixture Component Probabilities

The prior distribution of each row of the mixture component probabilities (MCP), $\mathbf{W}$, is the Dirichlet distribution,

$$w'_i \sim \text{Dir}(\alpha^{(i)}), i = 1, \dots, K$$

where w_i is the i th row of the MCP $\mathbf{W}$, $\alpha^{(i)}$ is the $M \times 1$ vector, M is the number of components at a state, and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{W} \sim \text{Dir}(\tilde{\mathbf{A}})$$

where the hyperparameter $\tilde{\mathbf{A}} = (\alpha^{(1)} \dots \alpha^{(K)})'$, which is the $K \times M$ matrix that results from stacking the transpose of $\alpha^{(i)}$, $i = 1, \dots, K$.

Prior Distribution of the Observation Parameters

Gaussian HMM In the Gaussian HMM, the observation parameters are $\{\mu_i, \Sigma_i\}, i = 1, \dots, K$, where μ_i and Σ_i are the mean and covariance parameters for state i . The prior distribution of μ_i and Σ_i is the normal-inverse-Wishart (NIW) distribution,

$$\mu_i, \Sigma_i \sim \text{NIW}(\tilde{\mu}_i, \tilde{\Sigma}_i, \tilde{\kappa}_i, \tilde{\nu}_i), i = 1, \dots, K$$

which is equivalent to

$$\begin{aligned}\mu_i | \Sigma_i &\sim N(\tilde{\mu}_i, \tilde{\kappa}_i^{-1} \Sigma_i) \\ \Sigma_i &\sim \text{IW}(\tilde{\Sigma}_i, \tilde{\nu}_i)\end{aligned}$$

where the hyperparameter $\tilde{\mu}_i$ is a $k_y \times 1$ vector; the hyperparameter $\tilde{\Sigma}_i$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_i$ is a positive scalar; the hyperparameter $\tilde{\nu}_i$ is a positive scalar, $\tilde{\nu}_i > k_y + 3$; K is the number of states; and k_y is the number of dependent variables.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned}f_{\text{NIW}}(\mu, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\ f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) &= \frac{1}{\sqrt{(2\pi)^k |\tilde{\kappa}^{-1} \Sigma|}} \exp\left(-\frac{1}{2}(\mu - \tilde{\mu})'(\tilde{\kappa}^{-1} \Sigma)^{-1}(\mu - \tilde{\mu})\right) \\ f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu}k}{2}} \Gamma_k(\frac{\tilde{\nu}}{2})} |\Sigma|^{-\frac{\tilde{\nu}+k+1}{2}} \exp\left(-\frac{1}{2}\text{tr}(\tilde{\Sigma} \Sigma^{-1})\right)\end{aligned}$$

where the parameter μ is a $k \times 1$ vector; the parameter Σ is a $k \times k$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k \times k$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a positive scalar; and the hyperparameter $\tilde{\nu}$ is a positive scalar, $\tilde{\nu} > k + 3$. The $\Gamma_k(\cdot)$ is the multivariate gamma function, which is defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^k \Gamma\left(x + \frac{1-j}{2}\right)$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned}\tilde{\mu} &= (\tilde{\mu}_1 \cdots \tilde{\mu}_K)' \\ \tilde{\Sigma} &= (\tilde{\Sigma}_1 \cdots \tilde{\Sigma}_K)' \\ \tilde{\kappa} &= (\tilde{\kappa}_1 \cdots \tilde{\kappa}_K)' \\ \tilde{\nu} &= (\tilde{\nu}_1 \cdots \tilde{\nu}_K)'\end{aligned}$$

where $\tilde{\mu}$ is a $K \times k_y$ matrix, $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix, $\tilde{\kappa}$ is a $K \times 1$ vector, $\tilde{\nu}$ is a $K \times 1$ vector, K is the number of states, and k_y is the number of dependent variables.

GM HMM In the GM HMM, the mean and covariance parameters for component j at state i are $\{\mu_{ij}, \Sigma_{ij}\}, i = 1, \dots, K, j = 1, \dots, M$, where M is the number of components at a state and K is the number of states.

The prior distribution of μ_{ij} and Σ_{ij} is the normal-inverse-Wishart (NIW) distribution,

$$\mu_{ij}, \Sigma_{ij} \sim \text{NIW}(\tilde{\mu}_{ij}, \tilde{\Sigma}_{ij}, \tilde{\kappa}_{ij}, \tilde{\nu}_{ij}), i = 1, \dots, K, j = 1, \dots, M$$

which is equivalent to

$$\begin{aligned} \mu_{ij} | \Sigma_{ij} &\sim N(\tilde{\mu}_{ij}, \tilde{\kappa}_{ij}^{-1} \Sigma_{ij}) \\ \Sigma_{ij} &\sim \text{IW}(\tilde{\Sigma}_{ij}, \tilde{\nu}_{ij}) \end{aligned}$$

where the hyperparameter $\tilde{\mu}_{ij}$ is a $k_y \times 1$ vector; the hyperparameter $\tilde{\Sigma}_{ij}$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_{ij}$ is a positive scalar; the hyperparameter $\tilde{\nu}_{ij}$ is a positive scalar, $\tilde{\nu}_{ij} > k_y + 3$; and k_y is the number of dependent variables.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned} f_{\text{NIW}}(\mu, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\ f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) &= \frac{1}{\sqrt{(2\pi)^k |\tilde{\kappa}^{-1} \Sigma|}} \exp\left(-\frac{1}{2}(\mu - \tilde{\mu})'(\tilde{\kappa}^{-1} \Sigma)^{-1}(\mu - \tilde{\mu})\right) \\ f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu}k}{2}} \Gamma_k\left(\frac{\tilde{\nu}}{2}\right)} |\Sigma|^{-\frac{\tilde{\nu}+k+1}{2}} \exp\left(-\frac{1}{2}\text{tr}(\tilde{\Sigma} \Sigma^{-1})\right) \end{aligned}$$

where the parameter μ is a $k \times 1$ vector; the parameter Σ is a $k \times k$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k \times k$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a positive scalar; and the hyperparameter $\tilde{\nu}$ is a positive scalar, $\tilde{\nu} > k + 3$. The $\Gamma_k(\cdot)$ is the multivariate gamma function defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^k \Gamma\left(x + \frac{1-j}{2}\right)$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned} \tilde{\mu} &= (\tilde{\mu}_{11} \cdots \tilde{\mu}_{1M} \tilde{\mu}_{21} \cdots \tilde{\mu}_{2M} \cdots \tilde{\mu}_{K1} \cdots \tilde{\mu}_{KM})' \\ \tilde{\Sigma} &= (\tilde{\Sigma}_{11} \cdots \tilde{\Sigma}_{1M} \tilde{\Sigma}_{21} \cdots \tilde{\Sigma}_{2M} \cdots \tilde{\Sigma}_{K1} \cdots \tilde{\Sigma}_{KM})' \\ \tilde{\kappa} &= (\tilde{\kappa}_{11} \cdots \tilde{\kappa}_{1M} \tilde{\kappa}_{21} \cdots \tilde{\kappa}_{2M} \cdots \tilde{\kappa}_{K1} \cdots \tilde{\kappa}_{KM})' \\ \tilde{\nu} &= (\tilde{\nu}_{11} \cdots \tilde{\nu}_{1M} \tilde{\nu}_{21} \cdots \tilde{\nu}_{2M} \cdots \tilde{\nu}_{K1} \cdots \tilde{\nu}_{KM})' \end{aligned}$$

where $\tilde{\mu}$ is a $KM \times k_y$ matrix, $\tilde{\Sigma}$ is a $k_y KM \times k_y$ matrix, $\tilde{\kappa}$ is a $KM \times 1$ vector, $\tilde{\nu}$ is a $KM \times 1$ vector, M is the number of components at a state, K is the number of states, and k_y is the number of dependent variables.

Regime-Switching Regression Model and Regime-Switching Autoregression Model In regime-switching regression models and regime-switching autoregression models, the observation parameters are $\{\mathbf{B}_i, \Sigma_i\}, i = 1, \dots, K$, where $\mathbf{B}_i$ and Σ_i are the mean and covariance parameters for state i . Because $\mathbf{B}_i z_t = (z_t' \otimes I_{k_y}) \text{vec}(\mathbf{B}_i) = (z_t' \otimes I_{k_y}) \beta_i$, where vec is the column stacking operator, the observation parameters in this section are denoted by $\{\beta_i, \Sigma_i\}, i = 1, \dots, K$. The prior distribution of β_i and Σ_i is the normal-inverse-Wishart (NIW) distribution,

$$\beta_i, \Sigma_i \sim \text{NIW}(\tilde{\mu}_i, \tilde{\Sigma}_i, \tilde{\kappa}_i, \tilde{\nu}_i), i = 1, \dots, K$$

which is equivalent to

$$\begin{aligned} \beta_i | \Sigma_i &\sim N(\tilde{\mu}_i, \tilde{\kappa}_i^{-1} \otimes \Sigma_i) \\ \Sigma_i &\sim \text{IW}(\tilde{\Sigma}_i, \tilde{\nu}_i) \end{aligned}$$

where the hyperparameter $\tilde{\mu}_i$ is a $k_y k_z \times 1$ vector; the hyperparameter $\tilde{\Sigma}_i$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_i$ is a $k_z \times k_z$ symmetric positive definite matrix; the hyperparameter $\tilde{\nu}_i$ is a positive scalar, $\tilde{\nu}_i > k_y + 3$; K is the number of states; k_y is the number of dependent variables; and k_z is the number of regressors. For information about how to calculate k_z , see the sections “[Regime-Switching Regression Model](#)” on page 1273 and “[Regime-Switching Autoregression Model](#)” on page 1274.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned} f_{\text{NIW}}(\beta, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_N(\beta, \tilde{\mu}, \tilde{\kappa}^{-1} \otimes \Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\ f_N(\beta, \tilde{\mu}, \tilde{\kappa}^{-1} \otimes \Sigma) &= \frac{1}{\sqrt{(2\pi)^{k_1 k_2} |\tilde{\kappa}^{-1} \otimes \Sigma|}} \exp\left(-\frac{1}{2}(\beta - \tilde{\mu})'(\tilde{\kappa}^{-1} \otimes \Sigma)^{-1}(\beta - \tilde{\mu})\right) \\ f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu} k_1}{2}} \Gamma_{k_1}(\frac{\tilde{\nu}}{2})} |\Sigma|^{-\frac{\tilde{\nu} + k_1 + 1}{2}} \exp\left(-\frac{1}{2} \text{tr}(\tilde{\Sigma} \Sigma^{-1})\right) \end{aligned}$$

where the parameter β is a $k_1 k_2 \times 1$ vector; the parameter Σ is a $k_1 \times k_1$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k_1 k_2 \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k_1 \times k_1$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a $k_2 \times k_2$ symmetric positive definite matrix; and the hyperparameter $\tilde{\nu}$ is a positive scalar, where $\tilde{\nu} > k_1 + 3$. The $\Gamma_k(\cdot)$ is the multivariate gamma function defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^k \Gamma\left(x + \frac{1-j}{2}\right)$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned} \tilde{\mu} &= (\tilde{\mu}_1 \cdots \tilde{\mu}_K)' \\ \tilde{\Sigma} &= (\tilde{\Sigma}_1 \cdots \tilde{\Sigma}_K)' \\ \tilde{\kappa} &= (\tilde{\kappa}_1 \cdots \tilde{\kappa}_K)' \\ \tilde{\nu} &= (\tilde{\nu}_1 \cdots \tilde{\nu}_K)' \end{aligned}$$

where $\tilde{\mu}$ is a $K \times k_y k_z$ matrix, $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix, $\tilde{\kappa}$ is a $k_z K \times k_z$ matrix, $\tilde{\nu}$ is a $K \times 1$ vector, K is the number of states, k_y is the number of dependent variables, and k_z is the number of regressors.

Poisson HMM In the Poisson HMM, the mean and variance for state i are both equal to $\lambda_i, i = 1, \dots, K$. The prior distribution of λ_i , the gamma distribution, is

$$\lambda \sim \text{gamma}(\alpha, \beta)$$

where the hyperparameter α, β is the $K \times 2$ matrix and K is the number of states. The probability density function (PDF) of the gamma distribution is

$$f(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{(\alpha-1)} \exp(-\beta\lambda)$$

where $\lambda > 0$.

Optimization Algorithms

The HMM procedure supports four types of optimization algorithms: the expectation-maximization (EM) algorithm, the active-set algorithm, the interior point algorithm, and the stochastic gradient descent (SGD) algorithm.

The EM algorithm is an iterative method of finding the local maximum likelihood estimates of parameters in models. After initializing the parameters—the ISPV π , the TPM A , the mean μ , and the covariance Σ —the EM algorithm iterates between two steps: the expectation (E) step and the maximization (M) step.

In the E step, γ_t and ξ_t are computed using α_t and β_t , which are based on the forward and backward algorithm:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^K \alpha_t(j)\beta_t(j)}$$

$$\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}p(y_{t+1}|s_{t+1}=j)\beta_{t+1}(j)}{\sum_{i=1}^K \sum_{j=1}^K \alpha_t(i)a_{ij}p(y_{t+1}|s_{t+1}=j)\beta_{t+1}(j)}$$

where $i, j \in [1, K]$. In the M step, the parameters are updated using γ_t and ξ_t from the E step: When ESTISPV is specified, ISPV is updated according to $\pi_i = \gamma_1(i)$. For finite-state-space and homogeneous HMM, the stationary distribution of the Markov chain is the value of π such that $\pi' A = \pi'$, where A is the transition probability matrix (TPM); that is, π is the leading left eigenvector of A . Other parameters are updated according to the following equations, if there are no sections in the data:

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\mu_i = \frac{\sum_{t=1}^T y_t * \gamma_t(i)}{\sum_{t=1}^T \gamma_t(i)}$$

$$\Sigma_i = \frac{\sum_{t=1}^T (y_t - \mu_i) * (y_t - \mu_i)^T * \gamma_t(i)}{\sum_{t=1}^T \gamma_t(i)}$$

Otherwise, the preceding equations become

$$\begin{aligned}\pi_i &= \frac{\sum_{s=1}^{\text{nsec}} \gamma_1^{(s)}(i)}{\text{nsec}} \\ a_{ij} &= \frac{\sum_{s=1}^{\text{nsec}} \sum_{t=1}^{T^{(s)}-1} \xi_t(i, j)}{\sum_{s=1}^{\text{nsec}} \sum_{t=1}^{T^{(s)}-1} \gamma_t^{(s)}(i)} \\ \mu_i &= \frac{\sum_{s=1}^{\text{nsec}} \sum_{t=1}^{T^{(s)}} y_t * \gamma_t^{(s)}(i)}{\sum_{s=1}^{\text{nsec}} \sum_{t=1}^{T^{(s)}} \gamma_t^{(s)}(i)} \\ \Sigma_i &= \frac{\sum_{s=1}^{\text{nsec}} \sum_{t=1}^{T^{(s)}} (y_t - \mu_i) * (y_t - \mu_i)^T * \gamma_t^{(s)}(i)}{\sum_{s=1}^{\text{nsec}} \sum_{t=1}^{T^{(s)}} \gamma_t^{(s)}(i)}\end{aligned}$$

where s is the index of section and nsec is the number of sections. Notice that μ_i is used to demean y_t when Σ_i is updated. The EM algorithm is supported only for the Gaussian HMM and the maximum likelihood method.

Starting in version 2023.05, you can execute the EM algorithm in parallel on multiple threads and nodes if you divide the data into sections and specify the `SECTION=` option in the `ID` statement. In this case, each thread is assigned some sections of the data on the available nodes. Or if backward compatibility is a concern, you can run the EM algorithm in nonparallel mode by specifying the `SINGLE` option in the `OPTIMIZE` statement. Parallel mode can reduce run time and improve the algorithm's performance. Also, there might be a numerical difference between your results in these two modes.

For more information about the active set and interior point algorithms and their related options, see the chapter “The Nonlinear Programming Solver” in *SAS/OR User's Guide: Mathematical Programming*. For more information about the SGD algorithm, see the section “[OPTIMIZE Statement](#)” on page 1259.

Distributed Multistart

In the distributed multistart algorithm, the local optimizations that start from various initial points run concurrently. Each of these local optimizations requires evaluation of the objective or constraints as well as their first or second derivatives at different points, which are referred to as function evaluations. If function evaluations are performed in distributed mode, as in the case of `PROC HMM`, the evaluations necessary for each local optimization are performed in a subsession that stems from the current main CAS session. You can specify the `SESSNODES=`, `FESESSIONS=`, and `FESESSNODES=` options to achieve a balanced and more efficient usage of the computing resources, in particular the available nodes and the threads on each node.

Suppose that there are a total of W worker nodes in the current main CAS session and you specify `SESSNODES= $m$` , `FESESSIONS= $s$` , and `FESESSNODES= $n$` . Then m out of W worker nodes are dedicated to the local optimizations of the multistart algorithm. Hence the total number of local optimizations that run concurrently is

$$N_{\text{locopt}} = \sum_{i=1}^m t_i$$

where t_i is the number of threads on node i . In addition, s subsessions, each of which has n worker nodes, are created and dedicated to function evaluations. When a function evaluation is requested in one of N_{locopt} local

optimizations, the request is sent to one of the s subsessions. Use the following guidelines when you select the values of the `SESSNODES=`, `FESESSIONS=`, and `FESESSNODES=` options:

- The value of m or n should never be greater than W .
- The total number of local optimizations N_{locopt} should be as close to s as possible. This allows a good balance between the resources for local optimizations and function evaluations.
- If you have a limited number of worker nodes W , then in order to achieve the correct balance, preference is often given to a combination that favors a larger s and smaller n over one that favors a smaller s and larger n . This helps reduce the number of function evaluation requests that are waiting to be processed in a function evaluation subsession.
- Avoid a scenario in which $m + sn$ is far greater than W . In such a scenario, competition for resources would occur and would degrade performance.

Matrix Expression

The `INITIAL` and `PRIOR` statements operate on matrices. That is, you can specify the parameter matrices or constant matrices through the `INITIAL` and `PRIOR` statements' built-in operators and functions. You can add elements of the matrices **A** and **B** by using the expression **A+B**, perform matrix multiplication by using the expression **A*B**, and perform elementwise multiplication by using the expression **A#B**. You can get the diagonal elements of the matrix **A** by using the function `DIAG (A)`, and you can get the $n \times n$ identity matrix by using the function `I (n)`.

Each equation is written as a matrix expression that is composed of constants, operators, and functions.

Constants

Constants are either scalar constants (such as -1.2 , 0.3 , and so on) or matrix constants enclosed in braces (such as the 2×2 matrix `{1 2, 3 4}` or the 1×3 row vector `{-0.2 5.3 12}`).

The matrix constant cannot be the first item in the `INITIAL` statement unless you enclose it in parentheses. For example, you cannot specify the following statement:

```
initial {-0.1 -0.2, -0.3 -0.4} = MU;
```

You can specify the preceding example by enclosing the first matrix constant in parentheses, as follows:

```
initial ({-0.1 -0.2, -0.3 -0.4}) = MU;
```

Operators

Operators define the operations on operands. [Table 20.2](#) lists all built-in operators that the `INITIAL` statement supports.

Table 20.2 Operators

Operator	Name	Description
+	Addition	Adds corresponding matrix elements
=	Comparison, equal	Compares matrix elements
<	Comparison, less than	Compares matrix elements
<=	Comparison, not greater than	Compares matrix elements
>	Comparison, greater than	Compares matrix elements
>=	Comparison, not less than	Compares matrix elements
	Concatenation, horizontal	Concatenates matrices horizontally
//	Concatenation, vertical	Concatenates matrices vertically
@	Direct product	Takes the direct product of two matrices
~	Distribution	Links the distribution function to the matrix elements
:	Index creation	Creates an index vector
#	Multiplication, elementwise	Performs elementwise multiplication
*	Multiplication, matrix	Performs matrix multiplication
—	Sign reverse	Reverses the signs of elements
[]	Subscripts	Selects submatrices
—	Subtraction	Subtracts corresponding matrix elements
`	Transpose	Transposes a matrix

For more information about each operator, see the section “[Details of Operators](#)” on page 1316.

[Table 20.3](#) shows the precedence of matrix operators.

Table 20.3 Precedence of Operators

Priority Group	Operators
I (highest)	[] (subscripts) ` (transpose)
II	— (sign reverse)
III	* # @
IV	— (subtraction) +
V	// :
VI (lowest)	= < <= > >= ~

Each equation can be a compound expression that involves several matrix operators and operands. The rules for evaluating compound expressions are as follows:

- Evaluation follows the order of operator precedence as described in [Table 20.3](#). Group I has the highest priority; that is, Group I operators are evaluated first. Group II operators are evaluated after Group I operators, and so on. For example, $1 + 2 * 3$ returns 7.
- If neighboring operators in an expression have equal precedence, the expression is evaluated from left to right, except for the Group I operators. For example, $1 - 2 - 3$ returns -4 .
- All expressions in parentheses are evaluated first, following the two preceding rules. For example, $3 * (2 + 1)$ returns 9.

Functions

Functions are divided into two types. One type of function refers to parameters to be estimated; examples are **MU**(**K**, **I**) and **SIGMA**(**K**, **I**, **J**). The other type does not; examples are **I**(**n**) and **DIAG**(**A**).

Functions that refer to parameters are listed in Table 20.4. The arguments of functions can be matrices. The simplest case, scalar arguments, is discussed first. For convenience, the scalar indices **i** and **j** refer to the position of the element in the coefficient matrix, the scalar **l** refers to the lag value, the scalar **c** refers to the component index, and the scalar **k** refers to the state value.

Table 20.4 Functions That Refer to Parameters

Function	Description
AR (k , l , i , j)	Autoregressive parameter of the lag l of the j th dependent variable, $y_{j,t-l}$, to the i th dependent variable at time t , y_{it} , for state k in regime-switching autoregression
CONST (k , i)	Intercept parameter of the i th time series, y_{it} , for state k in regime-switching regression or regime-switching autoregression, or the alias of MU for Gaussian HMM or GM HMM
COV (k , i , j)	Covariance of innovations parameter between the i th and j th error processes for state k in regime-switching regression or regime-switching autoregression, or the alias of SIGMA for Gaussian HMM or GM HMM
CPM (i , j)	The (i, j) th element in the category probability matrix in the finite HMM
LAMBDA (K)	Mean parameter of the time series, y_{it} , for state k in the Poisson HMM
ISPV (i)	The i th element in the initial state probability vector
LTREND (k , i)	Linear trend parameter of the i th time series, y_{it} , for state k in regime-switching regression or regime-switching autoregression when the TREND = option is specified in the MODEL statement
MCP (i , j)	The (i, j) th element in the mixture component probabilities in the GM HMM
MU (k , i)	Mean parameter of the i th time series, y_{it} , for state k in the Gaussian HMM, or alias for CONST for regime-switching regression or regime-switching autoregression
MU (k , c , i)	Mean parameter of the i th time series, y_{it} , for the c th component at state k in the GM HMM
QTREND (k , i)	Quadratic trend parameter of the i th time series, y_{it} , for state k in regime-switching regression or regime-switching autoregression when TREND = QUAD is specified in the MODEL statement
SD (k , i , j)	Alias of SDUMMY
SDUMMY (k , i , j)	The j th seasonal dummy of the i th time series at time t , y_{it} , for state k in regime-switching regression or regime-switching autoregression, where $j = 1, \dots, (nseason-1)$, where $nseason$ is the value of the NSEASON = option in the MODEL statement
SIGMA (k , i , j)	Covariance parameter between the i th and j th time series, y_{it} and y_{jt} , for state k in the Gaussian HMM, or the alias of COV for regime-switching regression or regime-switching autoregression
SIGMA (k , c , i , j)	Covariance parameter between the i th and j th time series, y_{it} and y_{jt} , for c th component at state k in the GM HMM
TPM (i , j)	The (i, j) th element in the transition probability matrix

Table 20.4 *continued*

Function	Description
XL (k , l , i , j)	Exogenous parameter of the lag l of the j th exogenous variable, $x_{j,t-l}$, to the i th dependent variable at time t , y_{it} , for state k in regime-switching regression or regime-switching autoregression

The functions that refer to parameters, as shown in Table 20.4, are valid only when the parameters are included in the model. For example, the **ISPV** function is valid only when you specify the **ESTISPV** option in the **MODEL** statement; otherwise, the HMM procedure issues errors and then stops.

The functions that refer to parameters, as shown in Table 20.4, accept vector and matrix arguments and return the matrix that is constructed by the corresponding parameters. According to the number of arguments, the following list shows what matrix a function returns when the arguments are vectors:

- A function, **FUNC**₁, that has one vector argument I , where $I = (i_1 \ i_2 \ \dots \ i_{n_I})'$, returns a vector $R = (r_1 \ r_2 \ \dots \ r_{n_I})'$, where $r_k = \mathbf{FUNC}_1(i_k)$, $k = 1, \dots, n_I$. **ISPV** is a type of **FUNC**₁.
- A function, **FUNC**₂, that has two vector arguments I and J , where $I = (i_1 \ i_2 \ \dots \ i_{n_I})'$ and $J = (j_1 \ j_2 \ \dots \ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_J} \\ \cdots & & & \\ r_{n_I,1} & r_{n_I,2} & \cdots & r_{n_I,n_J} \end{pmatrix}$$

where $r_{k,m} = \mathbf{FUNC}_2(i_k, j_m)$, $k = 1, \dots, n_I$, $m = 1, \dots, n_J$. **CONST**, **LTREND**, **MU**, **QTREND**, and **TPM** are types of **FUNC**₂.

- A function, **FUNC**_N, that has N vector arguments $I_1, I_2, \dots$, and I_N , where $N > 2$ and N is odd and where $I_n = (i_{n,1} \ i_{n,2} \ \dots \ i_{n,j_n})'$, $n = 1, \dots, N$, returns a block matrix,

$$R = \begin{pmatrix} B_1 \\ B_2 \\ \cdots \\ B_{j_n} \end{pmatrix}$$

where $B_k = \mathbf{FUNC}_N(i_{1,k}, I_2, \dots, I_N)$, $k = 1, \dots, j_1$, and B_k is constructed in the same way as **FUNC**_{N-1}($I_2, \dots, I_N$).

For example, a function, **FUNC**₃, that has three vector arguments K , I , and J , where $K = (k_1 \ k_2 \ \dots \ k_{n_K})'$, $I = (i_1 \ i_2 \ \dots \ i_{n_I})'$, and $J = (j_1 \ j_2 \ \dots \ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_J} \\ \cdots & & & \\ r_{n_K n_I,1} & r_{n_K n_I,2} & \cdots & r_{n_K n_I,n_J} \end{pmatrix}$$

where $r_{u,v} = \mathbf{FUNC}_3(k_{d_1(u,n_I)}, i_{d_2(u,n_I)}, j_v)$, $u = 1, \dots, n_{Kn_I}$, $v = 1, \dots, n_J$, and $d_1(m, n)$ is the integer such that $d_1(m, n)n \geq m > (d_1(m, n) - 1)n$ and $d_2(m, n) = m - (d_1(m, n) - 1)n$. **COV** and **MU** for GM HMM and **SIGMA** for Gaussian HMM are types of **FUNC**₃.

- A function, **FUNC**_N, that has N vector arguments $I_1, I_2, \dots$, and I_N , where $N > 2$ and N is even and where $I_n = (i_{n,1} \ i_{n,2} \ \dots \ i_{n,j_n})'$, $n = 1, \dots, N$, returns a block matrix,

$$R = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,j_2} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,j_2} \\ \cdots & & & \\ B_{j_1,1} & B_{j_1,2} & \cdots & B_{j_1,j_2} \end{pmatrix}$$

where $B_{k,l} = \mathbf{FUNC}_N(i_{1,k}, i_{2,l}, I_3, \dots, I_N)$, $k = 1, \dots, j_1$, $l = 1, \dots, j_2$, and $B_{k,l}$ is constructed in the same way as **FUNC**_{N-2}($I_3, \dots, I_N$).

For example, a function, **FUNC**₄, that has four vector arguments K, L, I , and J , where $K = (k_1 \ k_2 \ \dots \ k_{n_K})'$, $L = (l_1 \ l_2 \ \dots \ l_{n_L})'$, $I = (i_1 \ i_2 \ \dots \ i_{n_I})'$, and $J = (j_1 \ j_2 \ \dots \ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_L n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_L n_J} \\ \cdots & & & \\ r_{n_{Kn_I},1} & r_{n_{Kn_I},2} & \cdots & r_{n_{Kn_I},n_L n_J} \end{pmatrix}$$

where $r_{u,v} = \mathbf{FUNC}_4(k_{d_1(u,n_I)}, l_{d_1(v,n_J)}, i_{d_2(u,n_I)}, j_{d_2(v,n_J)})$, $u = 1, \dots, n_{Kn_I}$, $v = 1, \dots, n_{Ln_J}$, where $d_1(m, n)$ is the integer such that $d_1(m, n)n \geq m > (d_1(m, n) - 1)n$ and $d_2(m, n) = m - (d_1(m, n) - 1)n$. **AR** and **SIGMA** for GM HMM and **XL** are types of **FUNC**₄.

The functions that refer to parameters can accept empty arguments or omit any number of last arguments. The empty or omitted arguments are replaced by all possible values for those arguments. For example, PROC HMM is used to fit a bivariate three-state Gaussian HMM as follows:

```
model y1 y2 / type=gaussian nstate=3;
```

In order to initialize the second row of **TPM** to be {0.1 0.7 0.2}, you can use the following statement:

```
initial tpm(2, {1 2 3}) = {0.1 0.7 0.2};
```

Taking advantage of empty arguments, you can specify the preceding example as follows:

```
initial tpm(2,) = {0.1 0.7 0.2};
```

To get all coefficients of the covariance matrix for state 2, you can use **SIGMA**(2, {1 2}, {1 2}), or **SIGMA**(2, ,), or **SIGMA**(2). To get all coefficients of exogenous variables on dependent variables, you can use **SIGMA**({1 2 3}, {1 2}, {1 2}), or **SIGMA**(, ,), or **SIGMA**(), or even just **SIGMA**.

Another type of function does not refer to parameters but generates useful matrices. Table 20.5 lists all built-in functions.

Table 20.5 Functions That Do Not Refer to Parameters

Function	Description
DIAG (A)	Creates a diagonal matrix from a vector or extracts the diagonal elements of a matrix
I (n)	Creates an $n \times n$ identity matrix
J (m, n, elem)	Creates an $m \times n$ matrix whose elements are all equal to elem
SHAPE (A, m, n)	Creates an $m \times n$ matrix whose elements are from matrix A

For more information about each function in [Table 20.5](#), see the section “[Details of Functions](#)” on page 1320.

Details of Operators

This section describes all operators that are available in the HMM procedure. Each subsection describes the operator and shows how it is used.

Addition Operator: +

The addition operator (+) computes a new matrix whose elements are the sums of corresponding elements of the two operands. Following are examples of how the addition operator is used:

matrix1 + matrix2	adds the element in the i th row and j th column of the first matrix to the element in the i th row and j th column of the second matrix, for $i = 1, \dots, n$, $j = 1, \dots, p$, where matrix1 and matrix2 are both $n \times p$ matrices. For example, {1 2 3, 4 5 6} + {7 8 9, 10 11 12} results in {8 10 12, 14 16 18}.
matrix + scalar	adds the scalar value to each element of the matrix . For example, you can obtain {2 3 4, 5 6 7} from {1 2 3, 4 5 6} + 1.
matrix + vector	adds the vector value to each row or column of the $n \times p$ matrix : • If you add an $n \times 1$ column vector, each row of the vector is added to each row of the matrix. For example, you can obtain {2 3 4, 5 6 7} from {1 2 3, 4 5 6} + {1, 1}. • If you add a $1 \times p$ row vector, each column of the vector is added to each column of the matrix. For example, you can obtain {2 3 4, 5 6 7} from {1 2 3, 4 5 6} + {1 1 1}.

Comparison Operators: =, <, <=, >, >=

The comparison operators (=, <, <=, >, >=) compare two matrices element by element and return a list of equivalent restrictions on only scalar constants and parameters.

```

matrix1 = matrix2
matrix1 < matrix2
matrix1 <= matrix2
matrix1 > matrix2

```

```
matrix1 >= matrix2
```

For example, the INITIAL statement with matrix expressions

```
initial SIGMA(1, {1, 2}, {1, 2}) = SIGMA(2, {3, 4}, {3, 4});
```

is transformed into the following equivalent INITIAL statement with scalar parameters:

```
initial SIGMA(1, 1, 1) = SIGMA(2, 3, 3),
        SIGMA(1, 1, 2) = SIGMA(2, 3, 4),
        SIGMA(1, 2, 1) = SIGMA(2, 4, 3),
        SIGMA(1, 2, 2) = SIGMA(2, 4, 4);
```

You can also use the comparison operators to conveniently compare all elements of a matrix to a scalar. If either argument is a scalar, then the HMM procedure performs an elementwise comparison between each element of the matrix and the scalar.

You can also compare an $n \times p$ matrix to a row or column vector:

- If the comparison is to an $n \times 1$ column vector, the HMM procedure compares each row of the vector to each row of the matrix.
- If the comparison is to a $1 \times p$ row vector, the HMM procedure compares each column of the vector to each column of the matrix.

For example, the following statements are equivalent:

```
initial SIGMA(1, 4:5, 1:3) = 0.2;

initial SIGMA(1, 4:5, 1:3) = {0.2, 0.2};

initial SIGMA(1, 4:5, 1:3) = {0.2 0.2 0.2};
```

Concatenation Operator, Horizontal: ||

The horizontal concatenation operator (||) produces a new matrix by horizontally joining **matrix1** and **matrix2**.

```
matrix1 || matrix2
```

The matrices must have the same number of rows, which is also the number of rows in the new matrix. The number of columns in the new matrix is the number of columns in **matrix1** plus the number of columns in **matrix2**.

For example, {1 1 1, 7 7 7} || {0 0 0, 8 8 8} returns {1 1 1 0 0 0, 7 7 7 8 8 8}.

Concatenation Operator, Vertical: //

The vertical concatenation operator (//) produces a new matrix by vertically joining **matrix1** and **matrix2**.

```
matrix1 // matrix2
```

The matrices must have the same number of columns, which is also the number of columns in the new matrix. The number of rows in the new matrix is the number of rows in **matrix1** plus the number of rows in **matrix2**.

For example, {1 1 1} // {0 0 0, 8 8 8} returns {1 1 1, 0 0 0, 8 8 8}.

Direct Product Operator: @

The direct product operator (@) computes a new matrix that is the direct product (also called the Kronecker product) of **matrix1** and **matrix2**.

matrix1 @ matrix2

For matrices **A** and **B**, the direct product is denoted by $\mathbf{A} \otimes \mathbf{B}$. The number of rows in the new matrix equals the product of the number of rows in **matrix1** and the number of rows in **matrix2**; the number of columns in the new matrix equals the product of the number of columns in **matrix1** and the number of columns in **matrix2**.

Specifically, if **A** is an $n \times p$ matrix and **B** is an $m \times q$ matrix, then the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is the following $nm \times pq$ block matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{11}B & \cdots & A_{1p}B \\ \vdots & \ddots & \vdots \\ A_{n1}B & \cdots & A_{np}B \end{bmatrix}$$

For example, $\{1 \ 2, \ 3 \ 4\} @ \{0 \ 2\}$ returns $\{0 \ 2 \ 0 \ 4, \ 0 \ 6 \ 0 \ 8\}$, and $\{0 \ 2\} @ \{1 \ 2, \ 3 \ 4\}$ returns $\{0 \ 0 \ 2 \ 4, \ 0 \ 0 \ 6 \ 8\}$. Note that the direct product of two matrices is not commutative.

Distribution Operator: ~

The distribution operator links the matrix to its distribution function.

matrix ~ distribution-function

This operator is used in the PRIOR statement. For more information, see the section “PRIOR Statement” on page 1264.

For example, $\mathbf{ISPV} \sim \mathbf{DIR}(\{1, 1, 1\})$ means that the initial state probability vector (ISPV) follows a Dirichlet distribution with an argument of a 3×1 vector $\{1, 1, 1\}$.

Index Creation Operator: :

The index creation operator (:) creates a column vector whose first element is **value1**, second element is **value1+1**, and so on, until the last element, which is less than or equal to **value2**.

value1 : value2

For example, $3 : 6$ returns $\{3 \ 4 \ 5 \ 6\}$.

If **value1** is greater than **value2**, a reverse-order index is created. For example, $6 : 3$ returns $\{6 \ 5 \ 4 \ 3\}$.

Neither **value1** nor **value2** is required to be an integer.

Multiplication Operator, Elementwise: #

The elementwise multiplication operator (#) computes a new matrix whose elements are the products of the operands. Following are examples of how the elementwise multiplication operator is used:

matrix1 # matrix2 computes a new matrix whose elements are the products of the corresponding elements of **matrix1** and **matrix2**.

For example, $\{1 \ 2, \ 3 \ 4\} \# \{4 \ 8, \ 0 \ 5\}$ returns $\{4 \ 16, \ 0 \ 20\}$.

matrix # scalar multiplies each element in **matrix** by the **scalar** value.

matrix # vector multiplies each row or column of the $n \times p$ matrix by a corresponding element of the vector:

- If you multiply by an $n \times 1$ column vector, each row of the matrix is multiplied by the corresponding row of the vector.
- If you multiply by a $1 \times p$ row vector, each column of the matrix is multiplied by the corresponding column of the vector.

For example, a 2×3 matrix can be multiplied on either side by a 2×3 , 1×3 , 2×1 , or 1×1 scalar.

The product of elementwise multiplication is also known as the Schur or Hadamard product. Elementwise multiplication (which uses the # operator) should not be confused with matrix multiplication (which uses the * operator).

Multiplication Operator, Matrix: *

The matrix multiplication operator (*) computes a new matrix by performing matrix multiplication.

matrix1 * matrix2

The first matrix must have the same number of columns as the second matrix has rows. The new matrix has the same number of rows as the first matrix and the same number of columns as the second matrix. That is, if **A** is an $n \times p$ matrix and **B** is a $p \times m$ matrix, then the product **A * B** is an $n \times m$ matrix. The (i, j) element of the product is the sum $\sum_{k=1}^p A_{ik} B_{kj}$.

For example, **{1 2, 3 4} * {1, 2}** returns **{5, 11}**.

Sign Reversal Operator: -

The sign reversal operator (-) computes a new matrix whose elements are formed by reversing the sign of each element in **matrix**. The sign reversal operator is also called the unary minus operator.

- matrix

For example, **-{-1 7 6, 2 0 -8}** returns **{1 -7 -6, -2 0 8}**.

Subscripts: []

Subscripts are used with matrices to select submatrices, where **rows**, **columns**, and **elements** are expressions that evaluate to scalars or vectors. If these expressions are numeric, they must contain valid subscript values of rows and columns, or the indices, in the argument matrix.

matrix[rows, columns]

matrix[elements]

For example, **{1 2 3, 4 5 6, 7 8 9}[2,3]** returns **6**; **{1 2 3, 4 5 6, 7 8 9}[2,1:3]** returns **{4 5 6}**; and **{1 2 3, 4 5 6, 7 8 9}[,3]** returns **{3, 6, 9}**. Because the HMM procedure stores matrices in row-major order, **{11 22 33, 44 55 66, 77 88 99} [{3 5 9}]** returns **{33, 55, 99}**.

Subtraction Operator: -

The subtraction operator (-) computes a new matrix whose elements are formed by subtracting the corresponding elements of the second operand from the first operand.

matrix1 - matrix2	computes a new matrix whose elements are formed by subtracting the corresponding elements of matrix2 from those of matrix1 . For example, {1 2 3, 4 5 6} - {1 1 1, 1 1 1} returns {0 1 2, 3 4 5}.
matrix - scalar	subtracts the scalar value from each element of the matrix . For example, {1 2 3, 4 5 6} - 1 returns {0 1 2, 3 4 5}.
matrix - vector	subtracts the vector from each element of the matrix : • If you subtract an $n \times 1$ column vector, each row of the vector is subtracted from each row of the matrix. For example, {1 2 3, 4 5 6} - {1, 1} returns {0 1 2, 3 4 5}. • If you subtract a $1 \times p$ row vector, each column of the vector is subtracted from each column of the matrix. For example, {1 2 3, 4 5 6} - {1 1 1} returns {0 1 2, 3 4 5}.

Transpose Operator: `

The transpose operator, denoted by the backquote (`), exchanges the rows and columns of **matrix**, producing the transpose of **matrix**.

matrix`

If v is the value in the i th row and j th column of **matrix**, then the transpose of **matrix** contains v in the j th row and i th column. If **matrix** contains n rows and p columns, the transpose has p rows and n columns.

For example, {1 2, 3 4, 5 6}` returns {1 3 5, 2 4 6}.

Details of Functions**DIAG Function**

The **DIAG** function creates a diagonal matrix from a vector or extracts the diagonal elements of a matrix.

DIAG(matrix)

The **matrix** argument can be either a square matrix or a vector:

- If **matrix** is a square matrix, the **DIAG** function creates a vector from the diagonal elements of the matrix.
For example, **DIAG**({1 2 3, 4 5 6, 7 8 9}) returns {1, 5, 9}.
- If **matrix** is a vector, the **DIAG** function creates a matrix whose diagonal elements are the values in the vector. All off-diagonal elements are zeros.
For example, **DIAG**({1 5 9}) or **DIAG**({1, 5, 9}) returns {1 0 0, 0 5 0, 0 0 9}.

You can call the **DIAG** function repeatedly. For example, **DIAG**(**DIAG**({1 2 3, 4 5 6, 7 8 9})) returns {1 0 0, 0 5 0, 0 0 9}.

I Function

The **I** function creates an identity matrix that contains **dim** rows and columns.

I(dim)

The diagonal elements of an identity matrix are ones; all other elements are zeros. The value of **dim** must be an integer greater than or equal to 1. Noninteger operands are rounded to the nearest integer.

For example, **I(3)** returns {1 0 0, 0 1 0, 0 0 1}.

J Function

The **J** function creates a matrix that contains **nrow** rows and **ncol** columns, in which all elements are equal to **value**.

J(nrow, ncol, value)

The arguments **nrow** and **ncol** are both integers; **value** can be any expression that returns a linear combination of scalar constants and parameters.

For example, **J(2, 3, 1)** returns {1 1 1, 1 1 1}.

SHAPE Function

The **SHAPE** function creates a new matrix from data in **matrix**.

SHAPE(matrix, nrow, ncol)

The values **nrow** and **ncol** specify the number of rows and columns, respectively, in the new matrix. The **SHAPE** function produces the new matrix by traversing the argument matrix in row-major order until it reaches the specified number of elements. If necessary, the **SHAPE** function reuses elements.

For example, **SHAPE({1 2 3, 4 5 6}, 3, 2)** returns {1 2, 3 4, 5 6}; **SHAPE({1 2 3, 4 5 6}, 5, 2)** returns {1 2, 3 4, 5 6, 1 2, 3 4}; and **SHAPE({1 2 3, 4 5 6}, 1, 4)** returns {1 2 3 4}.

Label Switching Problem

Label switching is an important and severe problem for all kinds of HMMs. Nonidentifiability is caused by invariance to relabeling the states of the hidden Markov chain (or the components of the mixture distribution in the case of GH HMMs); for further discussion and illustration, see Frühwirth-Schnatter (2006, page 15 and 173) and the references therein.

Any two HMMs are mathematically equivalent if the only difference between them is that the states or components are labeled in different ways. For example, consider the following models, which are both univariate bistate Gaussian HMMs described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, and μ_1 and Σ_1 are mean and covariance parameters for state 1, and μ_2 and Σ_2 are mean and covariance parameters for state 2:

$$\text{model 1: } \pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{pmatrix}, \mu_1 = 1, \Sigma_1 = 1, \mu_2 = -1, \Sigma_2 = 4$$

$$\text{model 2: } \pi = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{pmatrix}, \mu_1 = -1, \Sigma_1 = 4, \mu_2 = 1, \Sigma_2 = 1$$

In theory, these two models are equivalent, and just the state labels “1” and “2” are switched. When the labels are switched, it is inconvenient to compare any state-related results, such as filtered or smoothed probabilities or decoded path. Because of the randomness in the optimization, you might even get “different” estimation results every time because of label switching.

You can use the LABELSWITCH= option in the PROC HMM statement to solve the label switching problem. When the INMODEL= option in the SCORE statement is not specified, the LABELSWITCH=(SORT=ASC(SIGMA)) option is applied by default; that is, the states are labeled in the ascending order of the covariance parameter estimates. When the INMODEL= option in the SCORE statement is specified, the LABELSWITCH=(SORT=NONE) option is applied by default; that is, there is no relabeling. You can specify additional suboptions in the LABELSWITCH= option to label the states (and components) in whichever way you prefer.

An Example to Illustrate the LABELSWITCH= Option

The following data are simulated for this example. The data generating process (DGP) follows a bistate tricomponent GM HMM.

```
%let a11 = 0.98;
%let a22 = 0.98;
%let c1_1 = 0.3;
%let c1_2 = 0.4;
%let c1_3 = 0.3;
%let c2_1 = 0.3;
%let c2_2 = 0.4;
%let c2_3 = 0.3;
%let mu1_1_1 = -1;
%let mu1_2_1 = -6;
%let mu1_3_1 = -11;
%let sigma1_1_11 = 1;
%let sigma1_2_11 = 2;
%let sigma1_3_11 = 3;
%let mu2_1_1 = 6;
%let mu2_2_1 = 12;
%let mu2_3_1 = 20;
%let sigma2_1_11 = 4;
%let sigma2_2_11 = 5;
%let sigma2_3_11 = 6;
%let seed = 1234;
%let T = 400000;
%let nSections = 200;

data labelSwitchDGP;
  do t = 1 to &T.;
    sec = ceil(t*&nSections./&T.);
    if(t=1) then do;
      * initial probability distribution;
      p = (1-&a22.)/((1-&a11.)+(1-&a22.));
    end;
    else do;
      * transition probability matrix;
      if(lags=1) then p = &a11.;
      else p = 1-&a22.;
    end;
  end;
```

```

end;
u = uniform(&seed.);
if(u<=p) then s=1;
else s = 2;
e1 = normal(&seed.);
if(s=1) then do;
  * choose component;
  u = uniform(&seed.);
  if(u<=&c1_1.) then do;
    * x ~ N(mu, Sigma) at state 1, component 1;
    x = &mu1_1_1. + sqrt(&sigma1_1_11.)*e1;
  end;
  else do;
    if(u<=&c1_1.+&c1_2.) then do;
      * x ~ N(mu, Sigma) at state 1, component 2;
      x = &mu1_2_1. + sqrt(&sigma1_2_11.)*e1;
    end;
    else do;
      * x ~ N(mu, Sigma) at state 1, component 3;
      x = &mu1_3_1. + sqrt(&sigma1_3_11.)*e1;
    end;
  end;
end;
else do;
  * choose component;
  u = uniform(&seed.);
  if(u<=&c2_1.) then do;
    * x ~ N(mu, Sigma) at state 2, component 1;
    x = &mu2_1_1. + sqrt(&sigma2_1_11.)*e1;
  end;
  else do;
    if(u<=&c1_1.+&c1_2.) then do;
      * x ~ N(mu, Sigma) at state 2, component 2;
      x = &mu2_2_1. + sqrt(&sigma2_2_11.)*e1;
    end;
    else do;
      * x ~ N(mu, Sigma) at state 2, component 3;
      x = &mu2_3_1. + sqrt(&sigma2_3_11.)*e1;
    end;
  end;
end;
output;
lags = s;
end;
run;

```

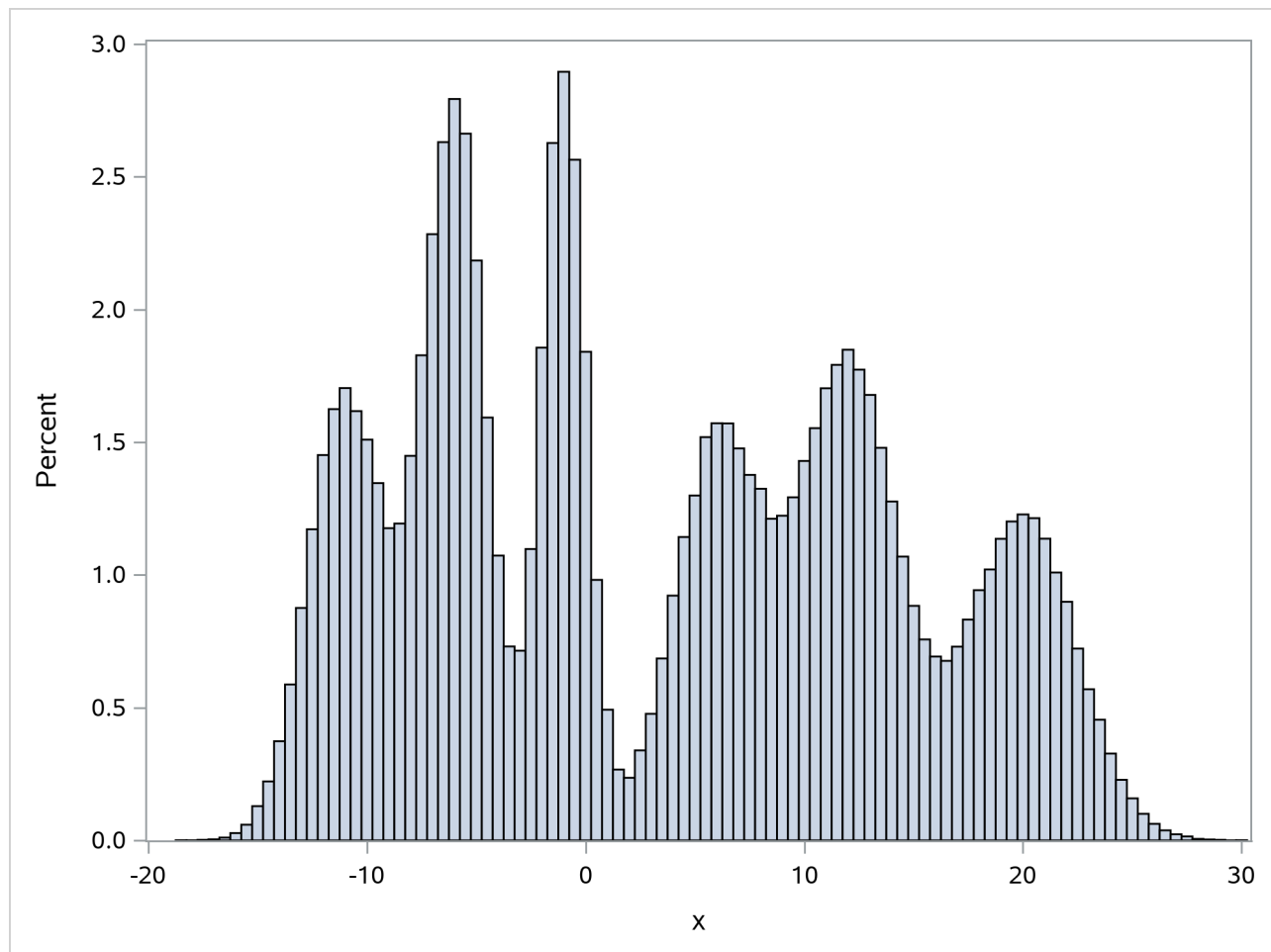
The following statements plot the histogram of the simulated x by ignoring the sectional and sequential information.

```

proc sgplot data=labelSwitchDGP;
  histogram x;
run;

```

Figure 20.90 shows the histogram of x , in which there are six modes around 20, 12, 6, -1 , -6 , and -11 , as expected.

Figure 20.90 Plot of Histogram of x 

The following statements make a copy of the data in the library:

```
data mylib.labelSwitchDGP; set labelSwitchDGP; run;
```

The following statements estimate the six-state Gaussian HMM by using the initial values {20, 12, 6, -1, -6, -11} for the mean parameter:

```
proc hmm data=mylib.labelSwitchDGP;
  id time=t section=sec;
  model x / type=gaussian nstate=6;
  optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
  initial mu={20, 12, 6, -1, -6, -11};
run;
```

The final parameter estimates after optimization, which are shown in Figure 20.91, are totally different from the reported parameter estimates in Figure 20.92. In Figure 20.91, the six states, described as Gaussian distribution, are in order $N(20, 6)$, $N(12, 5)$, $N(6, 4)$, $N(-1, 1)$, $N(-6, 2)$, and $N(-11, 3)$; however, in Figure 20.92, the six states are in order $N(-1, 1)$, $N(-6, 2)$, $N(-11, 3)$, $N(6, 4)$, $N(12, 5)$, and $N(20, 6)$. The difference is caused by the states being relabeled, by default, according to the estimates of the covariance matrix parameter of each state in ascending order; that is, the LABELSWITCH=(SORT=ASC(SIGMA)) option is applied by default.

Figure 20.91 Final Parameter Estimates after Optimization

Optimization Results	
Parameter	Estimate
TPM1_1	0.292461
TPM1_2	0.392902
TPM1_3	0.294507
TPM1_4	0.005954
TPM1_5	0.007939
TPM1_6	0.006237
TPM2_1	0.294241
TPM2_2	0.387707
TPM2_3	0.298507
TPM2_4	0.005716
TPM2_5	0.007708
TPM2_6	0.006121
TPM3_1	0.295218
TPM3_2	0.387390
TPM3_3	0.297737
TPM3_4	0.006181
TPM3_5	0.007747
TPM3_6	0.005726
TPM4_1	0.006300
TPM4_2	0.008122
TPM4_3	0.006358
TPM4_4	0.292704
TPM4_5	0.390938
TPM4_6	0.295579
TPM5_1	0.005966
TPM5_2	0.007509
TPM5_3	0.006372
TPM5_4	0.294351
TPM5_5	0.390263
TPM5_6	0.295538
TPM6_1	0.005998
TPM6_2	0.008252
TPM6_3	0.005837
TPM6_4	0.292779
TPM6_5	0.392673
TPM6_6	0.294461
MU1_1	19.994009
MU2_1	12.015924
MU3_1	6.036520
MU4_1	-1.002846
MU5_1	-5.994957
MU6_1	-10.977249
SIGMA1_1_1	6.082481
SIGMA2_1_1	4.881128
SIGMA3_1_1	4.037455
SIGMA4_1_1	1.007495
SIGMA5_1_1	1.987658
SIGMA6_1_1	3.042159

Figure 20.92 Parameter Estimates

Parameter	Parameter Estimates			
	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.292704	0.001985	147.45	<.0001
TPM1_2	0.390938	0.002676	146.08	<.0001
TPM1_3	0.295579	0.002419	122.18	<.0001
TPM1_4	0.006358	0.000369	17.23	<.0001
TPM1_5	0.008122	0.000429	18.93	<.0001
TPM1_6	0.006300	0.000352	17.91	<.0001
TPM2_1	0.294351	0.001805	163.09	<.0001
TPM2_2	0.390263	0.002506	155.74	<.0001
TPM2_3	0.295538	0.002251	131.31	<.0001
TPM2_4	0.006372	0.000331	19.24	<.0001
TPM2_5	0.007509	0.000368	20.40	<.0001
TPM2_6	0.005966	0.000304	19.62	<.0001
TPM3_1	0.292779	0.002057	142.35	<.0001
TPM3_2	0.392673	0.002761	142.23	<.0001
TPM3_3	0.294461	0.002483	118.61	<.0001
TPM3_4	0.005837	0.000367	15.91	<.0001
TPM3_5	0.008252	0.000435	18.99	<.0001
TPM3_6	0.005998	0.000350	17.14	<.0001
TPM4_1	0.006181	0.000354	17.46	<.0001
TPM4_2	0.007747	0.000411	18.83	<.0001
TPM4_3	0.005726	0.000356	16.10	<.0001
TPM4_4	0.297737	0.003360	88.62	<.0001
TPM4_5	0.387390	0.004081	94.94	<.0001
TPM4_6	0.295218	0.002374	124.34	<.0001
TPM5_1	0.005716	0.000305	18.73	<.0001
TPM5_2	0.007708	0.000360	21.42	<.0001
TPM5_3	0.006121	0.000320	19.13	<.0001
TPM5_4	0.298507	0.003211	92.97	<.0001
TPM5_5	0.387707	0.003928	98.70	<.0001
TPM5_6	0.294241	0.002163	136.02	<.0001
TPM6_1	0.005954	0.000337	17.67	<.0001
TPM6_2	0.007939	0.000400	19.87	<.0001
TPM6_3	0.006237	0.000354	17.64	<.0001
TPM6_4	0.294507	0.003291	89.49	<.0001
TPM6_5	0.392902	0.004027	97.58	<.0001
TPM6_6	0.292461	0.002318	126.17	<.0001
MU1_1	-1.002846	0.005048	-198.67	<.0001
MU2_1	-5.994957	0.008460	-708.59	<.0001
MU3_1	-10.977249	0.014627	-750.46	<.0001
MU4_1	6.036520	0.023829	253.33	<.0001
MU5_1	12.015924	0.019460	617.48	<.0001
MU6_1	19.994009	0.018676	1070.59	<.0001
SIGMA1_1_1	1.007495	0.007790	129.34	<.0001
SIGMA2_1_1	1.987658	0.023807	83.49	<.0001
SIGMA3_1_1	3.042159	0.033805	89.99	<.0001
SIGMA4_1_1	4.037455	0.053806	75.04	<.0001
SIGMA5_1_1	4.881128	0.097545	50.04	<.0001
SIGMA6_1_1	6.082481	0.063053	96.47	<.0001

The following statements explicitly show how the states are relabeled by using the LABELSWITCH=(OUT=) option:

```
proc hmm data=mylib.labelSwitchDGP labelSwitch=(out=mylib.olabel1);
  id time=t section=sec;
  model x / type=gaussian nstate=6;
  optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
  initial mu={20, 12, 6, -1, -6, -11};
run;

data olabel1; set mylib.olabel1; run;
proc sort data=olabel1; by oldStateLabel; run;
proc print data=olabel1 noobs label; run;
```

The first row in [Figure 20.93](#) shows that the old state label “1” is relabeled as the new state “6”; that is, $N(20, 6)$ was labeled as state 1 but is relabeled as state 6. The same logic applies to other rows. The second row in [Figure 20.93](#) shows that the old state label “2” is relabeled as the new state “5”; the third row shows the old state label “3” is relabeled as the new state “4”; and so on. Note that all parameter estimates (ISPV, TPM, and so on) are permuted accordingly after the states are relabeled.

Figure 20.93 Old and New Labels of States

Old State Label	New State Label
1	6
2	5
3	4
4	1
5	2
6	3

The following statements request that the states be labeled in ascending order of the mean parameter estimates. As shown in [Figure 20.94](#), all parameters are permuted according to the new state labels.

```
proc hmm data=mylib.labelSwitchDGP labelSwitch=(sort=asc(mu) out=mylib.olabel2);
  id time=t section=sec;
  model x / type=gaussian nstate=6;
  optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
  initial mu={20, 12, 6, -1, -6, -11};
run;
```

Figure 20.94 Parameter Estimates

Parameter	Parameter Estimates			
	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.294461	0.002483	118.61	<.0001
TPM1_2	0.392673	0.002761	142.23	<.0001
TPM1_3	0.292779	0.002057	142.35	<.0001
TPM1_4	0.005837	0.000367	15.91	<.0001
TPM1_5	0.008252	0.000435	18.99	<.0001
TPM1_6	0.005998	0.000350	17.14	<.0001
TPM2_1	0.295538	0.002251	131.31	<.0001
TPM2_2	0.390263	0.002506	155.74	<.0001
TPM2_3	0.294351	0.001805	163.09	<.0001
TPM2_4	0.006372	0.000331	19.24	<.0001
TPM2_5	0.007509	0.000368	20.40	<.0001
TPM2_6	0.005966	0.000304	19.62	<.0001
TPM3_1	0.295579	0.002419	122.18	<.0001
TPM3_2	0.390938	0.002676	146.08	<.0001
TPM3_3	0.292704	0.001985	147.45	<.0001
TPM3_4	0.006358	0.000369	17.23	<.0001
TPM3_5	0.008122	0.000429	18.93	<.0001
TPM3_6	0.006300	0.000352	17.91	<.0001
TPM4_1	0.005726	0.000356	16.10	<.0001
TPM4_2	0.007747	0.000411	18.83	<.0001
TPM4_3	0.006181	0.000354	17.46	<.0001
TPM4_4	0.297737	0.003360	88.62	<.0001
TPM4_5	0.387390	0.004081	94.94	<.0001
TPM4_6	0.295218	0.002374	124.34	<.0001
TPM5_1	0.006121	0.000320	19.13	<.0001
TPM5_2	0.007708	0.000360	21.42	<.0001
TPM5_3	0.005716	0.000305	18.73	<.0001
TPM5_4	0.298507	0.003211	92.97	<.0001
TPM5_5	0.387707	0.003928	98.70	<.0001
TPM5_6	0.294241	0.002163	136.02	<.0001
TPM6_1	0.006237	0.000354	17.64	<.0001
TPM6_2	0.007939	0.000400	19.87	<.0001
TPM6_3	0.005954	0.000337	17.67	<.0001
TPM6_4	0.294507	0.003291	89.49	<.0001
TPM6_5	0.392902	0.004027	97.58	<.0001
TPM6_6	0.292461	0.002318	126.17	<.0001
MU1_1	-10.977249	0.014627	-750.46	<.0001
MU2_1	-5.994957	0.008460	-708.59	<.0001
MU3_1	-1.002846	0.005048	-198.67	<.0001
MU4_1	6.036520	0.023829	253.33	<.0001
MU5_1	12.015924	0.019460	617.48	<.0001
MU6_1	19.994009	0.018676	1070.59	<.0001
SIGMA1_1_1	3.042159	0.033805	89.99	<.0001
SIGMA2_1_1	1.987658	0.023807	83.49	<.0001
SIGMA3_1_1	1.007495	0.007790	129.34	<.0001
SIGMA4_1_1	4.037455	0.053806	75.04	<.0001
SIGMA5_1_1	4.881128	0.097545	50.04	<.0001
SIGMA6_1_1	6.082481	0.063053	96.47	<.0001

The following statements estimate the bivariate tricomponent GM HMM. If you compare the final parameter estimates after optimization that are shown in [Figure 20.95](#) and the parameter estimates that are reported in [Figure 20.96](#), you see that the states and components are relabeled. State labels are exchanged, and the labels of component 1 and 3 for old state 1 are also exchanged, which is clearly shown in [Figure 20.97](#).

```
proc hmm data=mylib.labelSwitchDGP labelSwitch=(out=mylib.olabel3);
  id time=t section=sec;
  model x / type=gaussianMixture nstate=2 ncomponent=3;
  optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
  initial mu={20, 12, 6, -1, -6, -11};
  score outmodel=mylib.odel;
run;

data olabel3; set mylib.olabel3; run;
proc sort data=olabel3; by oldStateLabel oldComponentLabel; run;
proc print data=olabel3 noobs label; run;
```

Figure 20.95 Final Parameter Estimates after Optimization

Optimization Results	
Parameter	Estimate
TPM1_1	0.980249
TPM1_2	0.019751
TPM2_1	0.020196
TPM2_2	0.979804
MCP1_1	0.299927
MCP1_2	0.396951
MCP1_3	0.303122
MCP2_1	0.299444
MCP2_2	0.399150
MCP2_3	0.301406
MU1_1_1	19.994224
MU1_2_1	12.015837
MU1_3_1	6.036275
MU2_1_1	-1.002864
MU2_2_1	-5.994918
MU2_3_1	-10.977185
SIGMA1_1_1_1	6.081830
SIGMA1_2_1_1	4.882495
SIGMA1_3_1_1	4.037831
SIGMA2_1_1_1	1.007494
SIGMA2_2_1_1	1.987574
SIGMA2_3_1_1	3.042284

Figure 20.96 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.979804	0.000315	3107.71	<.0001
TPM1_2	0.020196	0.000315	64.06	<.0001
TPM2_1	0.019751	0.000308	64.04	<.0001
TPM2_2	0.980249	0.000308	3178.10	<.0001
MCP1_1	0.299444	0.001126	265.87	<.0001
MCP1_2	0.399150	0.001943	205.44	<.0001
MCP1_3	0.301406	0.001712	176.01	<.0001
MCP2_1	0.303122	0.002797	108.36	<.0001
MCP2_2	0.396951	0.003540	112.13	<.0001
MCP2_3	0.299927	0.001544	194.29	<.0001
MU1_1_1	-1.002864	0.005048	-198.67	<.0001
MU1_2_1	-5.994918	0.008460	-708.63	<.0001
MU1_3_1	-10.977185	0.014627	-750.48	<.0001
MU2_1_1	6.036275	0.023837	253.23	<.0001
MU2_2_1	12.015837	0.019464	617.34	<.0001
MU2_3_1	19.994224	0.018675	1070.63	<.0001
SIGMA1_1_1_1	1.007494	0.007789	129.35	<.0001
SIGMA1_2_1_1	1.987574	0.023804	83.50	<.0001
SIGMA1_3_1_1	3.042284	0.033806	89.99	<.0001
SIGMA2_1_1_1	4.037831	0.053783	75.08	<.0001
SIGMA2_2_1_1	4.882495	0.097590	50.03	<.0001
SIGMA2_3_1_1	6.081830	0.063045	96.47	<.0001

Figure 20.97 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	2	3
1	2	2	2
1	3	2	1
2	1	1	1
2	2	1	2
2	3	1	3

The following statements specify the INMODEL= option in the SCORE statement, which causes the LABELSWITCH=(SORT=NONE) option to be applied by default. It is not surprising to see that the parameter estimates in Figure 20.98 are exactly same as those in Figure 20.96; that is, labels of states and components are not reordered, as confirmed by Figure 20.99.

```
proc hmm data=mylib.labelSwitchDGP
    labelSwitch=(out=mylib.olabel4);;
    score inmodel=mylib.omodel;
run;
```

```
data olabel4; set mylib.odelabel4; run;
proc sort data=odelabel4; by oldStateLabel oldComponentLabel; run;
proc print data=odelabel4 noobs label; run;
```

Figure 20.98 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.979804	0.000315	3107.71	<.0001
TPM1_2	0.020196	0.000315	64.06	<.0001
TPM2_1	0.019751	0.000308	64.04	<.0001
TPM2_2	0.980249	0.000308	3178.10	<.0001
MCP1_1	0.299444	0.001126	265.87	<.0001
MCP1_2	0.399150	0.001943	205.44	<.0001
MCP1_3	0.301406	0.001712	176.01	<.0001
MCP2_1	0.303122	0.002797	108.36	<.0001
MCP2_2	0.396951	0.003540	112.13	<.0001
MCP2_3	0.299927	0.001544	194.29	<.0001
MU1_1_1	-1.002864	0.005048	-198.67	<.0001
MU1_2_1	-5.994918	0.008460	-708.63	<.0001
MU1_3_1	-10.977185	0.014627	-750.48	<.0001
MU2_1_1	6.036275	0.023837	253.23	<.0001
MU2_2_1	12.015837	0.019464	617.34	<.0001
MU2_3_1	19.994224	0.018675	1070.63	<.0001
SIGMA1_1_1_1	1.007494	0.007789	129.35	<.0001
SIGMA1_2_1_1	1.987574	0.023804	83.50	<.0001
SIGMA1_3_1_1	3.042284	0.033806	89.99	<.0001
SIGMA2_1_1_1	4.037831	0.053783	75.08	<.0001
SIGMA2_2_1_1	4.882495	0.097590	50.03	<.0001
SIGMA2_3_1_1	6.081830	0.063045	96.47	<.0001

Figure 20.99 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	1	1
1	2	1	2
1	3	1	3
2	1	2	1
2	2	2	2
2	3	2	3

However, you can reorder the labels of states and components in whichever way you prefer by using the LABELSWITCH=(REORDER=) and LABELSWITCH=(REORDERCOMPONENT=) options, as illustrated in the following statements:

```

proc hmm data=mylib.labelSwitchDGP
      labelSwitch=(reorder=(2 1) reorderComponent=(2 1 3 2 3 1)
                  out=mylib.olabel5);
      score inmodel=mylib.omodel;
run;

data olabel5; set mylib.olabel5; run;
proc sort data=olabel5; by oldStateLabel oldComponentLabel; run;
proc print data=olabel5 noobs label; run;

```

The parameter estimates are shown in Figure 20.100, and the label comparison is shown in Figure 20.101.

Figure 20.100 Parameter Estimates

Parameter	Parameter Estimates			
	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.980249	0.000308	3178.10	<.0001
TPM1_2	0.019751	0.000308	64.04	<.0001
TPM2_1	0.020196	0.000315	64.06	<.0001
TPM2_2	0.979804	0.000315	3107.71	<.0001
MCP1_1	0.299927	0.001544	194.29	<.0001
MCP1_2	0.303122	0.002797	108.36	<.0001
MCP1_3	0.396951	0.003540	112.13	<.0001
MCP2_1	0.399150	0.001943	205.44	<.0001
MCP2_2	0.299444	0.001126	265.87	<.0001
MCP2_3	0.301406	0.001712	176.01	<.0001
MU1_1_1	19.994224	0.018675	1070.63	<.0001
MU1_2_1	6.036275	0.023837	253.23	<.0001
MU1_3_1	12.015837	0.019464	617.34	<.0001
MU2_1_1	-5.994918	0.008460	-708.63	<.0001
MU2_2_1	-1.002864	0.005048	-198.67	<.0001
MU2_3_1	-10.977185	0.014627	-750.48	<.0001
SIGMA1_1_1_1	6.081830	0.063045	96.47	<.0001
SIGMA1_2_1_1	4.037831	0.053783	75.08	<.0001
SIGMA1_3_1_1	4.882495	0.097590	50.03	<.0001
SIGMA2_1_1_1	1.987574	0.023804	83.50	<.0001
SIGMA2_2_1_1	1.007494	0.007789	129.35	<.0001
SIGMA2_3_1_1	3.042284	0.033806	89.99	<.0001

Figure 20.101 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	2	2
1	2	2	1
1	3	2	3
2	1	1	2
2	2	1	3
2	3	1	1

The preceding statements illustrate how to use and interpret the LABELSWITCH= option. The statements use both Gaussian HMMs and GM HMMs, which differ as follows: Although it is trivial to know that a K-state Gaussian HMM can be regarded as a special case of a K-state unicomponent GM HMM, a K-state M-component GM HMM can also be regarded as a special case of a KM-state Gaussian HMM. In most cases, a KM-state Gaussian HMM has many more parameters (because of the huge transition probability matrix) than a K-state M-component GM HMM has. For the preceding statements, all information criteria, although not reported here, prefer the bivariate tricomponent GM HMM (which the true data generating process follows) to the six-state Gaussian HMM.

Analytic Store Technology

Scoring new data is a common and important task in deploying a model. You can score new data by using the SCORE statement in the HMM procedure, or you can do it by taking advantage of the analytic store technology. By using the STORE statement, you can save estimated models in an analytic store table. Then, you can download the table to a local analytic store file. Later, you can load the model and score new data on any platform that supports the analytic store technology—and you do not even need to have the HMM procedure available on those platforms.

The following example illustrates how to use the analytic store technology.

Generate your training data and upload them to the library. The following statements assume that your engine libref is named mylib, but you can substitute any appropriately defined libref.

```
%let pi1 = 0.5;
%let a11 = 0.2;
%let a22 = 0.2;
%let mu1_1 = 1;
%let mu1_2 = 1;
%let sigma1_11 = 1;
%let sigma1_21 = 0;
%let sigma1_22 = 1;
%let mu2_1 = -1;
%let mu2_2 = -1;
%let sigma2_11 = 1;
%let sigma2_21 = 0;
%let sigma2_22 = 1;

%macro dgpGHMM(table,T,nSections,seed);
  data &table;
    retain cd1_11 cd1_21 cd1_22 cd2_11 cd2_21 cd2_22;
    secLen = ceil(&T./&nSections.);
    do t = 1 to &T.;
      sec = floor((t-1)/secLen)+1;
      if(t=1) then do;
        /* initial probability distribution */
        p = &pi1.;
        /* Cholesky decomposition of sigma1 */
        cd1_11 = sqrt(&sigma1_11.);
        cd1_21 = &sigma1_21./sqrt(&sigma1_11.);
        cd1_22 = sqrt(&sigma1_22.-&sigma1_21.*&sigma1_21./&sigma1_11.);
        /* Cholesky decomposition of sigma2 */
```

```

        cd2_11 = sqrt(&sigma2_11.);
        cd2_21 = &sigma2_21./sqrt(&sigma2_11.);
        cd2_22 = sqrt(&sigma2_22.-&sigma2_21.*&sigma2_21./&sigma2_11.);
    end;
    else do;
        /* transition probability matrix */
        if(lags=1) then p = &a11.;
        else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e1 = normal(&seed.); e2 = normal(&seed.);
    if(s=1) then do;
        /* (x,y) ~ N(mu1, Sigma1) at state 1 */
        x = &mu1_1. + cd1_11*e1;
        y = &mu1_2. + cd1_21*e1+cd1_22*e2;
    end;
    else do;
        /* (x,y) ~ N(mu2, Sigma2) at state 2 */
        x = &mu2_1. + cd2_11*e1;
        y = &mu2_2. + cd2_21*e1+cd2_22*e2;
    end;
    output;
    lags = s;
end;
run;
proc sort data=&table.; by sec t; run;
%mend;

* prepare the training data;
%let T = 10000;
%let nSections = 10;
%let seed = 1234;
%dgpGHMM(gHmTrain,&T.,&nSections.,&seed.);
data mylib.gHmTrain; set gHmTrain; run;

```

First, you estimate the model and store it in an analytic store table by using the STORE statement as follows: You must perform this step in the HMM procedure within a CAS session.

```

* estimate the model and save it in an analytic store table;
proc hmm data=mylib.gHmTrain labelSwitch=(sort=desc(mu));
    id time=t section=sec;
    model x y / type=gaussian nstate=2;
    store out=mylib.ghmmModel;
run;

```

Then, you produce a local analytic store file as follows. You must perform this step in a CAS session that has access to the mylib.ghmmModel data table.

```

* produce a local analytic store file;
proc astore;
    download rstore=mylib.ghmmModel store="ls_ghmm";
run;

```


Now, you can use the local analytic store file to score new data on the platform of interest. For example, you can score new data as in the following code. In the SETOPTION statements, you set the DECODE option to 1 and the DECODE_WINDOW option to 8 to request the decoding results for an 8-unit time window; you set the EVALUATE option to 1 to request the evaluation up to current observation; that is, the log likelihood; you set the FILTER option to 1 to request the filtered probabilities up to current observation; you set the FORECAST option to 1, the FORECAST_ALPHA to 0.8, the FORECAST_COV to 1, and the FORECAST_LEAD to 4 to request the 4-step-ahead forecasts, their covariance matrices, and their confidence intervals at the significance level 0.80; you set the SMOOTH option to 1 and the SMOOTH_WINDOW option to 12 to request the smoothing results for a 12-unit time window. You use the DESCRIBE statement to output the basic model information and the information about input and output variables. In the SCORE statement, you specify the data to be scored in the DATA= option, the local analytic store file in the STORE= option, and the output table for the scored result in the OUT= option. For more information about the ASTORE procedure, see the chapter “The ASTORE Procedure” in *SAS Viya: Machine Learning Procedures*.

```
* get new data;
%let T = 40;
%let nSections = 2;
%let seed = 12345;
%dgpGHMM(gHmmTest, &T., &nSections., &seed.);

* score new data;
proc astore;
  setoption decode 1;
  setoption decode_window 8;
  setoption evaluate 1;
  setoption filter 1;
  setoption forecast 1;
  setoption forecast_alpha 0.8;
  setoption forecast_cov 1;
  setoption forecast_lead 4;
  setoption smooth 1;
  setoption smooth_window 12;
  describe store="ls_ghmm";
  score data=gHmmTest store="ls_ghmm" out=ghmmOut;
run;
```

If you need to score new data within a session, you can upload the analytic store file first, and then score the new data by using the RSTORE= option in the ASTORE procedure, as in the following code. Of course, if your session has access to the original analytic store table mylib.ghmmModel, you can skip the uploading step and directly use the option RSTORE=mylib.ghmmModel.

```
* get new data;
%let T = 40;
%let nSections = 2;
%let seed = 12345;
%dgpGHMM(gHmmTest, &T., &nSections., &seed.);
data mylib.gHmmTest; set gHmmTest; run;

* upload the local analytic store;
proc astore;
  upload store="ls_ghmm" rstore=mylib.rs_ghmm;
run;
```

```

* score new data;
proc astore;
    setoption decode 1;
    setoption decode_window 8;
    setoption evaluate 1;
    setoption filter 1;
    setoption forecast 1;
    setoption forecast_alpha 0.8;
    setoption forecast_cov 1;
    setoption forecast_lead 4;
    setoption smooth 1;
    setoption smooth_window 12;
    describe rstore=mylib.rs_ghmm;
    score data=mylib.gHmTest rstore=mylib.rs_ghmm out=mylib.ghmmCasOut;
run;

```

To score new data, you can use the SCORE statement in the HMM procedure, or you can use the STORE statement in the HMM procedure and then use the ASTORE procedure or other products that support the analytic store technology. Sometimes the two methods do not generate the same inference results for the same data set, because these methods treat the data in different ways. That is, using the SCORE statement in the HMM procedure reads all observations in the data set and then scores them; this is the so-called batch mode. In contrast, the analytic store technology deals with observations one by one, which is extremely suitable for streaming analytics; this is the so-called online mode. The following list compares the different kinds of results that these two methods produce:

- For evaluation and filtered results, the two methods are the same.
- For smoothed and decoded results, the two methods are different. Let T be the sample size. For smoothing, in batch mode, $p(S_\tau = i | y_1, \dots, y_T)$, $1 \leq \tau \leq T$, can be calculated; but in online mode, at time t , only $y_1, \dots, y_t$ are available, and then $p(S_\tau = i | y_1, \dots, y_t)$ can be calculated only if $\tau \leq t$. Because of the memory decay of the Markovian structure, $p(S_\tau = i | y_1, \dots, y_t)$ can approximate $p(S_\tau = i | y_1, \dots, y_T)$ very well when $t - \tau$ is large enough. When you set the value of the SMOOTH_WINDOW option to w in the SETOPTION statement in PROC ASTORE, the smoothed probabilities $p(S_\tau = i | y_1, \dots, y_t)$, $\tau = t - w, \dots, t$, $i = 1, \dots, K$, are calculated and output, where K is the number of states. In some literature, the window length is also called the detection delay, because $p(S_{t-w} = i | y_1, \dots, y_t)$ is assumed to be close enough to $p(S_{t-w} = i | y_1, \dots, y_T)$; that is, additional data after time t would not increase the accuracy of the estimation of the smoothed probabilities at time $t - w$. The same logic can be applied to the decoded results.
- For forecasted results, the two methods are the same if there are no exogenous variables in the models; otherwise, the methods are different only if the values of the exogenous variables at time $t + h$, $h > 0$, are needed in some forecast at time t . Let x_t denote the exogenous variables at time t . In batch mode, the h -step-ahead forecast $p(y_{t+h} | y_1, \dots, y_t; x_1, \dots, x_{t+h})$, $h > 0$, can be calculated when $t + h \leq T$ or extra exogenous variable values after T are provided. However, in online mode, x_{t+h} , $h > 0$, is not available at time t , and then some forecasts that need x_{t+h} are not calculable. In practice, in order to use the analytic store technology in online mode, you must make sure that the h -step-ahead forecast of interest is calculable. For example, by using the NOCURRENTX option in the MODEL statement in PROC HMM, and later by setting the FORECAST_LEAD option to 1 in the SETOPTION statement in PROC ASTORE, you can get the one-step-ahead forecast of dependent variables.

Information Criteria

When you compare models, choose the model that has the smallest information criterion values. The information criteria include Akaike's information criterion (AIC), the corrected Akaike's information criterion (AICC), the Bayesian information criterion (BIC, also referred to as the Schwarz Bayesian criterion, SBC), and the Hannan-Quinn criterion (HQC). These criteria are defined as

$$\begin{aligned} \text{AIC} &= -2\ell + 2r \\ \text{AICC} &= -2\ell + 2rT/(T - r - 1) \\ \text{BIC} &= -2\ell + r \log(T) \\ \text{HQC} &= -2\ell + 2r \log(\log(T)) \end{aligned}$$

where ℓ is the log likelihood, r is the total number of parameters in the model, and T is the number of observations that are used to estimate the model.

Missing Values

Exogenous variables cannot have missing values in all types of models. Missing values of dependent variables are treated as follows:

- For Gaussian HMMs and Gaussian mixture HMMs, all or some of the dependent variables in an observation can be missing. The estimation and inference are based on all observations. For example, if the maximum likelihood method is applied, the likelihood of the nonmissing observations and the partially nonmissing observations is calculated and maximized.
- For the regime-switching regression models, all or some of the dependent variables in an observation can be missing. The estimation and inference are based on the observations from the beginning to the last nonmissing or partially nonmissing observation. For example, if the maximum likelihood method is applied, the likelihood of the nonmissing observations and the partially nonmissing observations is calculated and maximized. For the purpose of forecasting, at the end of the data table (or the end of each section if you specify the SECTION= option), the dependent variables can be set as missing values while the exogenous variables are provided.
- For the regime-switching autoregression models, missing values in the dependent variables are not supported. But at the end of the data table (or the end of each section if you specify the SECTION= option), for the purpose of forecasting, the dependent variables can be set as missing values while the exogenous variables are provided.

Data Table Output

The HMM procedure can create the data tables that are specified in the OUT= suboption of the LABELSWITCH= option in the PROC HMM statement; the OUTSTAT= option in the PROC HMM statement; the OUT= options in the DECODE, EVALUATE, FILTER, and SMOOTH statements; the OUT= and OUTALL= options in the ESTIMATE and FORECAST statements; and the OUTMODEL= option in the SCORE statement. The column information for each table is described in the following sections.

OUT= Data Table Generated from the LABELSWITCH= Option in the PROC HMM Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= n_1 : n_2 option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= n_1 : n_2 option
OldStateLabel	old label of the state
OldComponentLabel	old label of the component for a GM HMM
NewStateLabel	new label of the state
NewComponentLabel	new label of the component for a GM HMM

OUTSTAT= Data Table Generated from the PROC HMM Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= n_1 : n_2 option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= n_1 : n_2 option
LogLikelihood	log likelihood
AIC	Akaike's information criterion
AICC	corrected Akaike's information criterion
BIC	Bayesian information criterion (also referred to as the Schwarz Bayesian criterion, SBC)
HQC	Hannan-Quinn criterion

OUT= Data Table Generated from the DECODE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State	state value in the most possible path

OUT= Data Table Generated from the ESTIMATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Index	index of the parameter
Type	type of values in the State k variables: EST for estimate and STD for standard error
Parameter	parameter name
State k	value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option

OUTALL= Data Table Generated from the ESTIMATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Index	index of the parameter
Parameter	parameter name
Estimate	parameter estimate
StdErr	standard error

COV_	n	value for column n , $n = 1, \dots, N$, in the covariance matrix of parameter estimates, where N is the number of parameters if only one model is estimated or scored; otherwise, N is the maximum of numbers of parameters for all models
------	-----	---

OUT= Data Table Generated from the EVALUATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
LogLikelihood	log likelihood

OUT= Data Table Generated from the FILTER Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State k	filtered probability value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option

OUT= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement

Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
Step	value of h in the h -step-ahead forecast
State k	forecasted probability value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
{DV p }_Forecast	mean forecast for dependent variable p , $p = 1, \dots, k_y$, where k_y is the number of dependent variables; {DV p } stands for the name of dependent variable p
Q1	first quantile value, $(1 - \alpha)/2$, where α is specified in the ALPHA= α option in the FORECAST statement
{DV p }_Q1	first quantile forecast for dependent variable p , $p = 1, \dots, k_y$, where k_y is the number of dependent variables; {DV p } stands for the name of dependent variable p
Q2	second quantile value, $(1 + \alpha)/2$, where α is specified in the ALPHA= α option in the FORECAST statement
{DV p }_Q2	second quantile forecast for dependent variable p , $p = 1, \dots, k_y$, where k_y is the number of dependent variables; {DV p } stands for the name of dependent variable p
Q3	third quantile value, $1/2$, for median
{DV p }_Q3	median forecast for dependent variable p , $p = 1, \dots, k_y$, where k_y is the number of dependent variables; {DV p } stands for the name of dependent variable p

OUTALL= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
Step	value of h in the h -step-ahead forecast
Index	index of the dependent variable
Variable	dependent variable name
State k	forecasted probability value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
Forecast	mean forecast for the dependent variable
StdErr	standard error of forecast for the dependent variable
LowerCL	lower confidence limit of forecast for the dependent variable

UpperCL	upper confidence limit of forecast for the dependent variable
Median	median forecast for the dependent variable
COV_{DV p }	value for column p , $p = 1, \dots, k_y$, in the covariance matrix of forecasts of dependent variables, where k_y is the number of dependent variables; {DV p } stands for the name of the dependent variable p

OUTMODEL= Data Table Generated from the SCORE Statement

The output data table consists of binary large object columns to store the binary data, which should be consumed only by the INMODEL= option in the SCORE statement. The number of columns and the definition of columns might be changed without notice in future releases. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters because the data type of each column is binary.

OUT= Data Table Generated from the SMOOTH Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State k	smoothed probability value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option

ODS Table Names

The HMM procedure assigns a name to each table that it creates. You can use this name to refer to the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in Table 20.6.

Table 20.6 ODS Tables Produced in the HMM Procedure

ODS Table Name	Description	Option
Algorithm	Option values for the optimization algorithm	PRINTLEVEL= n , where $n = 2$ or 3
ARCoef	Coefficients of autoregressive variables for a regime-switching autoregression model	Default

Table 20.6 *continued*

ODS Table Name	Description	Option
ConstCoef	Coefficients of constants for a regime-switching regression model or regime-switching autoregression model	Default
Cov	Covariance matrices for a regime-switching regression model or regime-switching autoregression model	Default
CPM	Category probability matrix for a finite HMM	Default
ExogCoef	Coefficients of exogenous variables for a regime-switching regression model or regime-switching autoregression model	Default
FinalObjectiveFunction	Objective function value at the end of the optimization	PRINTLEVEL=3
FinalParameterEstimates	Final parameter values at the end of the optimization	PRINTLEVEL=3
FitStatistics	Log likelihood and information criteria	Default
InitialObjectiveFunction	Objective function value at the start of the optimization	PRINTLEVEL= n , where $n = 1, 2$, or 3
InitialParameterEstimates	Initial parameter values at the start of the optimization	PRINTLEVEL= n , where $n = 1, 2$, or 3
ISPV	Initial state probability vector	Default
IterHistory	Optimization iterations for the active set and interior point algorithms	PRINTITERFREQ= n , $n > 0$
Lambda	Mean and variance vectors, λ , for a Poisson HMM	Default
LinearCoef	Coefficients of linear trends for a regime-switching regression model or regime-switching autoregression model	Default
MCP	Mixture component probabilities for a GM HMM	Default
ModelInfo	Model information	Default
Mu	Mean vectors, μ , for a Gaussian or GM HMM	Default
NObs	Observation information	Default
OptIterHistory	Optimization iterations for the SGD algorithm	PRINTITERFREQ= n , $n > 0$
ParameterEstimates	Model parameter estimates	Default
PriorHyperparm	Prior hyperparameters	METHOD=MAP

Table 20.6 *continued*

ODS Table Name	Description	Option
QuadCoef	Coefficients of quadratic trends for a regime-switching regression model or regime-switching autoregression model	Default
SeasCoef	Coefficients of seasonal dummies for a regime-switching regression model or regime-switching autoregression model	Default
Sigma	Covariance matrices, σ , for a Gaussian or GM HMM	Default
TPM	Transition probability matrix	Default

Several Types of Models as Special Cases of Hidden Markov Models

Several types of models can be treated as special cases of hidden Markov models, including Gaussian mixture models (GMMs), Gaussian mixture regressions (GMRs), regressions, autoregressive error models, vector autoregressions (VARs) with or without exogenous variables in the standard or mean-adjusted form, and so on. In this section, a few of these models are illustrated.

Gaussian Mixture Model and Gaussian Mixture Regressions

Let $\mathbf{Y}_i = (y_{1i}, \dots, y_{pi})'$, $i = 1, \dots, n$, denote a p -dimensional vector of random variables, and $\mathbf{Y}_i$ follows an iid Gaussian mixture distribution,

$$\mathbf{Y}_i \sim \text{GM}(\{w_1, \dots, w_M\}, \{\mu_1, \dots, \mu_M\}, \{\Sigma_1, \dots, \Sigma_M\})$$

where $\text{GM}(\cdot, \cdot, \cdot)$ represents the Gaussian mixture distribution; M is the number of components; and the w_j , $j = 1, \dots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_j \geq 0$, $j = 1, \dots, M$, and $\sum_{j=1}^M w_j = 1$). The μ_j and Σ_j , $j = 1, \dots, M$, are the mean and covariance parameters, respectively, for the j th Gaussian component. The preceding model is the Gaussian mixture model (GMM). For more information about GMM, see Frühwirth-Schnatter (2006). For more information about Gaussian mixture regression (GMR) and its relationship to GMM, see Sung (2004).

Consider a bivariate Gaussian mixture model that has the following parameter values:

$$w = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mu_1 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_2 = \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The following statements simulate the bivariate vectors from the previous model to provide training data. In fact, for GMMs, each observation is independent from the others. In order to use the HMM procedure to estimate the GMMs, you must assign each observation in the training data set a unique section ID and a constant time ID; that is, each section contains only one observation.

```

data one;
  call streaminit(12345);
  do sec = 1 to 10000;
    t = 1;
    u = rand('UNIFORM');
    if(u<=0.75) then do;
      s = 1;
      x = 2 + rand('NORMAL');
      y = 2 + rand('NORMAL');
    end;
    else do;
      s = 2;
      x = -2 + rand('NORMAL');
      y = -2 + rand('NORMAL');
    end;
    output;
  end;
run;

```

```
data mylib.one; set one; run;
```

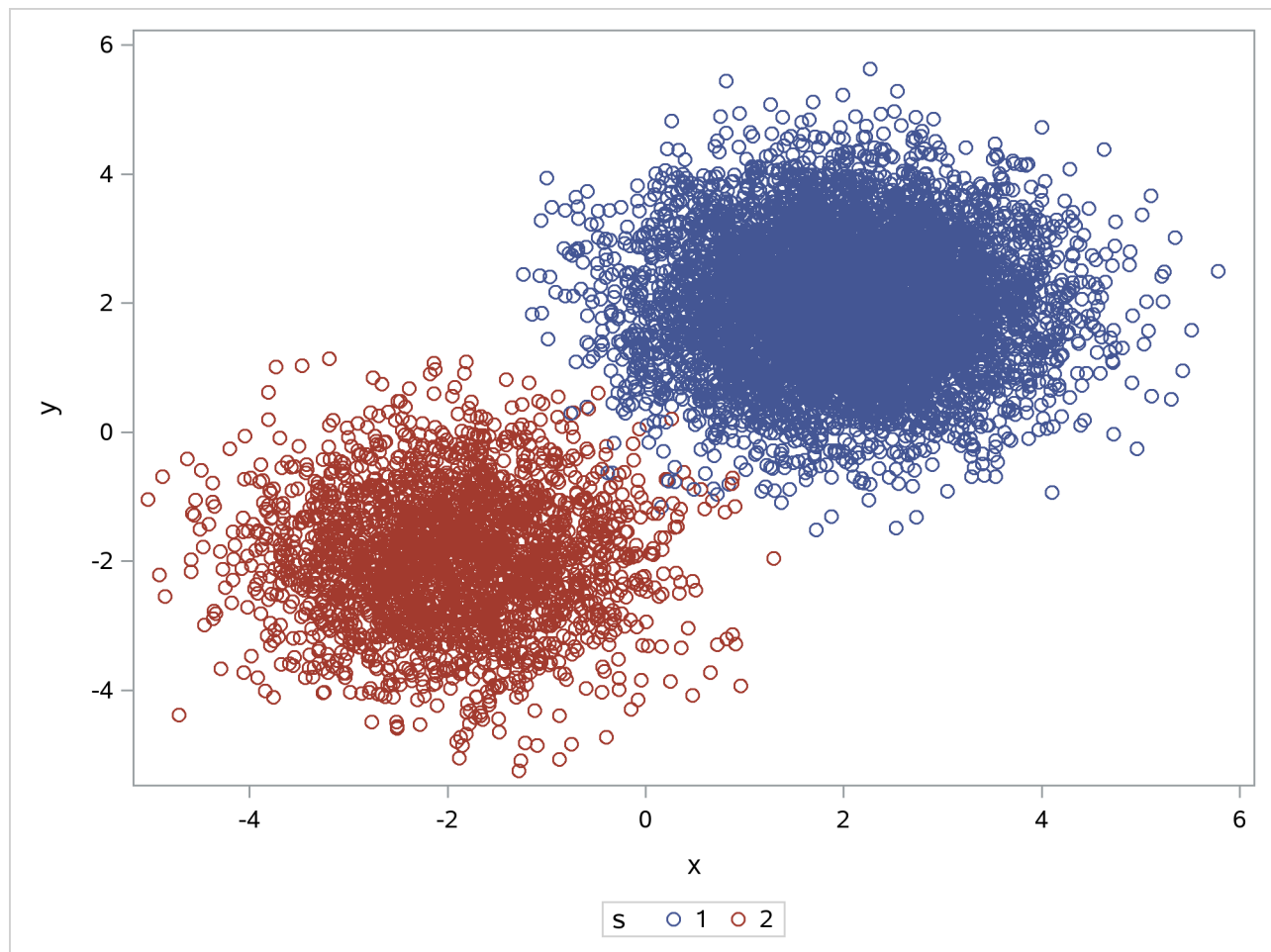
The following statements plot the observed points (x, y) :

```

proc sgplot data=one;
  scatter x=x y=y / group=s;
run;

```

As shown in [Figure 20.102](#), it seems that the points come from two different clusters.

Figure 20.102 Unclassified Observations

The following statements estimate the two-component Gaussian mixture model and then classify the data points through decoding; here decoding means finding the most likely cluster for the observed data. The STORE statement is used to save the model in the analytic store table for future scoring.

```
proc hmm data=mylib.one labelSwitch=(sort=desc(mu));
  id time=t section=sec;
  model x y / nstate=2 type=gaussian estispv;
  decode out=mylib.mydecode;
  store mylib.mygmm;
run;
```

The estimates of all parameters are shown in [Figure 20.103](#), which displays columns for the parameter name, estimate value, standard error, t value, and p -value. The TPM parameters are meaningless in this context. The ISPV parameters are the mixture component probabilities. The parameter estimates are very close to the true parameter values in the DGP.

Figure 20.103 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
ISPV1	0.741154	0.004402	168.35	<.0001
ISPV2	0.258846	0.004402	58.80	<.0001
TPM1_1	0.500000	0		
TPM1_2	0.500000	0		
TPM2_1	0.500000	0		
TPM2_2	0.500000	0		
MU1_1	2.017677	0.011520	175.14	<.0001
MU1_2	1.998486	0.011714	170.61	<.0001
MU2_1	-1.972142	0.019170	-102.88	<.0001
MU2_2	-1.971479	0.020118	-98.00	<.0001
SIGMA1_1_1	0.971840	0.016189	60.03	<.0001
SIGMA1_2_1	-0.003386	0.011752	-0.29	0.7732
SIGMA1_2_2	1.002999	0.016772	59.80	<.0001
SIGMA2_1_1	0.918582	0.026555	34.59	<.0001
SIGMA2_2_1	-0.009950	0.019897	-0.50	0.6170
SIGMA2_2_2	1.020699	0.029038	35.15	<.0001

The accuracy of classification through the GMM is calculated by the following statements:

```
data mydecode; set mylib.mydecode; run;
proc sort data=mydecode; by sec; run;

data combine;
  merge mydecode(in=a) one(in=b);
  by sec;
run;

data checkAccuracy;
  set combine;
  retain accuracy 0;
  if(state=s) then accuracy+1;
  if(_N_=10000) then do;
    accuracy = accuracy/_N_;
    output;
  end;
  keep accuracy;
run;

proc print data=checkAccuracy noobs; run;
```

The accuracy of the GMM, which is shown in [Figure 20.104](#), is very high. It implies that you can use the HMM procedure not only to learn but also to infer the GMM.

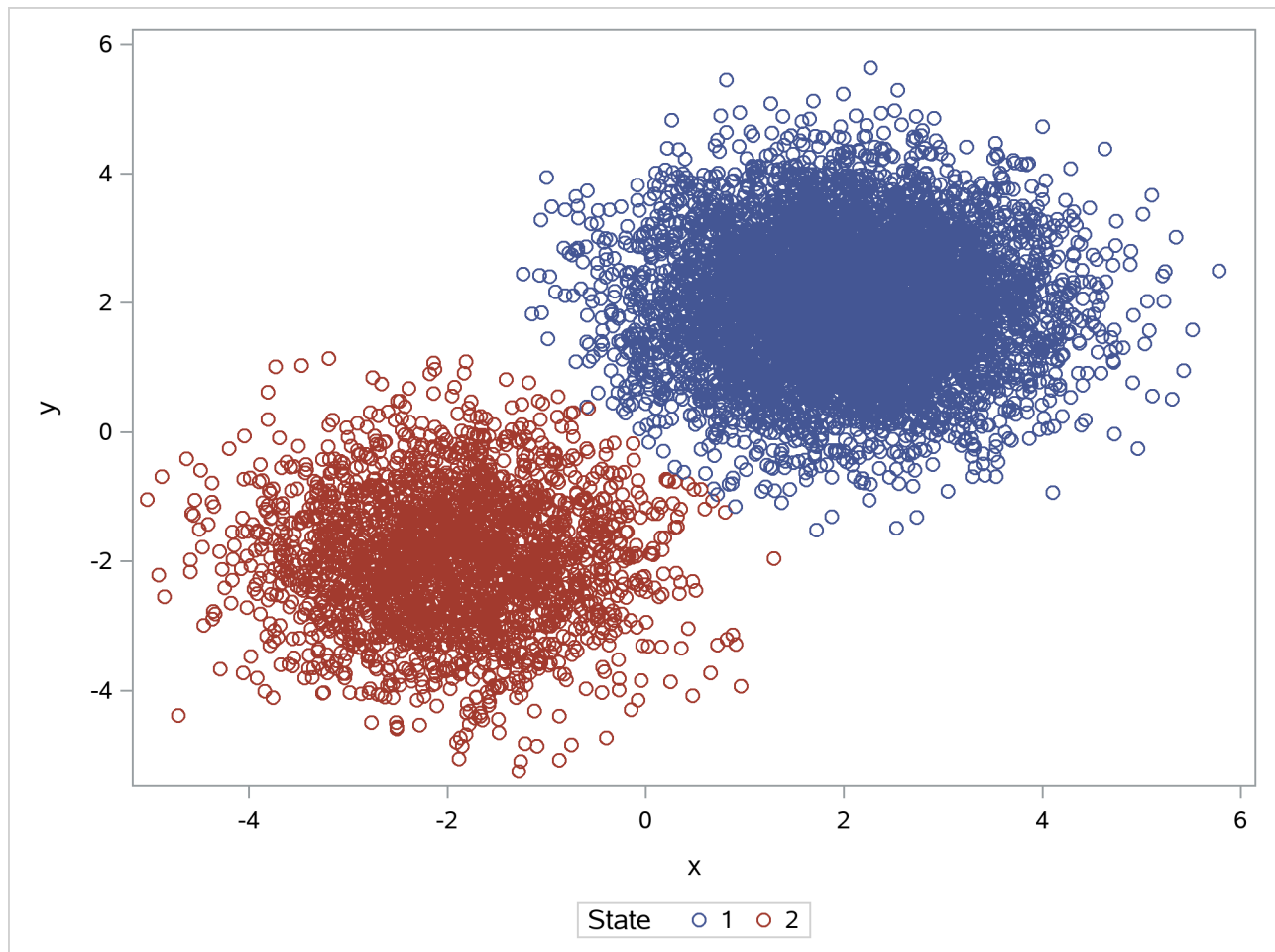
Figure 20.104 Accuracy of GMM on Training Data

accuracy
0.9985

To show how well the data are classified, the following statements plot the classified data points according to the estimated GMM. The plot is shown in [Figure 20.105](#). Comparing it to [Figure 20.102](#), you see that they are almost exactly same.

```
proc sgplot data=combine;
  scatter x=x y=y / group=state;
run;
```

Figure 20.105 Classified Data Points by GMM



It is also easy to apply the estimated GMM to score some new data set by using the analytic store technology. The following statements generate the testing data, apply the ASTORE procedure, and calculate the accuracy of classification. As [Figure 20.106](#) shows, the accuracy is still very high.

```
data two;
  call streaminit(12346);
  do sec = 1 to 10000;
    t = 1;
    u = rand('UNIFORM');
    if(u<=0.75) then do;
      s = 1;
      x = 2 + rand('NORMAL');
    end;
  end;
```

```

        y = 2 + rand('NORMAL');
    end;
    else do;
        s = 2;
        x = -2 + rand('NORMAL');
        y = -2 + rand('NORMAL');
    end;
    output;
end;
run;

proc astore;
    download rstore=mylib.mygmm store='ls_gmm';
    setoption decode 1;
    setoption decode_window 1;
    setoption evaluate 0;
    describe store='ls_gmm';
    score data=two store='ls_gmm' out=gmmout;
run;

data combine2;
    merge gmmout(in=a) two(in=b);
    by sec;
run;

data checkAccuracy2;
    set combine2;
    retain accuracy 0;
    if(DecodedState_BackStep0=s) then accuracy+1;
    if(_N_=10000) then do;
        accuracy = accuracy/_N_;
        output;
    end;
    keep accuracy;
run;

proc print data=checkAccuracy2 noobs; run;

```

Figure 20.106 Accuracy of GMM on Testing Data

accuracy
0.998

Regression and Vector Autoregression

When you specify `NSTATE=1` in the `MODEL` statement for regime-switching regression or autoregression, you are dealing with the regression or vector autoregression (VAR). The autoregressive error model can be treated as a univariate VAR in the mean-adjusted form. For more information about the autoregressive error model, see the chapter “The AUTOREG Procedure” in *SAS/ETS User’s Guide*. For more information about the VARs with or without exogenous variables, see the chapter “The VARMAX Procedure” in *SAS/ETS User’s Guide*.

Examples: HMM Procedure

Example 20.1: Discovering the Hidden Market States by Using the Regime-Switching Autoregression Model

To buy or to sell, that is the question! If you knew it would be a bull market, you might choose to buy the stock; if you knew it would be a bear market, you might choose to sell the stock. However, you cannot directly observe these market states. What you can observe is the stock prices. So, how can you discover the hidden market states from the observable stock price? The question is answered in this example by using regime-switching autoregression models (RS ARs).

Data

Consider the following CRSP NYSE/NYSEMKT/Nasdaq/Arca Value-Weighted Market Index (Center for Research in Security Prices 2018) from January 19, 1926, to December 29, 2017. You can access the CRSP data through the SASECRSP interface engine; for more information, see the chapter “The SASECRSP Interface Engine” in *SAS/ETS User’s Guide*. You can also directly access the data if you install the CRSP data locally. If you install the CRSP data in the folder XXX, the following statements extract the daily stock prices (that is, market indices) into the data set vwmiO:

```
*****;
* DATA;
*****;

title 'Discovering the Hidden Market States';
libname crspInd "XXX";
data vwmiO;
    set crspInd.sfz_dind(where=(KYINDNO=1000200
        and caldt>='19JAN1926'd and caldt<='29DEC2017'd));
    format date MMDDYY10.;
    date = caldt; price = aind;
    keep date price;
run;
```

The stock-price series is nonstationary and not suitable to be directly modeled; hence, the stationary weekly-return series is modeled. The following code generates the weekly returns through the daily stock prices and saves them in two data sets. One data set, vwmi, is the full-sample data set, which contains all weekly returns. The other data set, vwmiln, is the in-sample data set, which contains weekly returns before December 31, 2000. The weekly returns after January 1, 2001, are in the out-of-sample period and are used for testing the models.

```
%let ds = vwmi;
%let cutDate = '31DEC2000'd;
data &ds. &ds.In;
    set &ds.O;
    retain cumReturn 0;
    return = (log(price)-log(lag(price)))*100;
    if(return~=. ) then cumReturn + return;
```



```

if(mod(_N_,5)=1 and _N_>1) then do;
    returnw = cumReturn;
    w + 1;
    output &ds.;
    if (date<=&cutDate.) then output &ds.In;
    cumReturn = 0;
end;
keep w date returnw price;
label w="Week Index" date="Date" returnw="Weekly Return"
      price="Market Index";
run;

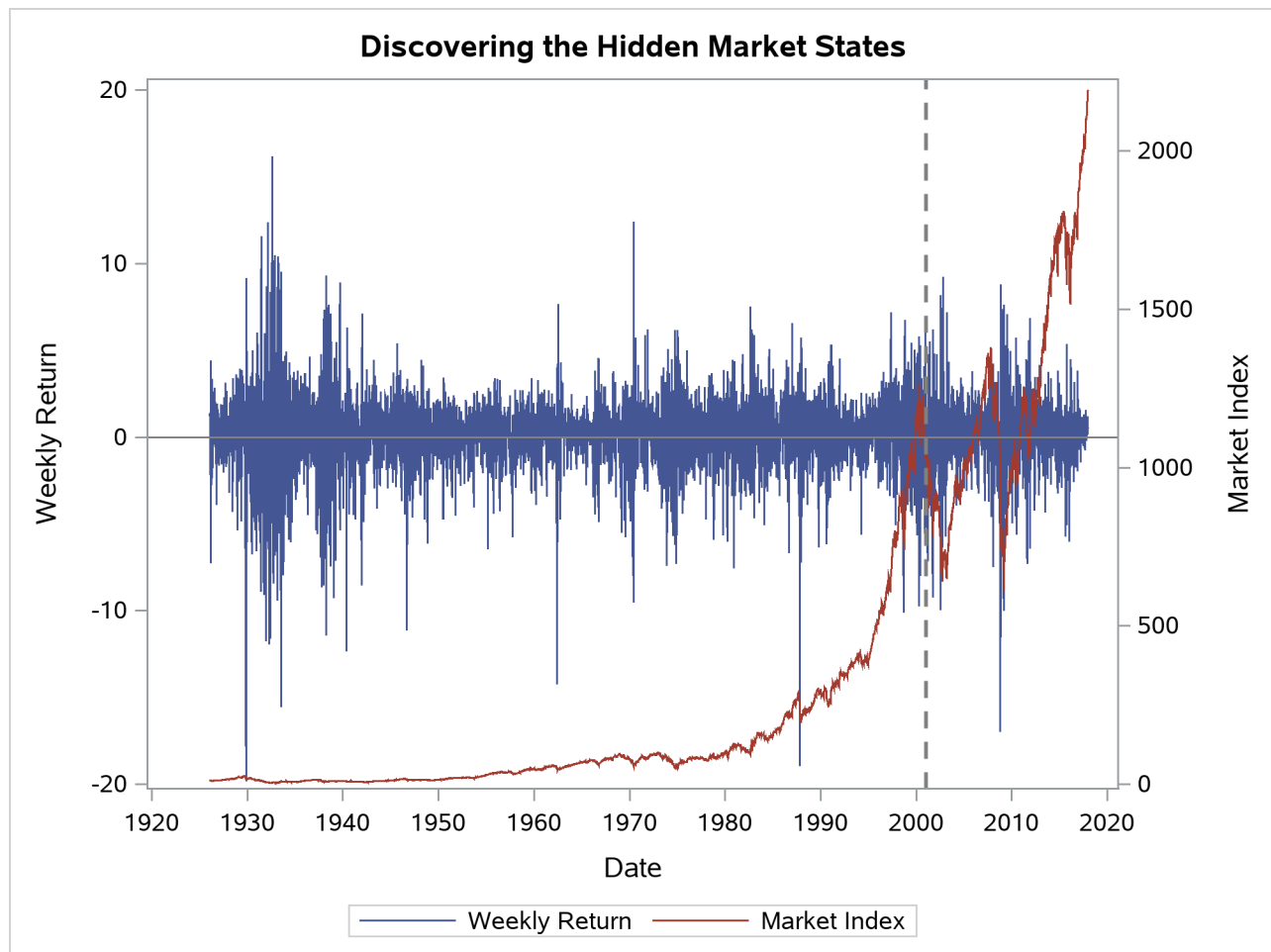
```

The following statements plot the stock prices and weekly returns, as shown in [Output 20.1.1](#). The in-sample and out-of-sample periods are separated by a vertical line.

```

proc sgplot data=&ds.;
    series x=date y=returnw;
    series x=date y=price / y2axis;
    refline 0 / axis=y lineattrs=(color=gray thickness=1);
    refline &cutDate. / axis=x
                    lineattrs=(color=gray thickness=2 pattern=dash);
    yaxis max=20 min=-20;
run;

```

Output 20.1.1 Stock Prices and Weekly Returns

Before you estimate any model, make a copy of the data in the library, as in the following DATA steps:

```
data mylib.&ds.In; set &ds.In; run;
data mylib.&ds.; set &ds.; run;
```

Analysis with a Simple Model

Consider a simple case. First, assume that there are two hidden market states, bear market state and bull market state, which can be modeled by exactly two corresponding hidden states in an HMM; that is, consider a bistate HMM. Second, assume that the lagged weekly returns cannot help predict current weekly return at each hidden state; that is, the autoregressive order is zero at each hidden state. Hence, as a start, consider a bistate RS-AR(0) model. The following statements estimate the bistate RS-AR(0) model on weekly returns. The input data table, specified in the required DATA= option in the PROC HMM statement, contains the in-sample data. In order to estimate the model, the ID and MODEL statements must be specified. Since the data are sorted by the variable date, the variable date is assigned to the required TIME= option. However, since the data can also be regarded as being sorted by the variable w, the variable w, instead of the variable date, can be assigned to the TIME= option; if you do so, there is no impact on the model estimation. In the MODEL statement, the dependent variable, the weekly return returnw, is specified first; the TYPE= option is set to AR for the RS-AR model; the NSTATE= option is set to 2 for the bistate model; and the YLAG= option

is set to 0 for an AR(0) model. The SMOOTH, DECODE, and EVALUATE statements specify where to output, respectively, the smoothed probabilities, decoded state path, and log likelihood up to each time point.

```
*****;
* ANALYSIS WITH A SIMPLE MODEL
*****;

proc hmm data=mylib.&ds.In;
  id time=date;
  model returnw / type=ar nstate=2 ylag=0;
  smooth out=mylib.&ds.inSmooth_k2To2_p0To0;
  decode out=mylib.&ds.inDecode_k2To2_p0To0;
  evaluate out=mylib.&ds.inEval_k2To2_p0To0;
run;
```

The number of observations and model information are shown in [Output 20.1.2](#).

Output 20.1.2 Number of Observations and Model Information
Discovering the Hidden Market States

Observation Information	
Number of Observations	3999
Number of Missing Observations	0

Model Information	
Type of Model	Regime-Switching Autoregression Model
Stationary	Yes
Number of States	2
Number of Dependent Variables	1
Number of Exogenous Variables	0

The estimates of the initial state probability vector (ISPV) are shown in [Output 20.1.3](#). In this example, the hidden Markov chain is assumed to be stationary. Hence, the steady-state probability distribution (STPD, which is also called the equilibrium distribution or stationary distribution) of the Markov chain is equal to the ISPV.

Output 20.1.3 Estimates of Initial State Probability Vector
Discovering the Hidden Market States

Initial State Probabilities	
State	Estimation
1	0.76747
2	0.23253

The estimates of the transition probability matrix (TPM) are shown in [Output 20.1.4](#).

Output 20.1.4 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	1	2
1	0.98111	0.01889
2	0.06235	0.93765

The estimates of the intercept for each state are shown in [Output 20.1.5](#).

Output 20.1.5 Estimate of Intercept for Each State

Constant	
State	returnw
1	0.28710
2	-0.47243

The estimates of the variance for each state are shown in [Output 20.1.6](#).

Output 20.1.6 Estimate of Variance for Each State

Innovations Covariances		
State	Variable	returnw
1	returnw	2.29936
2	returnw	17.02648

The estimates of all parameters are shown in [Output 20.1.7](#), which displays columns for the parameter name, estimate value, standard error, t value, and p -value.

Output 20.1.7 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.981109	0.003507	279.78	<.0001
TPM1_2	0.018891	0.003507	5.39	<.0001
TPM2_1	0.062352	0.011274	5.53	<.0001
TPM2_2	0.937648	0.011274	83.17	<.0001
CONST1_1	0.287100	0.030130	9.53	<.0001
CONST2_1	-0.472426	0.142740	-3.31	0.0009
COV1_1_1	2.299364	0.090090	25.52	<.0001
COV2_1_1	17.026478	1.052833	16.17	<.0001

The log likelihood and information criteria are shown in [Output 20.1.8](#).

Output 20.1.8 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-8497.46701
AIC	17006.934019
AICC	17006.955061
BIC	17044.696817
HQC	17020.320118

The smoothed probabilities of states for each observation are output in the data table `mylib.vwmilnSmooth_k2To2_p0To0`. This example discusses one type of inference. The mean and variance (or standard error) of each state can be calculated through the weighted sample, where the smoothed probabilities are the weights. The following statements calculate and print the sample mean and standard error of each state, as shown in [Output 20.1.9](#). In the case of an RS-AR(0) model that has no regressors other than the intercept (which is equivalent to the Gaussian HMM), the sample mean and standard error of each state are almost the same as, respectively, the intercept estimate and the square root of variance estimate of each state.

```

*print sample mean and standard error for each state
  according to the weighted sample,
  where the smoothed probabilities are the weight;
data &ds.inSmooth_k2to2_p0to0;
  set mylib.&ds.inSmooth_k2to2_p0to0;
run;
proc sort data=&ds.inSmooth_k2to2_p0to0; by date; run;
data &ds.inWeighted;
  merge &ds.inSmooth_k2to2_p0to0(in=a) &ds.in(in=b);
  by date;
  if(a=b);
run;
%macro printMeanAndStdErrOfAState(k);
  data &ds.inMeanStdErr;
    set &ds.inWeighted end=eof;
    array cWeights{&k.} _TEMPORARY_;
    array cMeans{&k.} mean1-mean&k.;
    array cVars{&k.} sd1-sd&k.;
    array weights{&k.} state1-state&k.;
    retain cWeights cMeans cVars;
    if(_N_=1) then do;
      do i = 1 to &k.;
        cWeights[i] = weights[i];
        cMeans[i] = weights[i]*returnw;
        cVars[i] = weights[i]*returnw*returnw;
      end;
    end;
  else do;
    do i = 1 to &k.;
      cWeights[i] = cWeights[i] + weights[i];
      cMeans[i] = cMeans[i] + weights[i]*returnw;
      cVars[i] = cVars[i] + weights[i]*returnw*returnw;
    end;
  end;
end;
if eof then do;

```

```

do i = 1 to &k.;
  if(cWeights[i]>0) then do;
    cMeans[i] = cMeans[i] / cWeights[i];
    cVars[i] = cVars[i] / cWeights[i] - cMeans[i]*cMeans[i];
    if(cVars[i]>=0) then cVars[i] = sqrt(cVars[i]);
  end;
  else do;
    cMeans[i] = .;
    cVars[i] = .;
  end;
end;
%do i = 1 %to &k.;
  call symputx("mu&i.",cMeans[&i.], 'G');
  call symputx("sigma&i.",cVars[&i.], 'G');
  state = &i.; mean = mean&i.; stdErr = sd&i.;
  output;
%end;
end;
label state="State" mean="Sample Mean"
      stdErr="Sample Standard Error";
keep state mean stdErr;
run;
proc print data=&ds.inMeanStdErr noobs label; run;
%mend;
%printMeanAndStdErrOfAState(2);

```

Output 20.1.9 Sample Mean and Standard Error of Each State**Discovering the Hidden Market States**

State	Sample	
	Mean	Standard Error
1	0.28710	1.51637
2	-0.47243	4.12632

The following statements plot each state as a Gaussian kernel, which shows how each state looks. As shown in [Output 20.1.10](#), it is obvious that state 1 should represent the bull market state (positive expected return and low risk) and state 2 the bear market state (negative expected return and high risk).

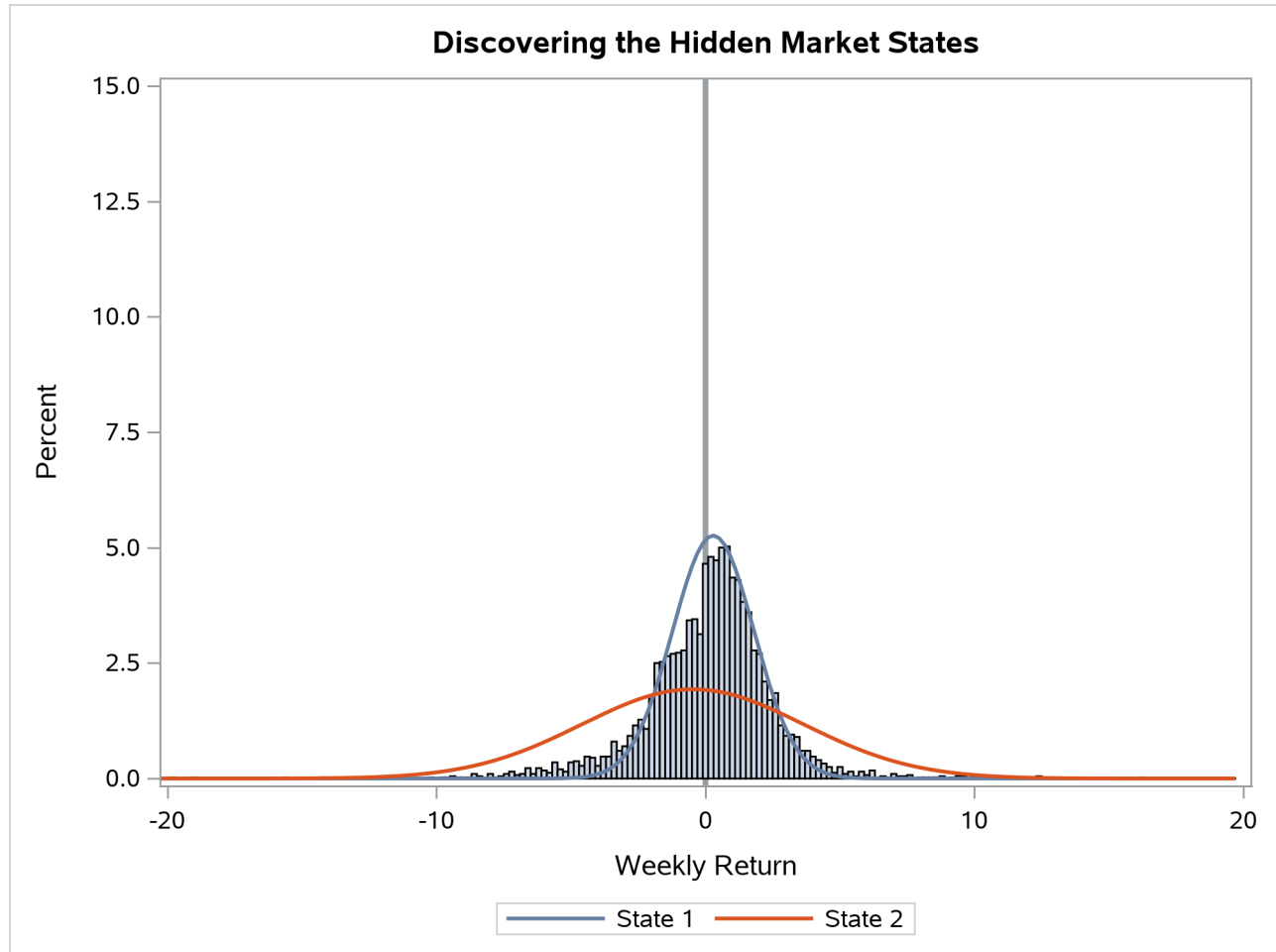
```

%macro plotState(myData,myColumn,ks,ke);
  proc sgplot data=&myData.;
    refline 0 / axis=x lineattrs=(thickness=3);
    histogram &myColumn. / nbins=200;
    %do i = &ks. %to &ke.;
      density &myColumn. / type=normal(mu=&&mu&i. sigma=&&sigma&i.)
        name="state&i."
        legendlabel="State &i.";
    %end;
    keylegend %do i = &ks. %to &ke.; "state&i." %end;;
    xaxis min=-20 max=20 ;
    yaxis min=0 max=15;
  run;
%mend;

```

```
%plotState(&ds.In, returnw, 1, 2);
```

Output 20.1.10 Gaussian Distribution of Each State



The decoded path of states, which provide a historical view of what happened, given all the available data, is output in the data table `mylib.vwmInDecode_k2to2_p0to0`. The decoded path assigns each time point a market state. It seems that the hidden market states have become observable. The following statements plot the decoded market states, as shown in [Output 20.1.11](#):

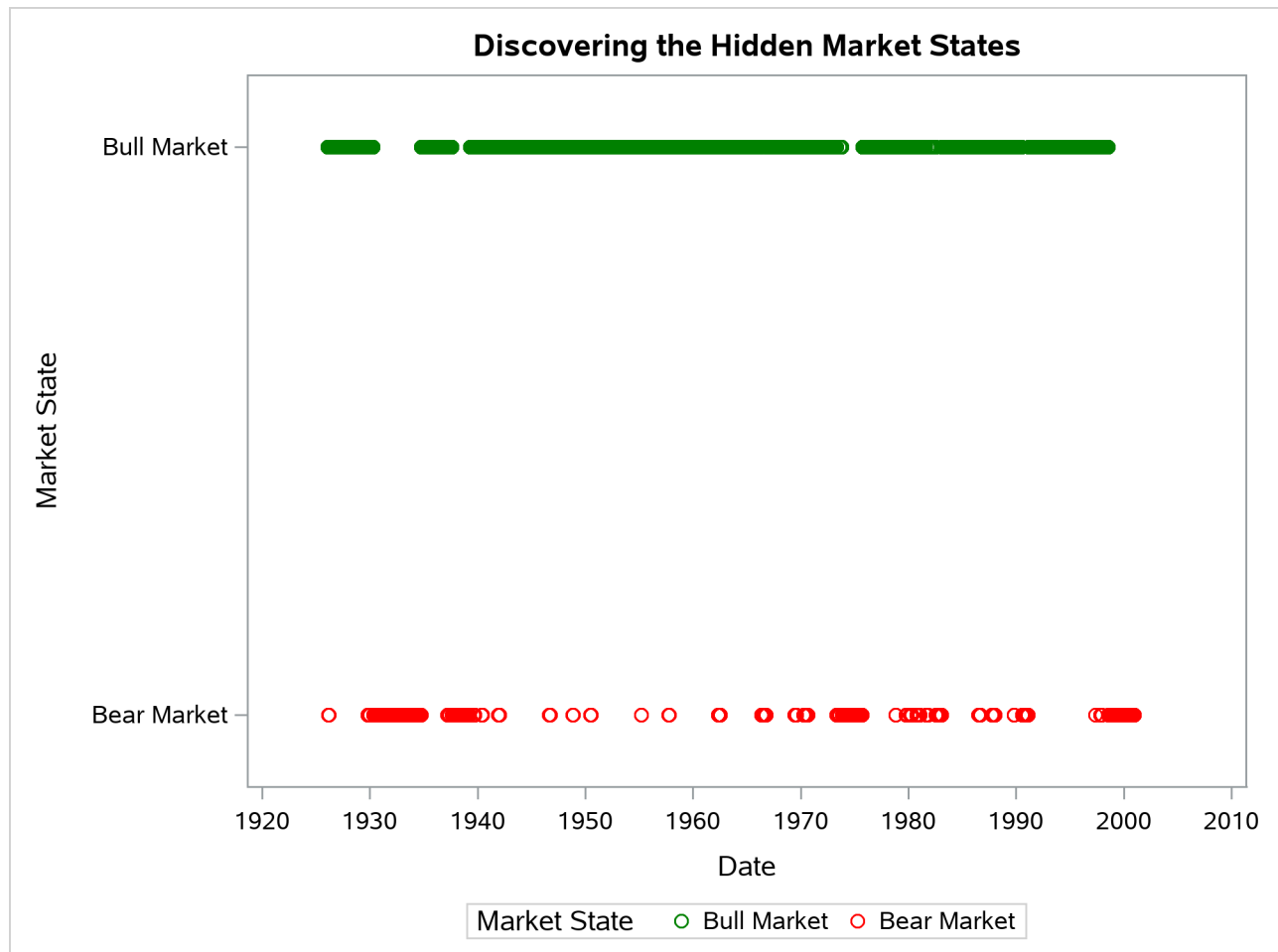
```
* plot decoded results;
data &ds.InDecode_k2to2_p0to0; set mylib.&ds.InDecode_k2to2_p0to0; run;
proc sort data=&ds.InDecode_k2to2_p0to0; by date; run;
data decodeMaketStates;
  set &ds.In;
  set &ds.InDecode_k2to2_p0to0;
  array returnws returnw1-returnw2;
  array prices price1-price2;
  if(state~=.) then do;
    if(state=1) then do;
      returnws[1] = returnw; prices[1] = price;
      marketState="Bull Market";
    end;
    if(state=2) then do;
```

```

        returnws[2] = returnw; prices[2] = price;
        marketState="Bear Market";
    end;
end;
label marketState="Market State";
run;
proc sgplot data=decodeMaketStates;
    styleattrs datacontrastcolors=(green red);
    scatter x=date y=marketState / group=marketState;
    yaxis label="Market State" offsetmin=0.1 offsetmax=0.1 reverse;
run;

```

Output 20.1.11 Decoded Market States



You can incorporate the decoded market states with other variables of interest. The following statements plot the decoded weekly returns, as shown in [Output 20.1.12](#):

```

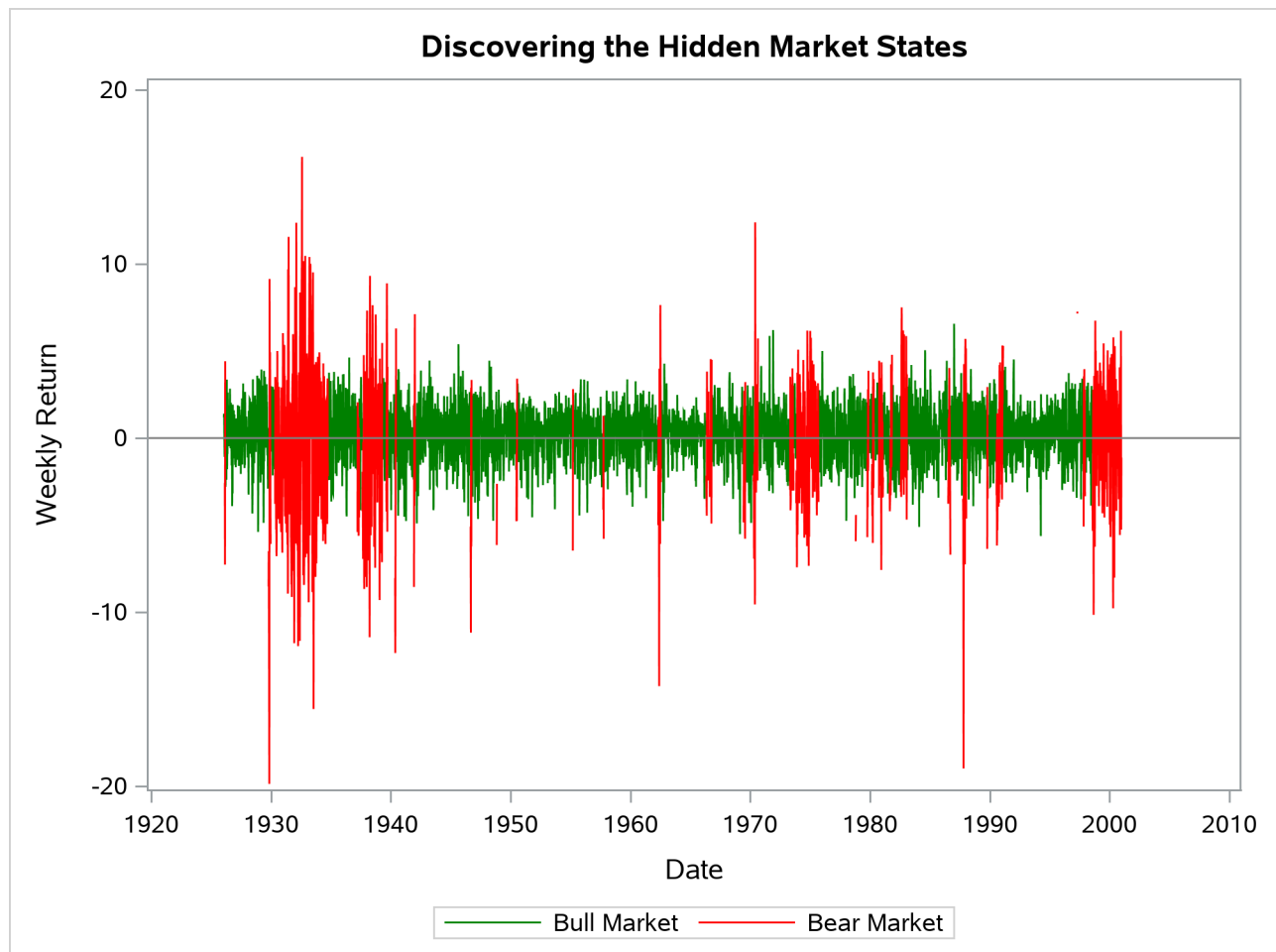
proc sgplot data=decodeMaketStates;
    series x=date y=returnw1 / break lineattrs=(color=green)
        legendlabel="Bull Market";
    series x=date y=returnw2 / break lineattrs=(color=red)
        legendlabel="Bear Market";
    refline 0 / axis=y lineattrs=(color=gray thickness=1);

```



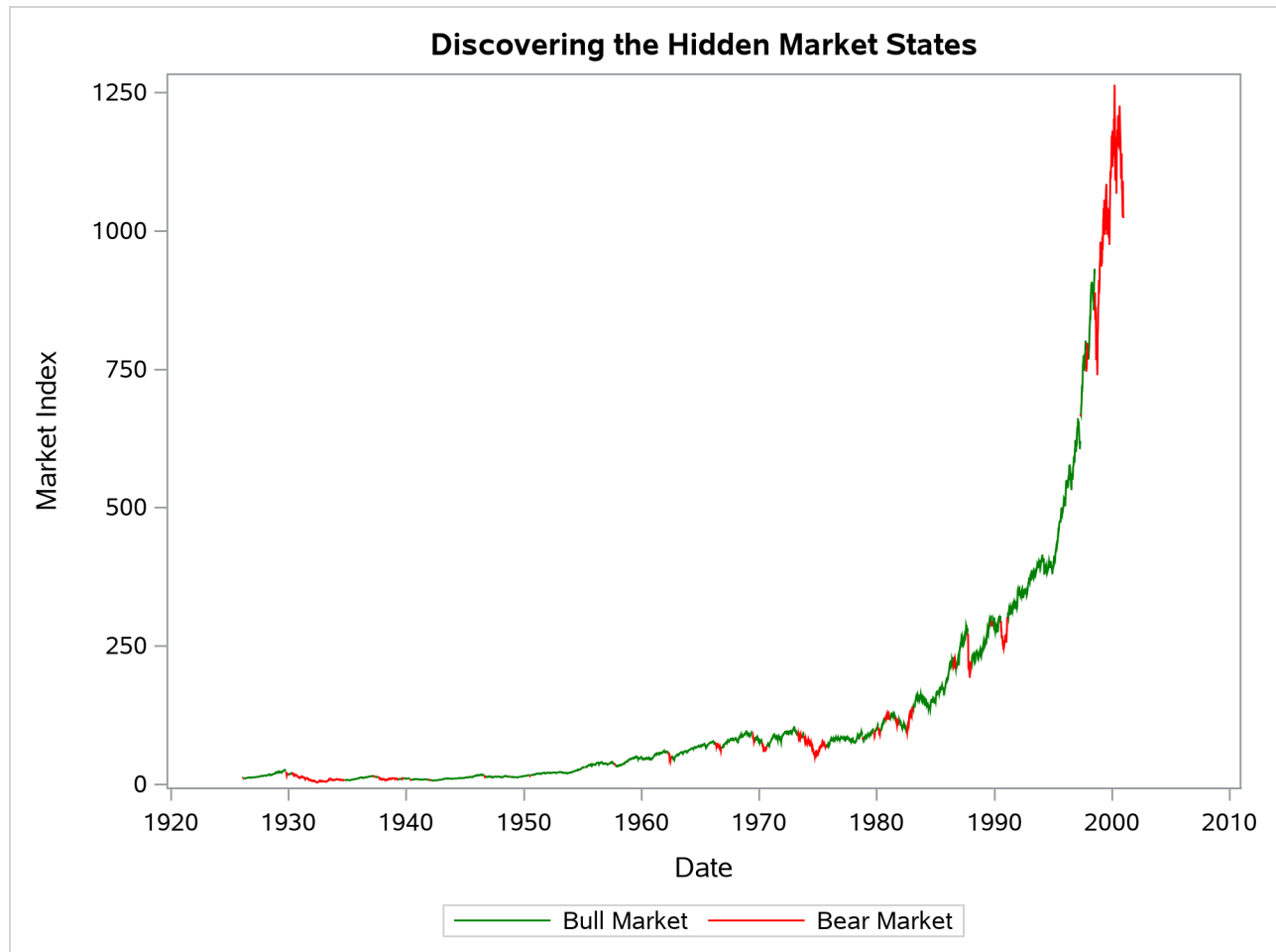
```
yaxis max=20 min=-20 label="Weekly Return";
run;
```

Output 20.1.12 Decoded Weekly Returns



The following statements plot the decoded stock prices, as shown in [Output 20.1.13](#):

```
proc sgplot data=decodeMaketStates;
  series x=date y=price1 / break lineattrs=(color=green)
          legendlabel="Bull Market";
  series x=date y=price2 / break lineattrs=(color=red)
          legendlabel="Bear Market";
  yaxis label="Market Index";
run;
```

Output 20.1.13 Decoded Stock Prices

The following statements plot the log likelihood of each observation conditional on its past; the plot is shown in [Output 20.1.14](#).

```
* plot conditional log likelihood;
%macro plotEvaluation(inds, outds, qConditional);
  data &outds.;
    set &inds.;
    format date MMDDYY10.;
run;
%if (&qConditional.=1) %then %do;
  proc sort data=&outds.;
    by date;
  run;
  data &outds.;
    set &outds.;
    if(_N_=1) then
      condLogLikelihood = logLikelihood;
    else
      condLogLikelihood = logLikelihood - lag(logLikelihood);
  run;
```

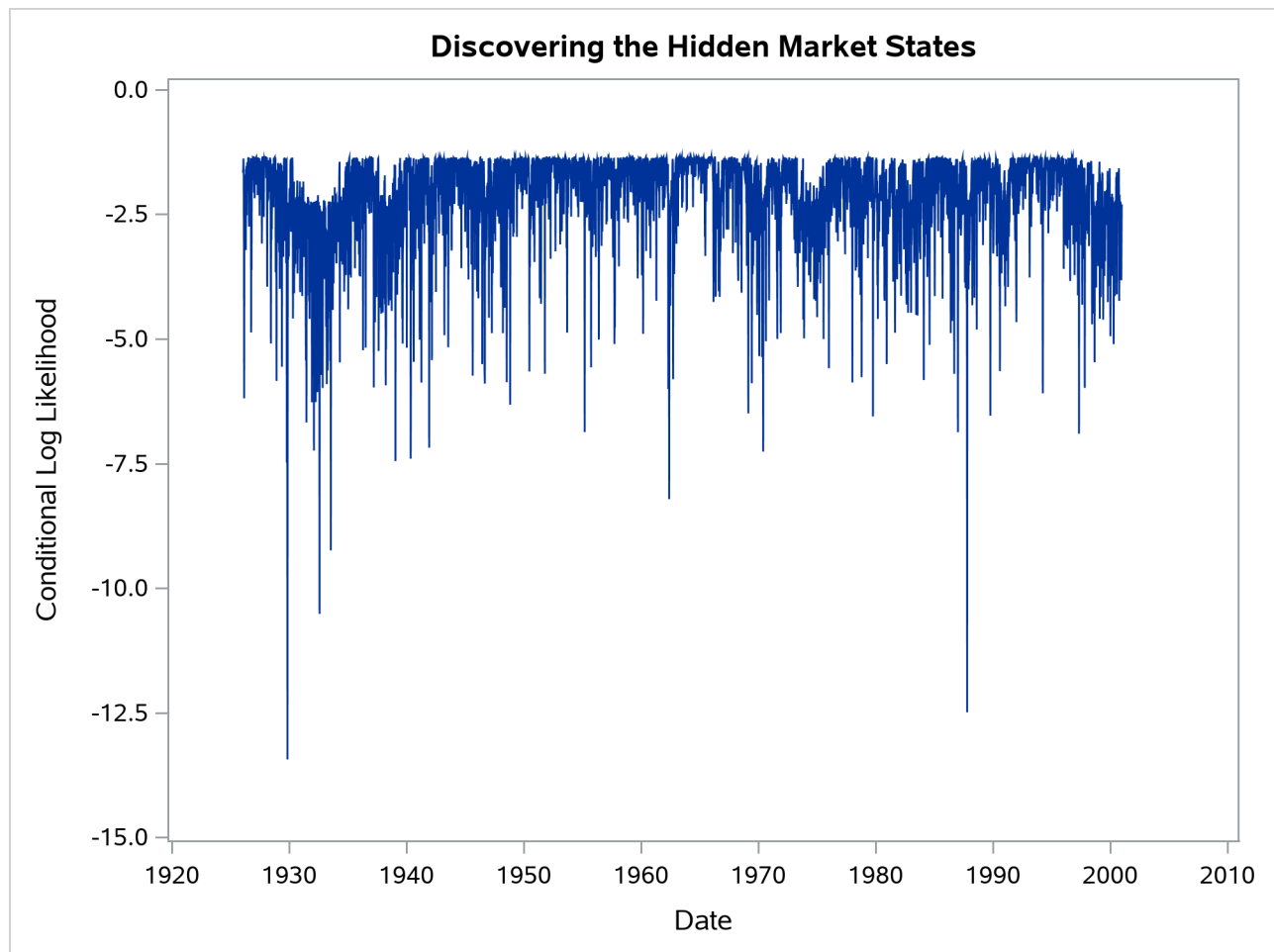
```

proc sgplot data=&outds.;
  series x=date y=condLogLikelihood;
  yaxis label='Conditional Log Likelihood' min=-15 max=0;
run;
%end;
%else %do;
  proc sgplot data=&outds.;
    series x=date y=logLikelihood;
    yaxis label='Log Likelihood';
  run;
%end;
%mend plotEvaluation;

%plotEvaluation(mylib.&ds.inEval_k2to2_p0to0, &ds.inEval_k2to2_p0to0, 1);

```

Output 20.1.14 Conditional Log Likelihood



As shown in [Output 20.1.14](#), some observations are explained well by the model, but there are still many exceptional observations whose conditional log likelihoods are much smaller than others. This means that there might be model misspecification, which can be addressed by solving the model selection problem.

Model Selection

To model the weekly returns, how many states should be considered? How many AR lags should a state include? These are common questions in the model selection process. In this example, the best model is selected using Akaike's information criterion (AIC): the smaller the AIC, the better the model.

The following macro estimates the model or models according to the arguments:

```
*****;
* MODEL SELECTION;
*****;

%macro estimateRSAR(myds, inEstDs, kStart, kEnd, pStart, pEnd, method,
                    maxiter, qMultiStart);
  proc hmm data=&myds. labelSwitch=(sort=none)
    outstat=&myds.&method.Stat_k&kStart.To&kEnd._p&pStart.To&pEnd.;
    id time=date;
    model returnw / type=ar nstate=&kStart.:&kEnd. ylag=&pStart.:&pEnd.
      method=&method.;
    optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint
      maxiter=&maxiter. Multistart=&qMultiStart.;
    score outmodel=&myds.&method.Model_k&kStart.To&kEnd._p&pStart.To&pEnd.;
    learn out=&myds.&method.Learn_k&kStart.To&kEnd._p&pStart.To&pEnd.
      %if %length(&inEstDs.)>0 %then %do; in=&inEstDs. %end;
    ;
    display / excludeall;
  run;
%mend;
```

First, the following statements estimate two- to nine-state RS-AR(0) models. The MAP method is applied to bound and smooth the objective functions. The parameter estimates are saved for further use as the initial values for new estimations. For the best results, multistart mode is strongly recommended, although it might be time consuming.

```
* (1) MAP + MULTISTART for AR(0);
* be aware that the following macro might take tens of hours to finish;
* uncomment it to run;
* %estimateRSAR(myds=mylib.&ds.In, inEstDs=,
  kStart=2, kEnd=9, pStart=0, pEnd=0,
  method=MAP, maxiter=128, qMultiStart=1);
```

Then, the following statements estimate two- to nine-state RS-AR(0) to AR(5) models. The ML method is used because the AIC is based on the log likelihood. The parameter estimates of RS AR(0) models from previous MAP method estimations are used to ensure that the estimates in the ML method are reasonable. The k -state RS AR(p) models, $k = 2, \dots, 9$, $p = 1, \dots, 5$, use the corresponding k -state RS AR($p - 1$) models' parameter estimates as initial values. This is the default implementation in the HMM procedure. The multistart mode is not needed any longer because the initial values are good.

```
* (2) ML + initial values estimated from MAP's AR(0) for AR(0)
  and initial values estimated from AR(p-1) for AR(p), p>0;
* be aware that the following macro might take tens of minutes to finish;
* uncomment it to run;
* %estimateRSAR(myds=mylib.&ds.In, inEstDs=mylib.&ds.InMAPLearn_k2to9_p0to0,
  kStart=2, kEnd=9, pStart=0, pEnd=5,
```

```
method=ML, maxiter=128, qMultiStart=0);
```

Estimating all 48 of the two- to nine-state RS AR(0) to AR(5) models produces AIC values. The following statements print the AIC values, which are shown in [Output 20.1.15](#):

```
* print fit statistics in matrix form;
data &ds.InMLStat_k2to9_p0to5; set mylib.&ds.InMLStat_k2to9_p0to5; run;
proc sort data=&ds.InMLStat_k2to9_p0to5; by modelIndex; run;
%macro printFitStat(fitStat);
  data &ds.In&fitStat.;
    set &ds.InMLStat_k2to9_p0to5 end=eof;
    array &fitStat.s{8,6} _TEMPORARY_;
    array yLags{6} yLag0-yLag5;
    &fitStat.s[nState-1,yLag+1] = &fitStat.;
    if eof then do;
      do i = 1 to 8;
        nState = i+1;
        do j = 1 to 6; yLags(j) = &fitStat.s[i,j]; end;
        output;
      end;
    end;
    keep nState yLag0-yLag5;
run;
proc print data=&ds.In&fitStat. label noobs
  style(header)={textalign=center};
  var nState yLag0-yLag5;
  label nState='k' yLag0='p = 0' yLag1='p = 1' yLag2='p = 2'
        yLag3='p = 3' yLag4='p = 4' yLag5='p = 5';
run;
%mend;
%printFitStat(AIC);

* Uncomment the following macros if you want to see other fit statistics;
* %printFitStat(logLikelihood);
* %printFitStat(AICC);
* %printFitStat(BIC);
* %printFitStat(HQC);
```

[Output 20.1.15](#) indicates that the smallest AIC corresponds to the eight-state RS AR(4) model.

Output 20.1.15 AICs for 48 RS AR Models

Discovering the Hidden Market States

k	p = 0	p = 1	p = 2	p = 3	p = 4	p = 5
2	16987.55	16979.59	16981.64	16985.00	16988.80	16992.07
3	16820.74	16811.38	16816.45	16821.02	16825.88	16826.56
4	16772.85	16764.64	16754.79	16757.74	16763.77	16767.28
5	16728.05	16733.38	16738.14	16743.40	16750.67	16753.11
6	16697.18	16684.36	16684.90	16689.57	16695.59	16700.58
7	16701.64	16693.28	16682.39	16685.51	16690.77	16695.31
8	16685.88	16688.43	16687.35	16668.80	16650.11	16659.51
9	16711.81	16699.75	16698.66	16707.04	16679.98	16684.87

The Selected Model and In-Sample Inference

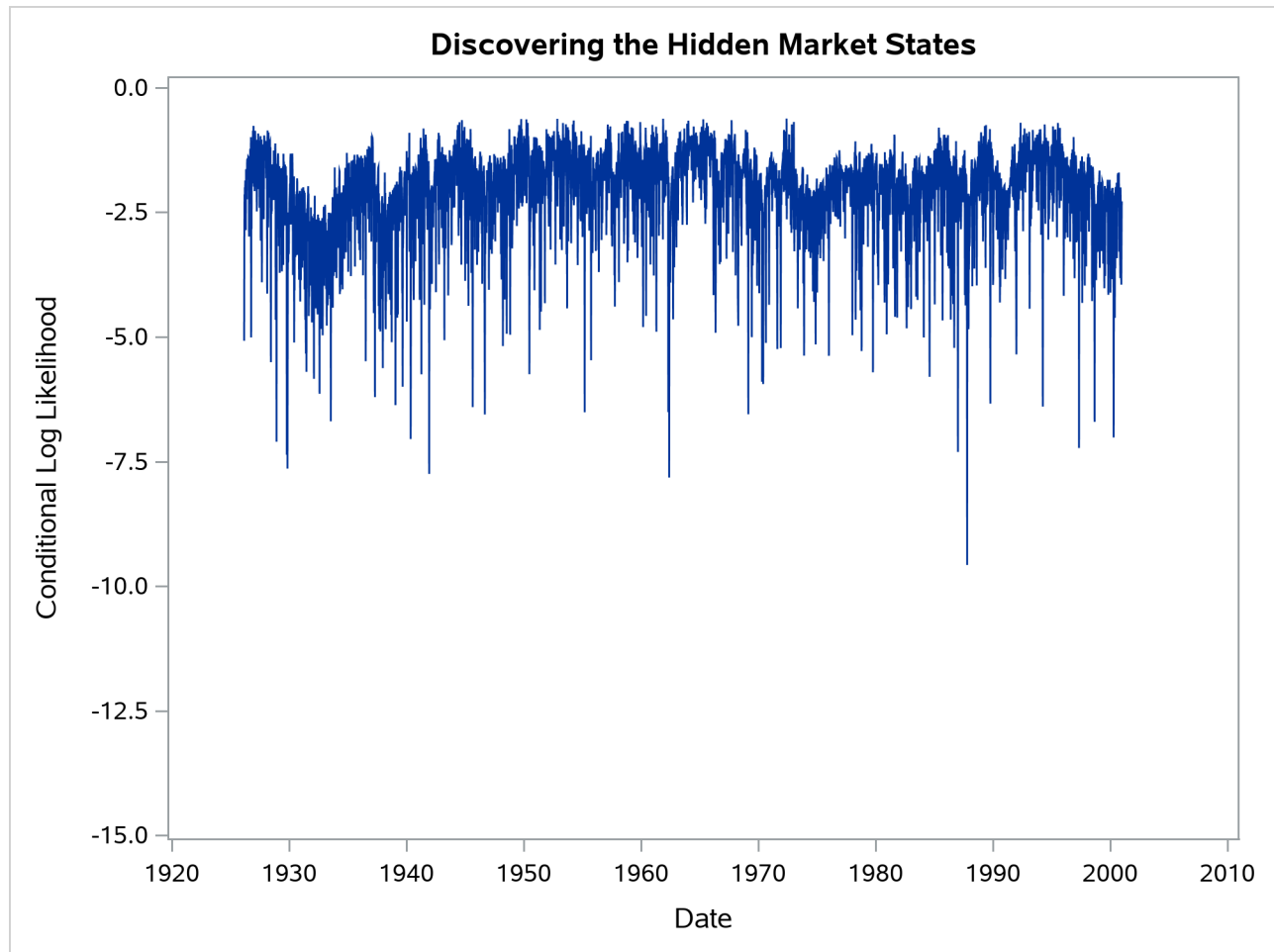
The following statements estimate the AIC-selected eight-state RS AR(4) model:

```
*****;
* THE SELECTED MODEL AND IN-SAMPLE INFERENCE
*****;

* select eight-state RS-AR(4) model according to AIC or AICC;
* adjust data to be the same sample size for AR(5);
proc hmm data=mylib.&ds.in(where=(date>='30JAN1926'd));
  id time=date;
  model returnw / type=ar nstate=8 ylag=4;
  optimize printLevel=3 algorithm=interiorpoint maxiter=0;
  score outmodel=mylib.&ds.inMLModel_k8To8_p4To4;
  learn out=mylib.&ds.inMLLearn_k8To8_p4To4
    in=mylib.&ds.inMLLearn_k2to9_p0to5;
  smooth out=mylib.&ds.inSmooth_k8to8_p4to4;
  decode out=mylib.&ds.inDecode_k8to8_p4to4;
  evaluate out=mylib.&ds.inEval_k8to8_p4to4;
  display tpm;
run;
```

The following statement plots the conditional log likelihoods. Compared to the bivariate RS-AR(0) model (as shown in [Output 20.1.14](#)), the eight-state RS-AR(0) model (as shown in [Output 20.1.16](#)) has much higher conditional log likelihoods, which means a much better in-sample goodness of fit.

```
%plotEvaluation(mylib.&ds.inEval_k8to8_p4to4, &ds.inEval_k8to8_p4to4, 1);
```

Output 20.1.16 Conditional Log Likelihoods

The selected model has 112 parameters. 48 of which are related to observation distributions and are often called observation parameters. The following statements print the observation parameters in [Output 20.1.17](#). The analysis of each AR(4) process for each state is beyond this example.

```
* print observation parameters;
data &ds.InObsParm;
  set mylib.&ds.InMLLearn_k8To8_p4To4(where=(type='EST' and index>=17));
  if(index=17) then parameter='Constant';
  if(index=19) then parameter='AR(1)';
  if(index=21) then parameter='AR(2)';
  if(index=23) then parameter='AR(3)';
  if(index=25) then parameter='AR(4)';
  if(index=27) then parameter='Variance';
run;
proc sort data=&ds.InObsParm; by index; run;
proc print data=&ds.InObsParm noobs label;
  var parameter state1-state8;
  label state1='State 1' state2='State 2' state3='State 3'
        state4='State 4' state5='State 5' state6='State 6'
        state7='State 7' state8='State 8';
run;
```

Output 20.1.17 Observation Parameters**Discovering the Hidden Market States**

Parameter	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8
Constant	0.93305	1.15326	-0.59159	1.68887	-0.82124	-2.68701	-0.01897	-0.5056
AR(1)	-0.23663	0.03568	-0.06096	-0.17268	-0.04365	-0.34339	-0.05648	0.1213
AR(2)	0.00568	-0.07571	0.33570	-0.08520	-0.09594	-0.11391	0.06549	0.0259
AR(3)	-0.06493	-0.07301	0.01605	-0.06019	0.04540	-0.60512	0.11289	0.0025
AR(4)	-0.02869	-0.08348	-0.01595	-0.02281	0.12060	-0.29044	-0.11970	0.0613
Variance	0.23055	0.77141	1.21104	1.44655	1.54215	3.64774	9.42764	31.8773

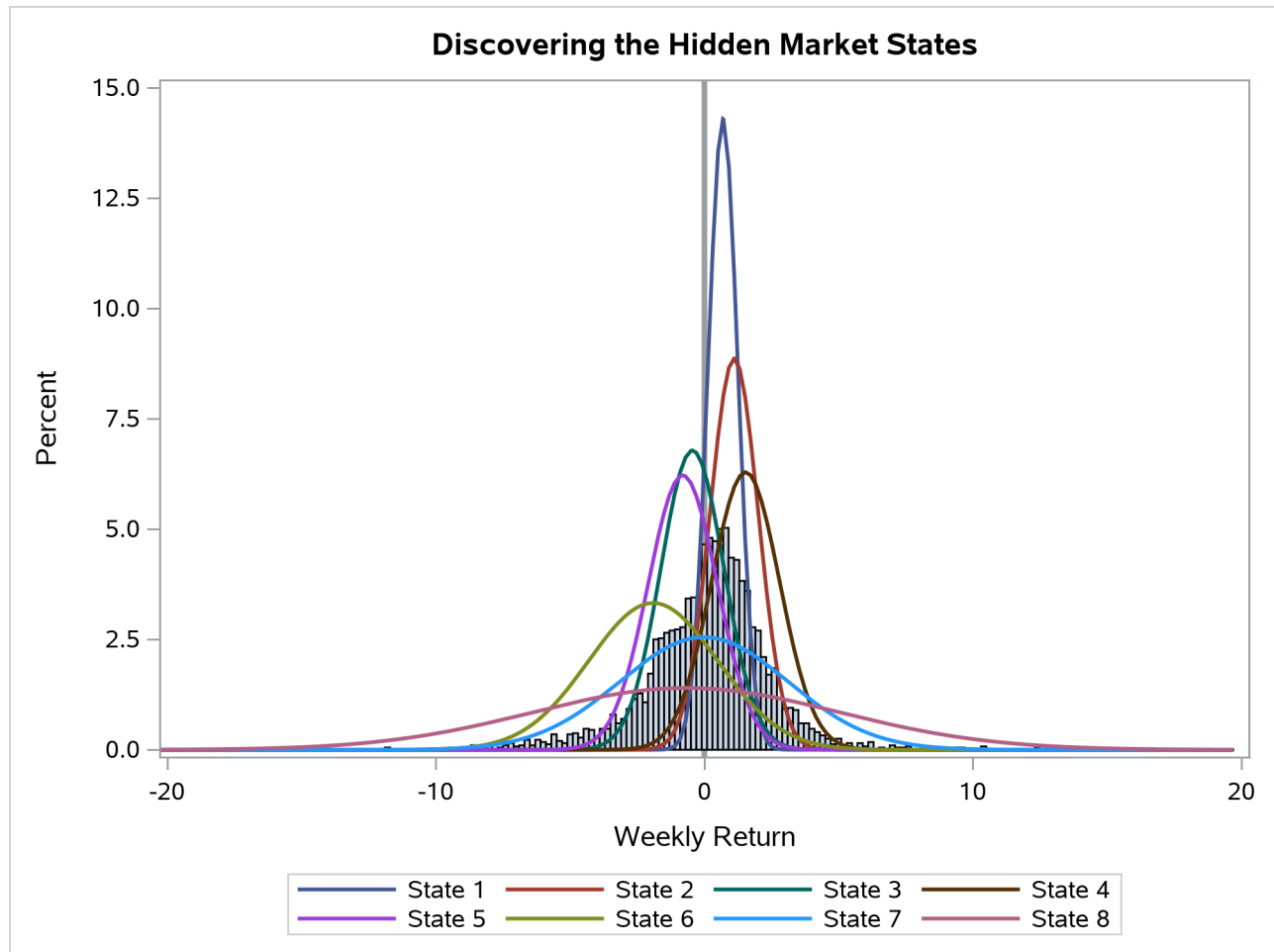
The following statements calculate and print the sample mean and standard error for each state in [Output 20.1.18](#) and then plot the corresponding Gaussian kernels in [Output 20.1.19](#):

```
data &ds.inSmooth_k8to8_p4to4;
    set mylib.&ds.inSmooth_k8to8_p4to4 (where=(date>='25FEB1926'd));
run;
proc sort data=&ds.inSmooth_k8to8_p4to4; by date; run;
data &ds.inTruncated; set &ds.in (where=(date>='25FEB1926'd)); run;
data &ds.inWeighted;
    merge &ds.inSmooth_k8to8_p4to4 (in=a) &ds.inTruncated (in=b);
    by date;
    if (a=b);
run;
%printMeanAndStdErrOfAState(8);

%plotState(&ds.In, returnw, 1, 8);
```

Output 20.1.18 Sample Mean and Standard Error of Each State**Discovering the Hidden Market States**

State	Sample Mean	Sample Standard Error
1	0.68995	0.55836
2	1.10751	0.90016
3	-0.44647	1.17678
4	1.53207	1.26888
5	-0.80688	1.28357
6	-1.91024	2.39908
7	-0.01947	3.12820
8	-0.67999	5.70094

Output 20.1.19 Gaussian Kernels for Eight States

As shown in [Output 20.1.19](#), states 1, 2, and 4 seem like a bull market; they have positive expected returns with low risk. The remaining states seem like a bear market; they have negative expected returns or very high risk (or both). However, the market states and their corresponding states are not that simple. The transition probability matrix of the eight-state AR(4) model is shown in [Output 20.1.20](#).

Output 20.1.20 Estimates of Transition Probability Matrix

Discovering the Hidden Market States

Estimated Transition Probability Matrix								
State	1	2	3	4	5	6	7	8
1	0.00000	0.00000	0.98736	0.00000	0.00000	0.01264	0.00000	0.00000
2	0.57934	0.36501	0.00000	0.00000	0.02789	0.02776	0.00000	0.00000
3	0.00000	0.42100	0.57900	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.01976	0.00000	0.61521	0.36503	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.00000	0.33913	0.55466	0.10621	0.00000	0.00000
6	0.05948	0.00653	0.00000	0.17954	0.05488	0.59808	0.03352	0.06797
7	0.00000	0.00000	0.00000	0.02594	0.00000	0.00000	0.97186	0.00220
8	0.00000	0.00000	0.00000	0.03679	0.00000	0.00000	0.02078	0.94243

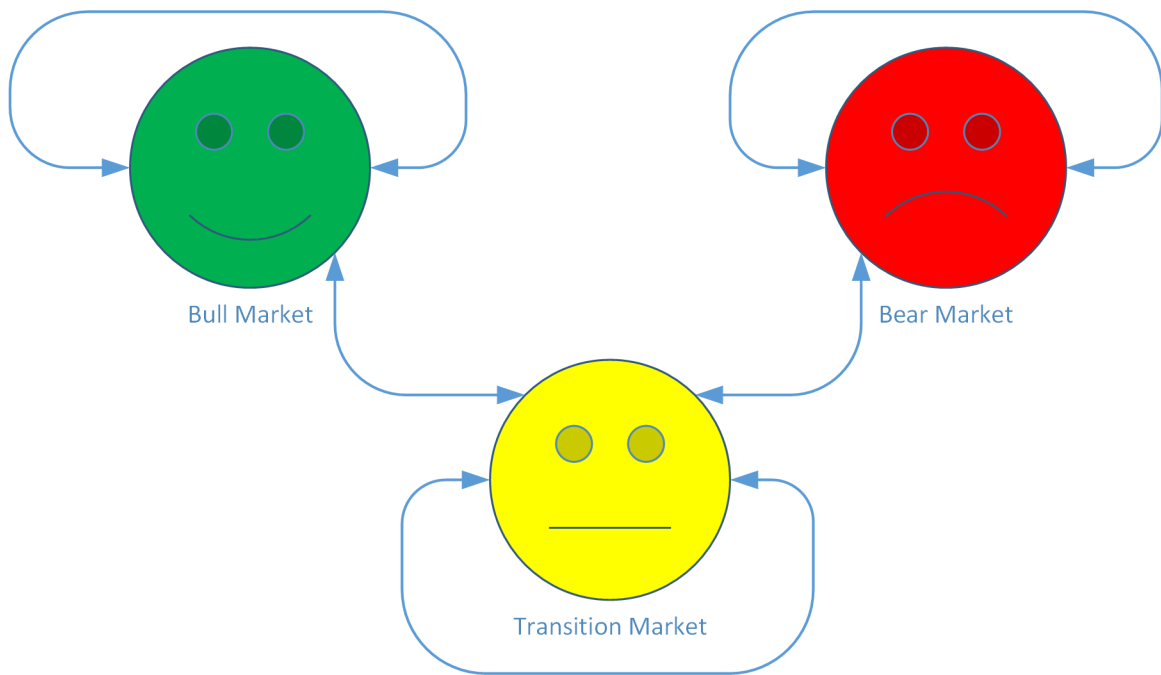
The eight states seem to be categorized into three groups:

- States 1, 2, and 3 belong to first group. Although there are very upward-moving days (state 2) and somewhat downward-moving days (state 3), the main trend is upward (state 1). The risk in this group is pretty low.
- States 4, 5, and 6 belong to second group. There are good days (state 4), bad days (state 5), and even very bad days (state 6). This group has medium risk.
- States 7 and 8 belong to the third group, where the risk is extremely high.

The transition probability matrix tells the more detailed story:

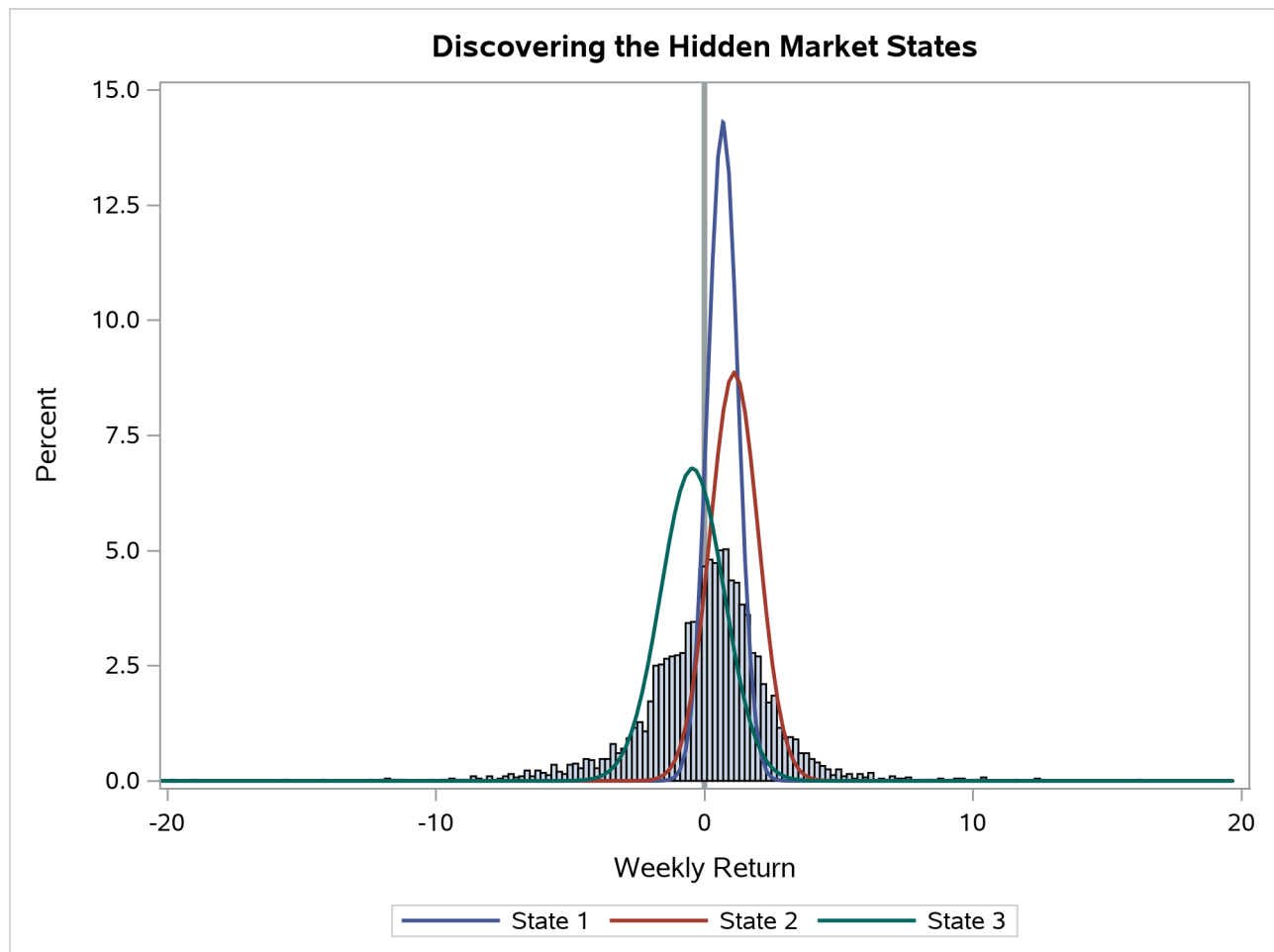
- The first group has a 98% chance of switching to itself and a 2% chance of switching to the second group.
- The third group has a 97% chance of switching to itself and a 3% chance of switching to the second group.
- The second group has a 94% chance of switching to itself and equal 3% chances of switching to the first and third groups.
- The first and third group have zero chance of switching to each other.
- When the market is in one group, it lasts for a long time before switching to the other groups.

Hence, according to the eight-state RS-AR(4) model, there is one more market state besides the well-known bear market (the first group) and bull market (the third group): the transition market state (the second group). [Figure 20.107](#) shows the relationship between these market states.

Figure 20.107 Market States and Their Transitions

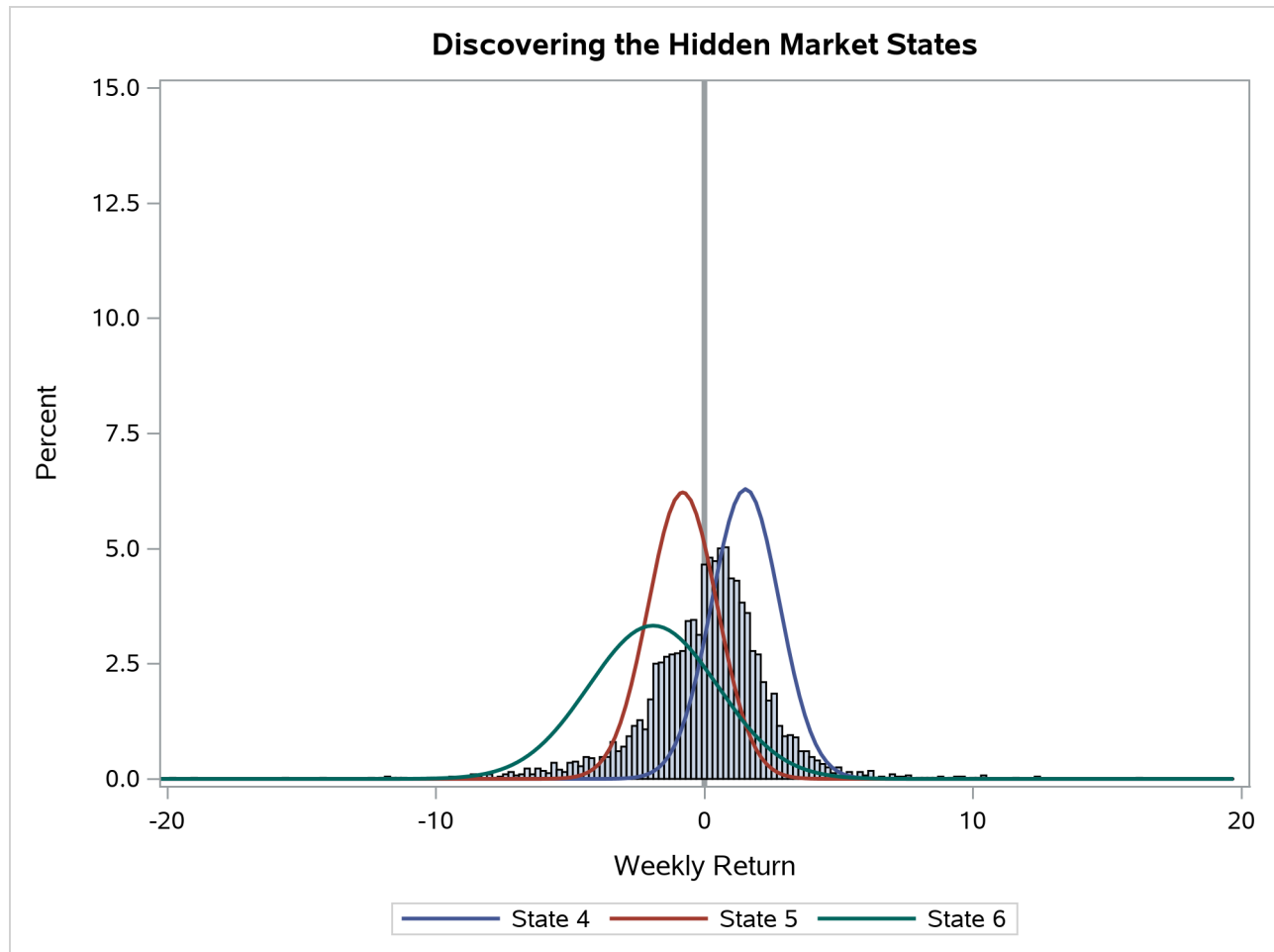
The following statement plots three states that correspond to the bull market state, which are shown in [Output 20.1.21](#):

```
%plotState(&ds.In, returnw, 1, 3);
```

Output 20.1.21 Three Gaussian Kernels for the Bull Market State

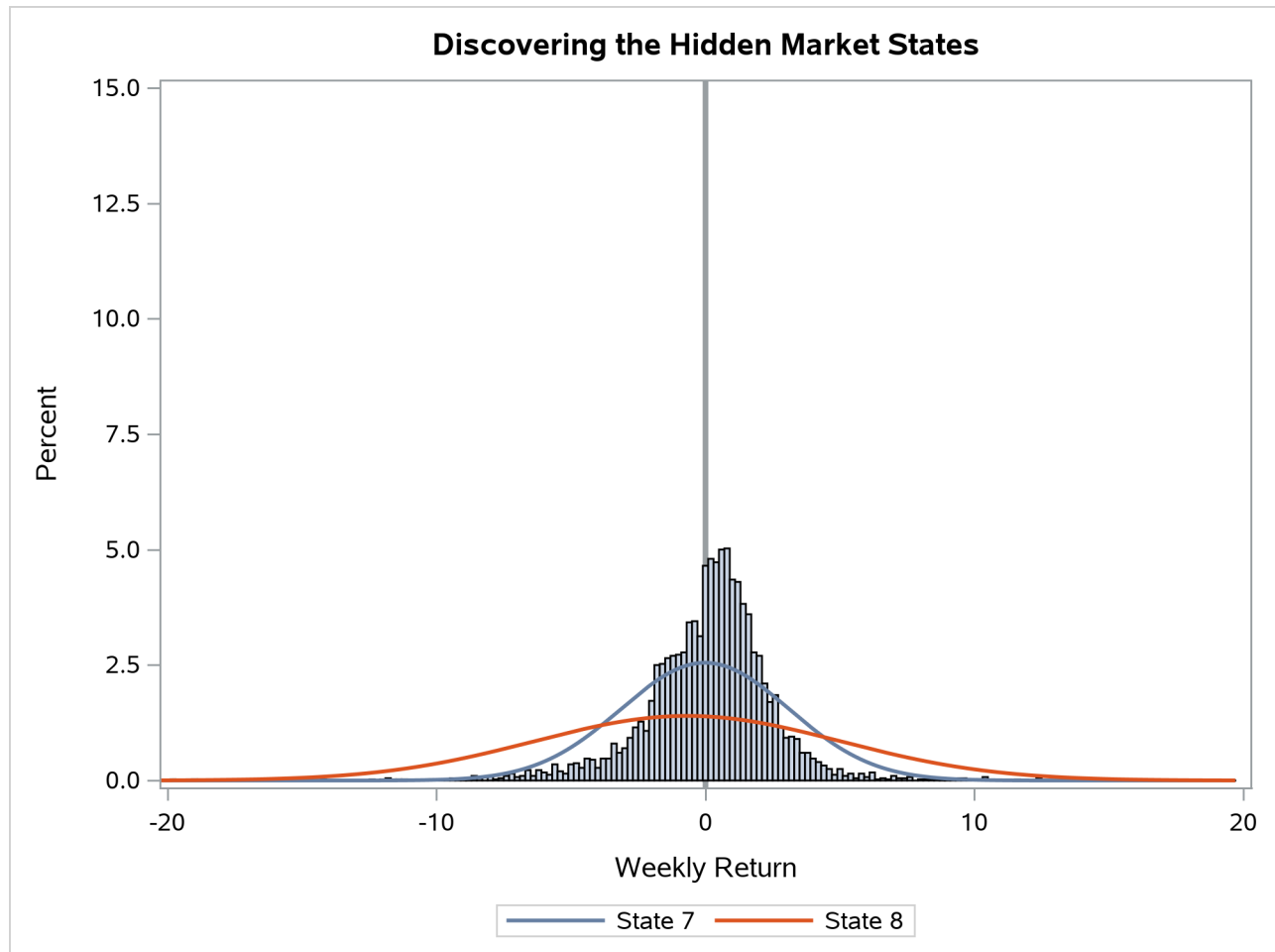
The following statement plots three states that correspond to the transition market state, which are shown in [Output 20.1.22](#):

```
%plotState(&ds.In,returnw,4,6);
```

Output 20.1.22 Three Gaussian Kernels for the Transition Market State

The following statement plots two states that correspond to the bear market state, which are shown in [Output 20.1.23](#):

```
%plotState(&ds.In,returnw,7,8);
```

Output 20.1.23 Two Gaussian Kernels for the Bear Market State

The following statements plot the decoded path of states according to the eight-state RS-AR(4) model; the result is shown in [Output 20.1.24](#).

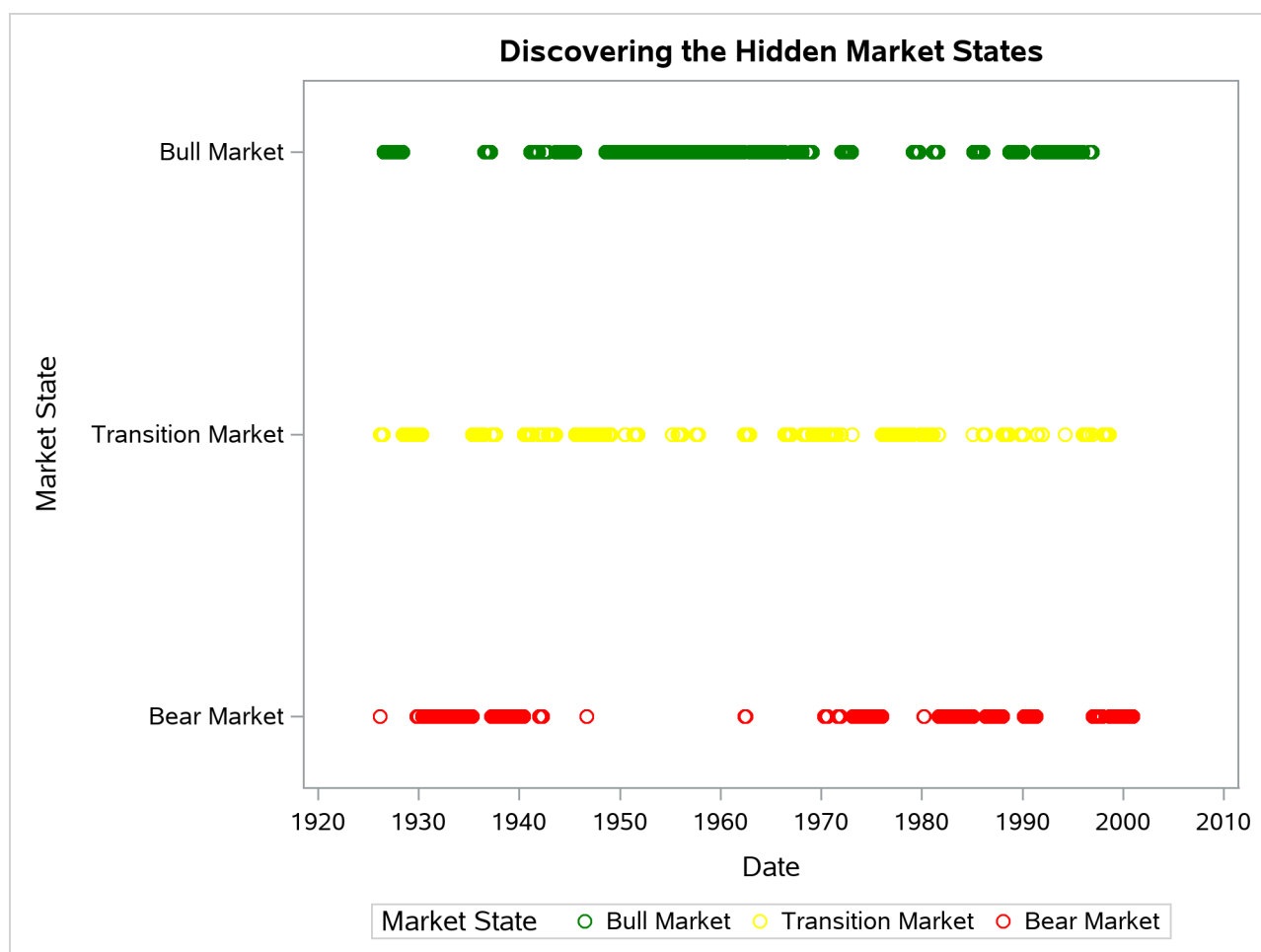
```
* plot decoded results;
data &ds.InDecode_k8to8_p4to4; set mylib.&ds.InDecode_k8to8_p4to4; run;
proc sort data=&ds.InDecode_k8to8_p4to4; by date; run;
data decodeMaketStates;
  set &ds.In(where=(date>='30JAN1926'd));
  set &ds.InDecode_k8to8_p4to4;
  array returnws returnw1-returnw3;
  array prices price1-price3;
  if(state~=.) then do;
    if(state=1 | state=2 | state=3) then do;
      returnws[1] = returnw; prices[1] = price;
      marketState="Bull Market    ";
    end;
    if(state=4 | state=5 | state=6) then do;
      returnws[2] = returnw; prices[2] = price;
      marketState="Transition Market";
    end;
  end;
```

```

    if (state=7 | state=8) then do;
        returnws[3] = returnw; prices[3] = price;
        marketState="Bear Market      ";
    end;
end;
label marketState="Market State";
run;
proc sgplot data=decodeMaketStates;
    styleattrs datacontrastcolors=(red yellow green);
    scatter x=date y=marketState / group=marketState nomissinggroup;
    yaxis label="Market State" offsetmin=0.1 offsetmax=0.1;
    keylegend / sortorder=reverseauto;
run;

```

Output 20.1.24 Decoded Market States



The following statements plot the decoded weekly returns, which are shown in [Output 20.1.25](#):

```

proc sgplot data=decodeMaketStates;
    series x=date y=returnw1 / break lineattrs=(color=green)
        legendlabel="Bull Market";
    series x=date y=returnw2 / break lineattrs=(color=yellow)
        legendlabel="Transition Market";

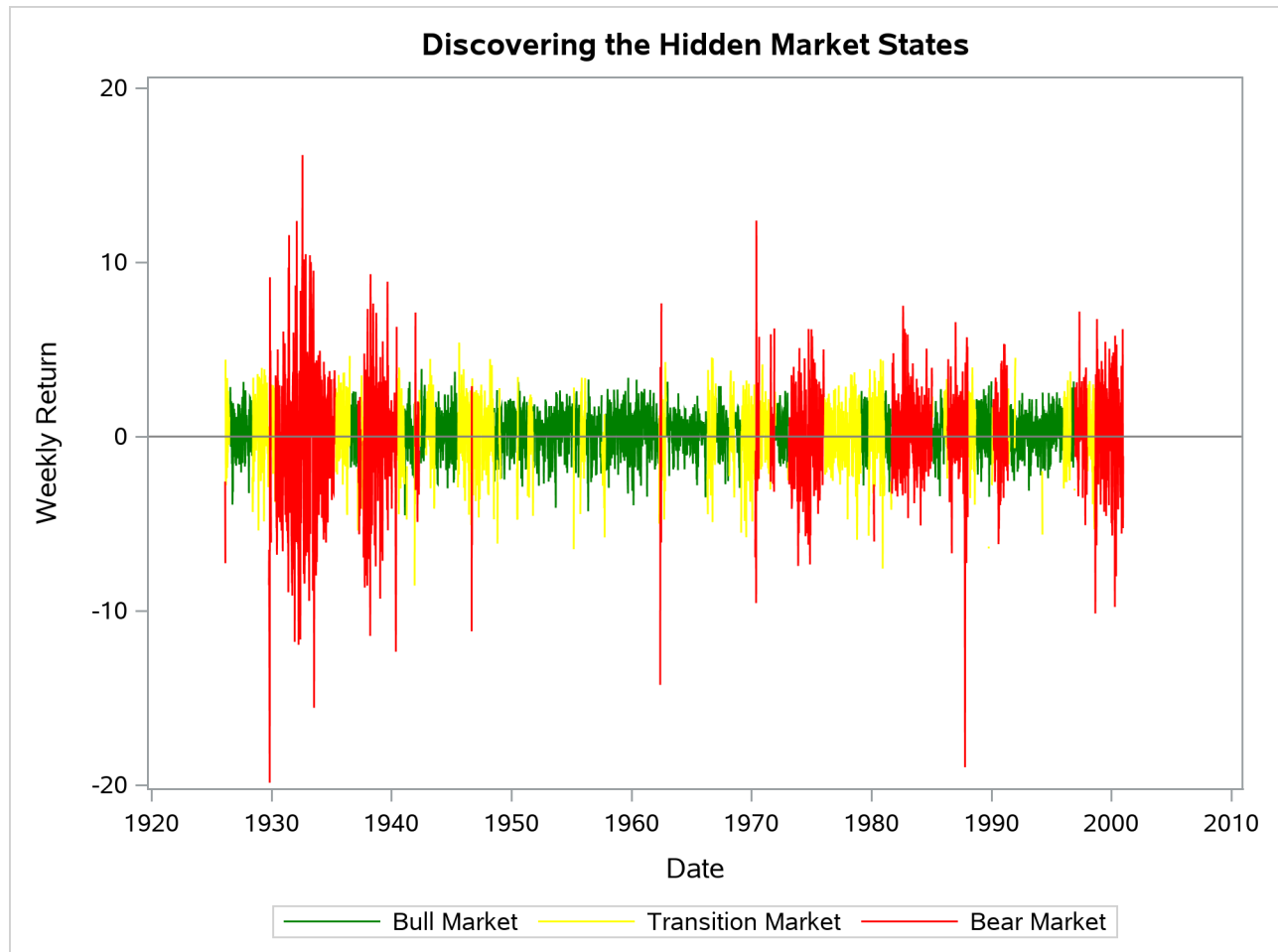
```

```

series x=date y=returnw3 / break lineattrs=(color=red)
                        legendlabel="Bear Market";
refline 0 / axis=y lineattrs=(color=gray thickness=1);
yaxis max=20 min=-20 label="Weekly Return";
run;

```

Output 20.1.25 Decoded Weekly Returns

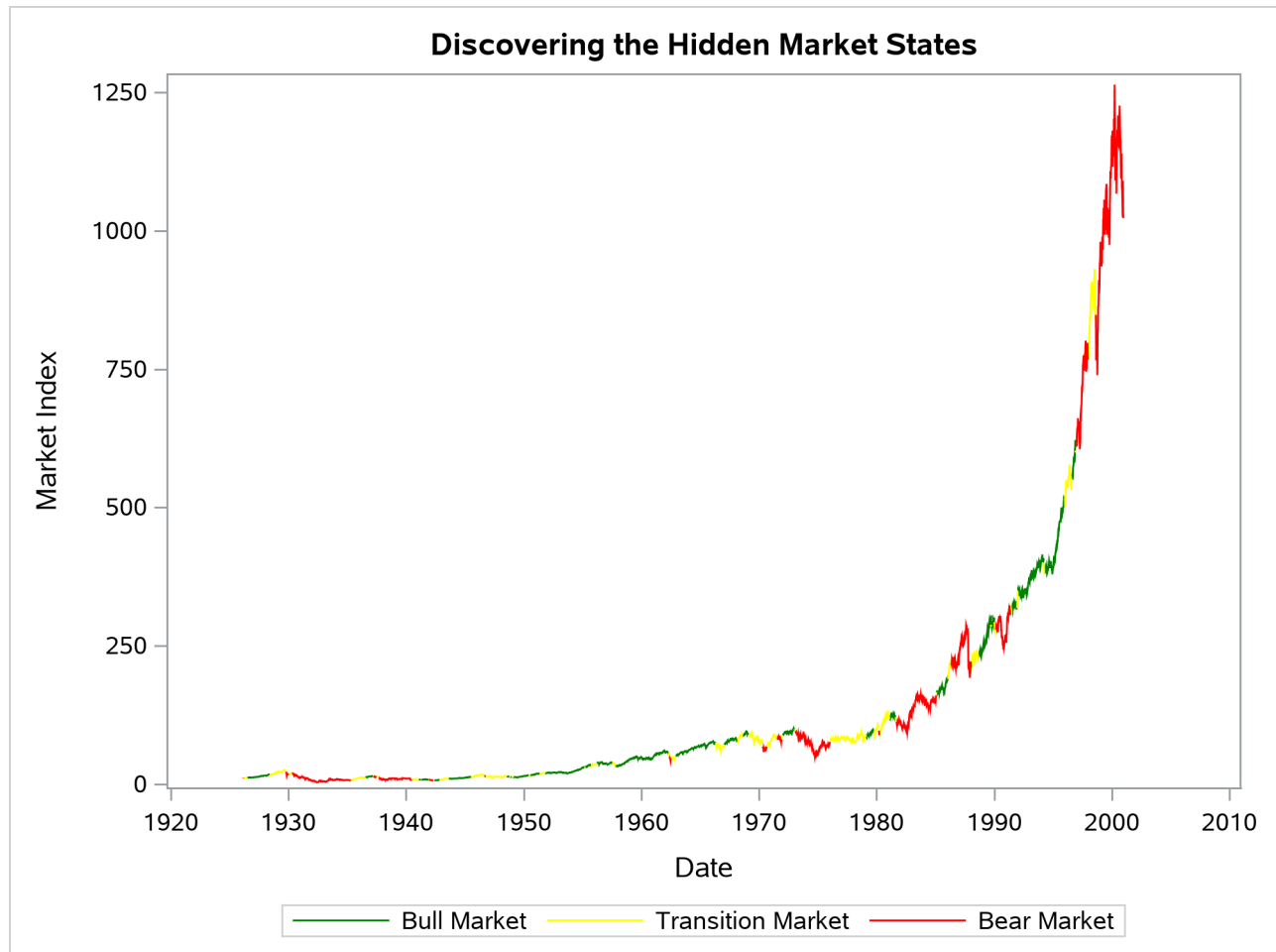


The following statements plot the decoded stock prices, which are shown in [Output 20.1.26](#):

```

proc sgplot data=decodeMaketStates;
  series x=date y=price1 / break lineattrs=(color=green)
                        legendlabel="Bull Market";
  series x=date y=price2 / break lineattrs=(color=yellow)
                        legendlabel="Transition Market";
  series x=date y=price3 / break lineattrs=(color=red)
                        legendlabel="Bear Market";
  yaxis label="Market Index";
run;

```


Output 20.1.26 Decoded Stock Prices

The Out-of-Sample Testing

In the model selection process, the AIC, but not the in-sample goodness of fit (that is, the log likelihood), is used as the criterion for selecting the best model because a bias-variance trade-off exists. In this section, three types of out-of-sample forecasts are tested to check whether the selected model has a good forecast performance.

The Value-at-Risk (VaR) Forecast

You can specify the INMODEL= option in the SCORE statement to forecast the VaRs at different levels, such as 1%, 5%, and 10%, as shown in [Output 20.1.27](#). The following macros can predict the VaRs for the out-of-sample period and evaluate the predictive performance by the likelihood ratio (LR) test for the unconditional coverage of the VaR forecast (Kuester, Mittnik, and Paolella 2006):

```
* print Predictive Performance of VaR Forecasts;
%let alpha1 = 0.80;
%let alpha2 = 0.90;
%let alpha3 = 0.98;
%macro forecastVaR(myLib, dsData, dsModel, timeId, varReturn,
    k, p, iStart, iEnd, oosStart, oosEnd,
```

```

dsForecastPrefix);
%do i = &iStart. %to &iEnd.;
  proc hmm data=&myLib..&dsData.;
    score inmodel=&myLib..&dsModel.;
    forecast out=&myLib..&dsForecastPrefix.&i. alpha=&&alpha&i. online;
  run;
  data &dsForecastPrefix.&i.;
    set &myLib..&dsForecastPrefix.&i.;
  run;
  proc sort data=&dsForecastPrefix.&i.; by &timeId.; run;
  data &dsForecastPrefix.&i.;
    set &dsForecastPrefix.&i. (FIRSTOBS=&oosStart. OBS=&oosEnd.
      keep=&timeId. &varReturn._Q1
      rename=(&varReturn._Q1=&varReturn._Q1_&k.&p.&i.));
    time=_N_; nState = &k; yLag = &p;
  run;
%end;
%mend forecastVaR;

%macro evaluateVaR(dsData,dsForecastPrefix,varReturn,k,p,iStart,iEnd,n,dsOut);
  data &dsData.out;
    set &dsData.;
    if (date>&cutDate.) then output;
  run;
  data forecastData;
    set &dsData.out;
    time=_N_;
  run;
  data forecastData;
    merge &dsForecastPrefix.: forecastData;
    by time;
  run;
  data &dsOut.;
    set forecastData;
    retain %do i = &iStart. %to &iEnd.; cviol&k.&p.&i. 0 %end; ;
    retain %do i = &iStart. %to &iEnd.; c&varReturn.&k.&p.&i. 0 %end; ;
    %do i = &iStart. %to &iEnd.;
      if &varReturn. le &varReturn._Q1_&k.&p.&i. then do;
        cviol&k.&p.&i.=cviol&k.&p.&i.+1;
      end;
      c&varReturn.&k.&p.&i. = c&varReturn.&k.&p.&i.
        + &varReturn._Q1_&k.&p.&i.;
    %end;
    if _N_ = &n. then do;
      %do i = &iStart. %to &iEnd.;
        Norminal = (1-&&alpha&i.)/2;
        Viol = cviol&k.&p.&i. / &n.;
        LR = 2*(cviol&k.&p.&i.*log(Viol)+(&n.-cviol&k.&p.&i.)*log(1-Viol)
          -(cviol&k.&p.&i.*log(Norminal)+(&n.-cviol&k.&p.&i.)
            *log(1-Norminal)));
        pValue = 1 - cdf("CHISQUARE", LR, 1);
        meanVaR = c&varReturn.&k.&p.&i. / &n.;
        output;
      %end;
    end;
  run;
%mend evaluateVaR;

```

```

end;
label Norminal='Target Prob.' Viol='Violation Ratio' LR='LR Stat.'
      pValue='Pr > ChiSq' meanVaR='Avg. of VaR' nState='Number of States'
      yLag='AR Lag';
keep nState yLag Norminal Viol LR pValue meanVaR;
run;
proc print data=&dsOut. noobs label; format Viol LR pValue meanVaR 6.4; run;
%mend evaluateVaR;

%macro VaR(k,p);
  %forecastVaR(myLib=mylib, dsData=&ds.,
    dsModel=&ds.inMLModel_k&k.To&k._p&p.To&p.,
    timeId=date, varReturn=returnw,
    k=&k., p=&p., iStart=1, iEnd=3,
    oosStart=%eval(3999-(5-&p.)), oosEnd=%eval(4854-1-(5-&p.)),
    dsForecastPrefix=&ds.Forecastk&k._p&p.);
  %evaluateVaR(dsData=&ds., dsForecastPrefix=&ds.Forecastk&k._p&p.,
    varReturn=returnw, k=&k., p=&p., iStart=1, iEnd=3, n=855,
    dsOut=VaR_outputk&k._p&p.);
%mend VaR;

%VaR(8,4);

```

Output 20.1.27 Predictive Performance of VaR Forecasts
Discovering the Hidden Market States

Number of AR States	AR Lag	Target Prob.	Violation Ratio	LR Stat.	Pr > ChiSq	Avg. of VaR
8	4	0.10	0.1053	0.2592	0.6107	-2.718
8	4	0.05	0.0538	0.2541	0.6142	-3.691
8	4	0.01	0.0129	0.6503	0.4200	-5.974

According to the p -values in the Pr > ChiSq column in [Output 20.1.27](#), at the 5% significance level, no tests can reject the null hypothesis that the number of violations is correct. Hence, the eight-state RS-AR(4) model has the correct unconditional coverage for the 1%, 5%, and 10% VaR forecasts.

The Out-of-Sample Log Likelihood

In fact, besides the good predictability of the tail of the distribution of weekly returns as shown in the VaR forecast analysis, the eight-state RS-AR(4) model also provides a very good prediction of the whole distribution of weekly returns, which can be seen by comparing the average weekly log likelihoods of the in-sample period and the out-of-sample period. As shown in [Output 20.1.28](#), compared to both the simplest model (the two-state RS-AR(0) model, which has the fewest parameters and the worst in-sample fit) and the most complex model (the nine-state RS-AR(5) model, which has the most parameters and the best in-sample fit), the eight-state RS-AR(4) model has the best out-of-sample forecast ability (that is, the largest average weekly log likelihood in the out-of-sample period).

```

* print in-sample and out-of-sample average weekly log likelihood;
proc hmm data=mylib.&ds.
  outstat=mylib.&ds.Stat_k2to9_p0to5;
  score inmodel=mylib.&ds.inmlmodel_k2to9_p0to5;

```

```

    evaluate out=mylib.&ds.Eval_k2to9_p0to5;
run;
data inLL;
    set mylib.&ds.Eval_k2to9_p0to5 (where=(date='26DEC2000'd)
        rename=(logLikelihood=isLL));
run;
data fullLL;
    set mylib.&ds.Eval_k2to9_p0to5 (where=(date='22DEC2017'd)
        rename=(logLikelihood=fullLL));
run;
data oosLL;
    merge inLL(in=a) fullLL(in=b);
    by modelIndex;
    oosLL = (fullLL - isLL)/855;
    if(a=b);
    drop date isLL fullLL;
run;

data inLL;
    set mylib.&ds.InMLStat_k2to9_p0to5;
    isLL = logLikelihood / (3999-5);
    keep modelIndex nState yLag isLL;
run;
proc sort data=inLL; by modelIndex; run;

data avgLL;
    merge inLL(in=a) oosLL(in=b);
    by modelIndex;
    if(a=b and (modelIndex=1 or modelIndex=41 or modelIndex=48));
    drop modelIndex;
run;

proc print data=avgLL noobs label;
    label nState='k' yLag='p' isLL='In-Sample' oosLL='Out-Of-Sample';
run;

```

Output 20.1.28 Comparison of Average Weekly Log Likelihoods**Discovering the Hidden Market States**

k	p	In-Sample	Out-Of-Sample
2	0	-2.12513	-2.14688
8	4	-2.05835	-2.10316
9	5	-2.05494	-2.12056

Trading Strategies and Portfolio Values

As mentioned at the beginning of this example, if you knew it would be a bull market, you might choose to buy the stock; if you knew it would be a bear market, you might choose to sell the stock. In this section, the market states are forecasted and then several trading strategies are compared.

The following statements forecast and plot the probability distribution of market states in [Output 20.1.29](#). The forecast of market states is in probabilities, which is similar to a weather forecast. For example, a forecast of

60% chance of bear market, 39% chance of transition market, and 1% chance of bull market is comparable to a forecast of 60% chance of raining, 39% chance of being cloudy, and 1% chance of being sunny.

```

proc hmm data=mylib.&ds.;
    score inmodel=mylib.&ds.InMLModel_k8To8_p4To4;
    forecast out=mylib.&ds.Forecast_k8to8_p4to4 online;
    display / excludeall;
run;

%let forecastStartDate = '26DEC2000'd;
%let forecastEndDate = '22DEC2017'd;
data ForecastProb;
    set mylib.&ds.Forecast_k8to8_p4to4
        (where=(date>=&forecastStartDate. and date<=&forecastEndDate.));
run;
proc sort data=ForecastProb; by date; run;
data ForecastProb;
    set ForecastProb(keep=state1 - state8) end=last;
    set &ds. (where=(date>=&forecastStartDate. and date<=&forecastEndDate.)
        keep=date price);
    * probability distribution of market states;
    bullProb = state1+state2+state3;
    transitionProb = state4+state5+state6;
    bearProb = state7+state8;
    bearTransitionProb = bearProb + transitionProb;
run;
* plot forecasted prob. dist. of market states and trading strategies;
%macro plotForecastAndTradingStrategies(ds, bearThreshold, bullThreshold,
    oneMinusBullThreshold);

proc sgplot data=&ds.;
    band x=date lower=0 upper=bearProb /
        fillattrs=(color=red) name="bear" legendlabel="Bear Market";
    band x=date lower=bearProb upper=bearTransitionProb /
        fillattrs=(color=yellow)
        name="transition" legendlabel="Transition Market";
    band x=date lower=bearTransitionProb upper=1 /
        fillattrs=(color=green) name="bull" legendlabel="Bull Market";
    series x=date y=price / y2axis lineattrs=(color=blue)
        name="index" legendlabel="Market Index";
    %if %length(&bearThreshold.)>0 %then %do;
        refline &bearThreshold. / axis=y lineattrs=(color=red thickness=3)
            label="Bear Threshold=&bearThreshold." labelloc=inside
            labelattrs=(weight=bold);
    %end;
    %if %length(&bearThreshold.)>0 %then %do;
        refline &oneMinusBullThreshold. / axis=y
            lineattrs=(color=green thickness=3)
            label="Bull Threshold=&bullThreshold." labelloc=inside
            labelattrs=(weight=bold);
    %end;
    xaxis values=("01JAN2001"d to "31DEC2017"d by year) display=(nolabel)
        tickvalueformat=year4. offsetmin=0.03 offsetmax=0.15;
    yaxis label="Forecasted Prob. Dist. of Market States";
    y2axis label="Market Index";

```

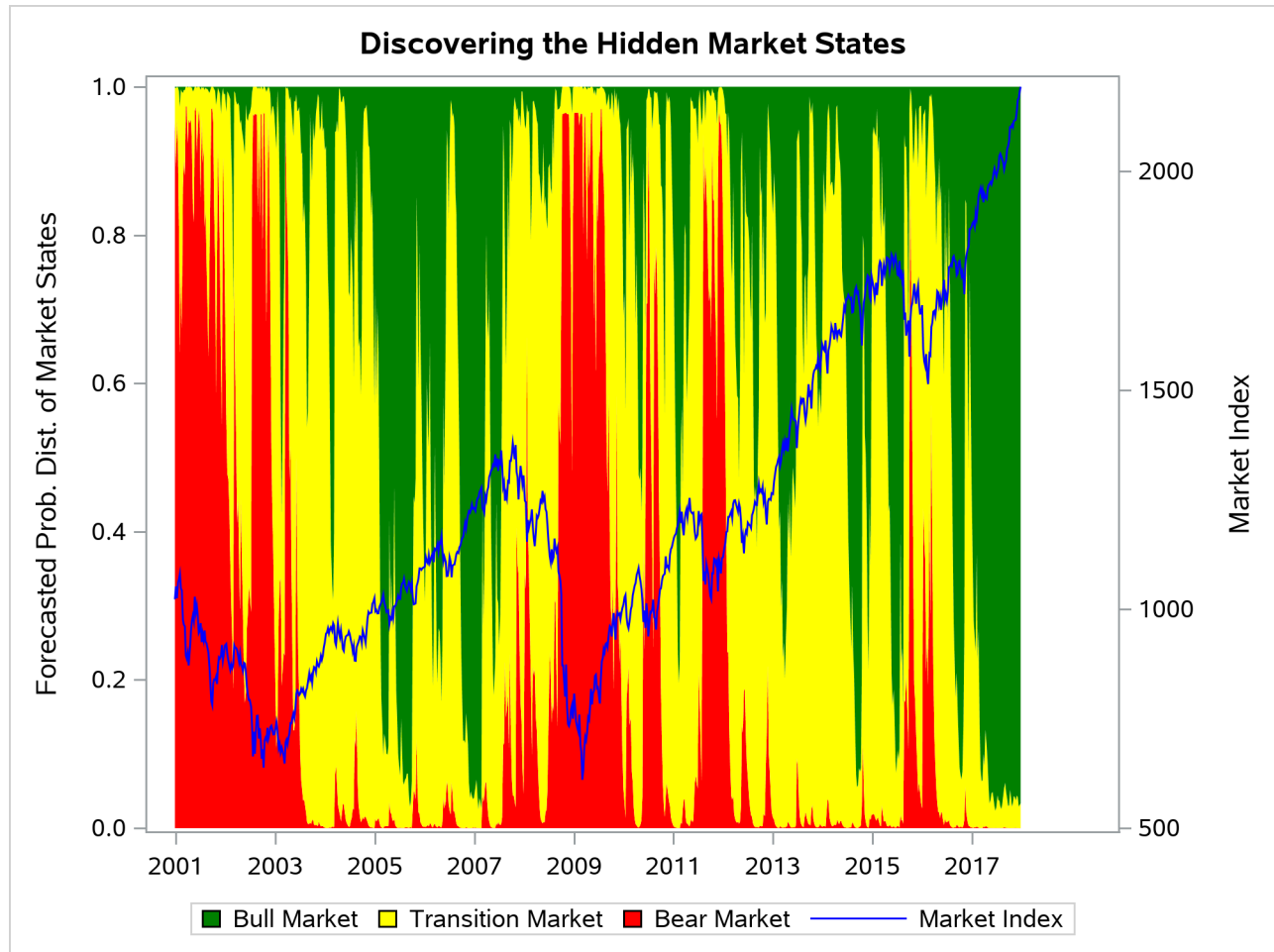
```

    keylegend "bull" "transition" "bear" "index"/
        location=outside position=bottom;
    run;
%mend;

%plotForecastAndTradingStrategies (ForecastProb);

```

Output 20.1.29 Forecasted Probabilities of Market States



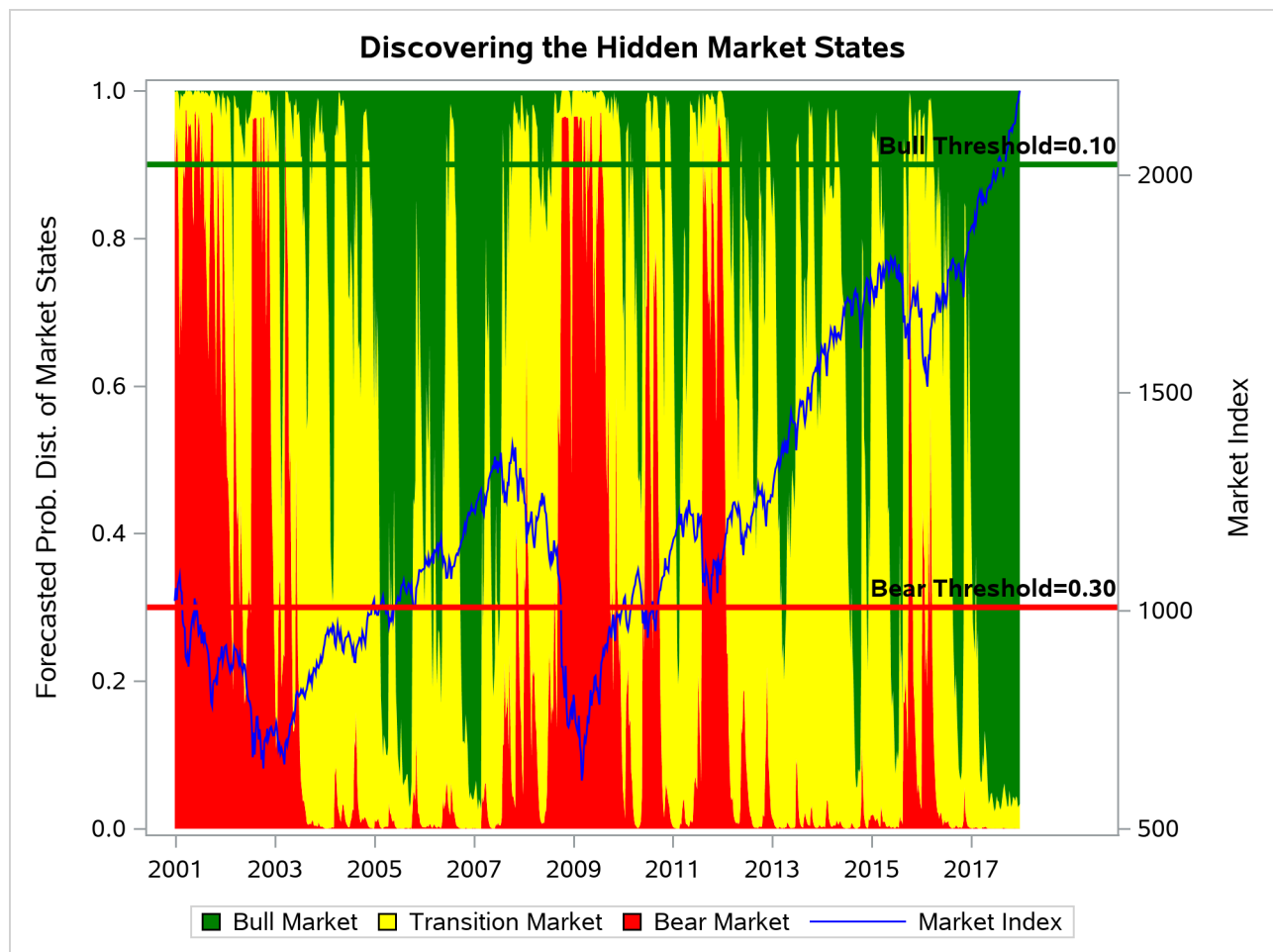
In this example, four simple trading strategies are considered:

- Risk-free strategy: hold the bond for the entire time period. This strategy does not depend on any forecast, and it is the first benchmark strategy.
- Market strategy: hold the stock for the entire time period. This strategy does not depend on any forecast, either. It reflects the market performance, and it is the second benchmark strategy.
- Bear strategy: sell the stock if the chance of a bear market in the next week is no less than 30%; otherwise, buy the stock.
- Bull strategy: buy the stock if the chance of a bull market in the next week is no less than 10%; otherwise, sell the stock.

The thresholds in bear and bull strategies, 30% and 10% respectively, are ad hoc, which reflect the investor's risk attitude. The trading strategy of switching between bond and stock can also be analogous to the decision of traveling with or without umbrella. The risk-free strategy is analogous to a decision to always travel with an umbrella; the market strategy is analogous to never traveling with umbrella; the bear strategy is analogous to traveling with an umbrella if the chance of raining tomorrow is no less than 30%; and the bull strategy is analogous to traveling without an umbrella if the chance of being sunny tomorrow is no less than 10%. The following statements plot the thresholds for the bear and bull trading strategies:

```
%let bearThreshold=0.30;
%let bullThreshold=0.10;
%let oneMinusBullThreshold=0.90;
%plotForecastAndTradingStrategies(ForecastProb, &bearThreshold.,
                                &bullThreshold., &oneMinusBullThreshold.);
```

Output 20.1.30 Thresholds for Bull and Bear Trading Strategies



The following statements calculate the portfolio values and plot the wealth curves of four trading strategies:

```
%let strategy1=RiskFree;
%let strategy2=Market;
%let strategy3=Bear;
%let strategy4=Bull;
```

```

%let strategy1Label=Risk Free;
%let strategy2Label=Market;
%let strategy3Label=Bear;
%let strategy4Label=Bull;
* assume that 1 year = 252 trading days = 50.4 weeks;
* weekly risk free rate =  $\exp(\ln(1+APY)/50.4)-1$ ;
%let riskFreeRate = 0.0002661; * APY=1.35%;
%let initialMoney=1;

%macro port_val_cal(stockPosition, stockPrice, riskFreeRate, strategy,
                    initialMoney, rowNo);
  if (&rowNo. = 1) then do;
    &strategy.CashValue = &initialMoney.-&initialMoney.*&stockPosition.;
    &strategy. = &initialMoney.;
    &strategy.StockAmount = &strategy.*&stockPosition./&stockPrice.;
    &strategy.CurrentCashValue = &strategy.CashValue;
  end;
  else do;
    &strategy.CurrentCashValue = &strategy.CashValue*(1+&riskFreeRate.);
    &strategy. = &strategy.CurrentCashValue
      + &strategy.StockAmount*&stockPrice.;
    &strategy.StockAmount = &strategy.*&stockPosition./&stockPrice.;
    &strategy.CashValue = &strategy.*(1-&stockPosition.);
  end;
  &strategy.ExcessReturn = &strategy./lag(&strategy.)-1-&riskFreeRate.;
  &strategy.ExcessReturn2 = &strategy.ExcessReturn**2;
%mend port_val_cal;

%macro port_val_gen(ForecastProb, bearThreshold, bullThreshold, riskFreeRate,
                    initialMoney, startDate, endDate);
  data PortValPath;
    set &ForecastProb. end=last;
    array stockPos[4] %do i = 1 %to 4; &&strategy&i..StockPos %end; ;
    retain
      %do i = 1 %to 4;
        &&strategy&i..CashValue 0 &&strategy&i..StockAmount 0
        &&strategy&i..CurrentCashValue 0 &&strategy&i..CumExcessReturn 0
        &&strategy&i..CumExcessReturn2 0
      %end; ;
    * stock position for each strategy;
    stockPos[1] = 0;
    stockPos[2] = 1;
    if(bearProb>=&bearThreshold.) then stockPos[3]=0;
    else stockPos[3]=1;
    if(bullProb>=&bullThreshold.) then stockPos[4]=1;
    else stockPos[4]=0;
    * wealth for each strategy;
    %do i = 1 %to 4;
      %port_val_cal(stockPosition=stockPos[&i.], stockPrice=price,
                    riskFreeRate=&riskFreeRate., strategy=&&strategy&i.,
                    initialMoney=&initialMoney., rowNo=_N_);
      &&strategy&i..CumExcessReturn+&&strategy&i..ExcessReturn;
      &&strategy&i..CumExcessReturn2+&&strategy&i..ExcessReturn2;
    %end;
  run;

```



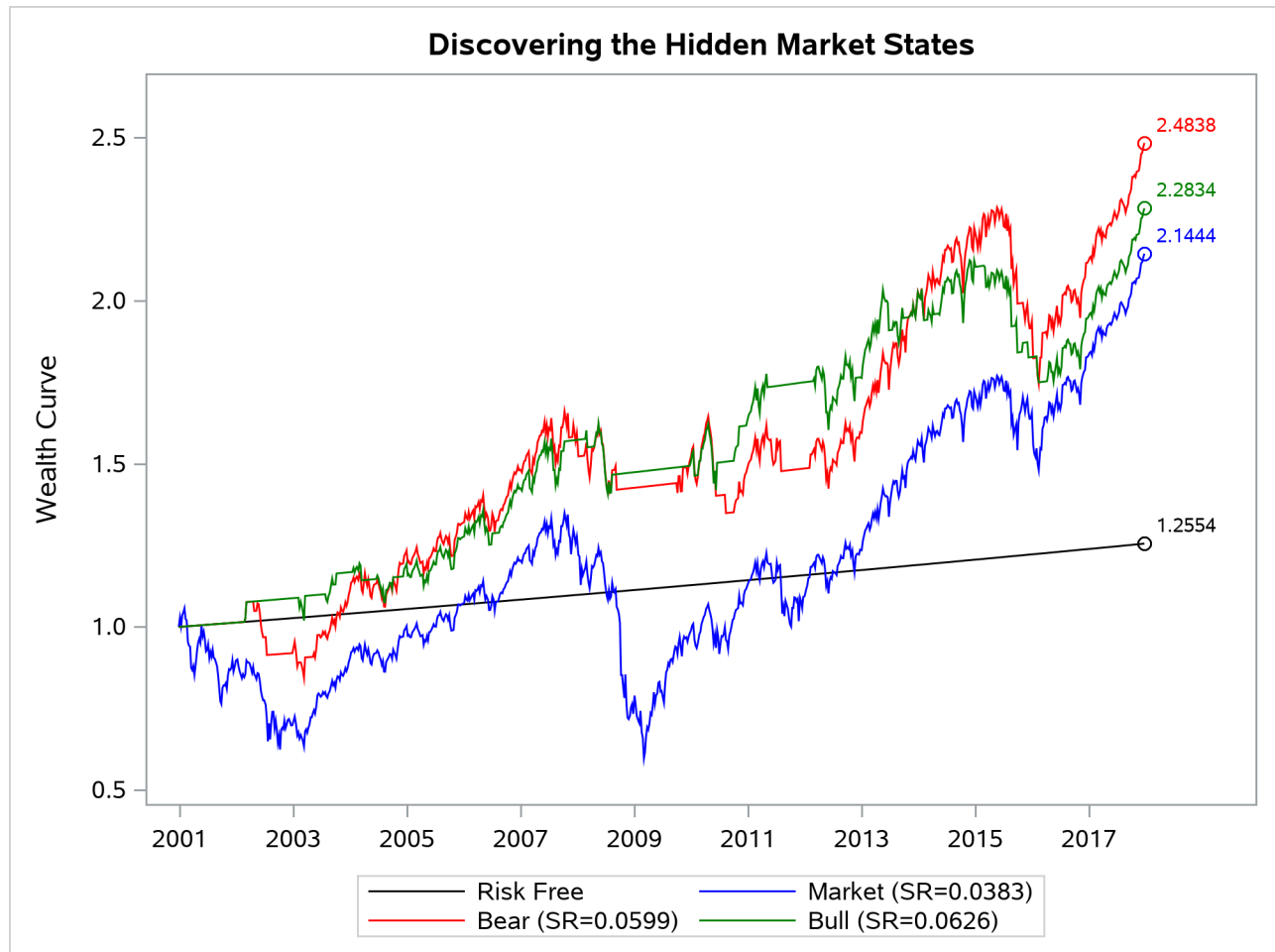
```

* Sharpe ratio and portfolio final wealth;
if last then do;
  &strategy1.LastPorVal = &strategy1.;
  %do i = 2 %to 4;
    &&strategy&i..LastPorVal = &&strategy&i..;
    &&strategy&i..ExcessReturnMean =
      &&strategy&i..CumExcessReturn/(_N_-1);
    &&strategy&i..ExcessReturnStd =
      sqrt(&&strategy&i..CumExcessReturn2/(_N_-1)
        - &&strategy&i..ExcessReturnMean**2);
    &&strategy&i..SharpeRatio = round(&&strategy&i..ExcessReturnMean
      /&&strategy&i..ExcessReturnStd, 0.0001);
    call symputx("&&strategy&i..SharpeRatio",
      &&strategy&i..SharpeRatio, 'G');
  %end;
end;
run;
%mend;

* plot portfolio values of different trading strategies;
%let color1=black;
%let color2=blue;
%let color3=red;
%let color4=green;
%macro perf_comp_plot_gen(nStrategies);
  proc sgplot data=PortValPath cycleattrs;
    %do i = 1 %to &nStrategies.;
      %let strategy=&&strategy&i..;
      series x=date y=&&strategy&i.. / name="&strategy."
        lineattrs=(color=&&color&i..)
        %if (&i.=1) %then %do;
          legendlabel="&&strategy&i..Label.";
        %end;
        %else %do;
          legendlabel="&&strategy&i..Label. (SR=&&&strategy.SharpeRatio.)";
        %end;
      scatter x=date y=&&strategy&i..LastPorVal / datalabelpos=topright
        markerattrs=(color=&&color&i..)
        datalabel=&&strategy&i..LastPorVal;
    %end;
    xaxis values=("01JAN2001"d to "31DEC2017"d by year) display=(nolabel)
      tickvalueformat=year4. offsetmin=0.03 offsetmax=0.15;
    yaxis max=2.6 min=0.5 label="Wealth Curve";
    keylegend %do i = 1 %to &nStrategies.; "&&strategy&i.." %end; /
      location=outside position=bottom;
  run;
%mend perf_comp_plot_gen;

%port_val_gen(ForecastProb=ForecastProb,
  bearThreshold=&bearThreshold., bullThreshold=&bullThreshold.,
  riskFreeRate=&riskFreeRate., initialMoney=&initialMoney.,
  startDate=&forecastStartDate., endDate=&forecastEndDate.);
%perf_comp_plot_gen(nStrategies=4);

```

Output 20.1.31 Wealth Curves of Four Trading Strategies

As shown in [Output 20.1.31](#), thanks to the forecast of market states by the eight-state RS-AR(4) model, both bear and bull strategies beat the market: compared to the market strategy, they increase the Sharpe ratio more than 50% and get higher portfolio values at the end of out-of-sample period.

Example 20.2: Clustering Time Series by Using the Gaussian Hidden Markov Model

Clustering time series is a very common method of analyzing cross-sectional time series data. For more information, see Liao (2005) and Rani and Sikka (2012) and references therein. Especially for methods that use hidden Markov models (HMMs), see Smyth (1997), Oates, Firoiu, and Cohen (1999), and Ghassempour, Girosi, and Maeder (2014).

This example illustrates an algorithm for quickly clustering time series. With prior knowledge of the number of clusters and the number of states for each cluster, the algorithm creates the initial clusters according to the distribution of the likelihoods on an arbitrary HMM. Then, one HMM on each cluster is estimated and the time series are moved between clusters according to their likelihoods on those HMMs. The process is repeated until the algorithm converges or the maximum number of iterations is reached (Liao 2005).

In this example, the data generating process (DGP) is the same one that is described in Smyth (1997). There are two bistate Gaussian HMMs. For the first Gaussian HMM, the parameters are

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}, \mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

For the second Gaussian HMM, the parameters are

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.4 & 0.6 \\ 0.6 & 0.4 \end{pmatrix}, \mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

The only difference between these two Gaussian HMMs is that they have different transition probability matrices, so the time series that are generated from the first HMM have slower dynamics than those from the second HMM. Hence, the clustering time series algorithm should assign the time series that are generated from the first HMM to a cluster, and assign those from the second HMM to another cluster, if there is prior knowledge that the number of clusters is two and the number of states for each cluster is two.

The following macro generates the cross-sectional time series data table for analysis:

```

title "Clustering Time Series";

* create a data table dgpTable
consisting of nSections sections and T observations for each section;
* then copy the data table dgpTable to the data table trainingTable
and the data table libTrainingTable;
%macro createClustersTable(nSections,T,
                                dgpTable,trainingTable,libTrainingTable);

    %let seed = 12345;
    %let nClustersDGP = 2;
    %let clportion = 0.5;

    %let clpi1 = 0.5;
    %let cla11 = 0.6;
    %let cla22 = 0.6;
    %let clmu1 = 0;
    %let clsigma1 = 1;
    %let clmu2 = 3;
    %let clsigma2 = 1;

    %let c2pi1 = 0.5;
    %let c2a11 = 0.4;
    %let c2a22 = 0.4;
    %let c2mu1 = 0;
    %let c2sigma1 = 1;
    %let c2mu2 = 3;
    %let c2sigma2 = 1;
    data &dgpTable.;

```

```

do sec = 1 to &nSections.;
  * which cluster;
  * (1) randomly assign clusters;
  * u = uniform(&seed.);
  * if(u<=&clportion.) then cluster = 1;
  * else cluster = 2;
  * (2) all time series for one cluster are in one block;
  if(sec/&nSections.<=&clportion.) then cluster = 1;
  else cluster = 2;
  %do i = 1 %to &nClustersDGP.;
    if(cluster=&i.) then do;
      do t = 1 to &T.;
        if(t=1) then p = &c&i.p1;
        else do;
          if(lagState=1) then p = &c&i.a1;
          else p = 1-&c&i.a2;
        end;
        u = uniform(&seed.);
        if(u<=p) then state = 1;
        else state = 2;
        if(state=1) then
          y = &c&i.mu1. + sqrt(&c&i.sigma1.)*normal(&seed.);
        else
          y = &c&i.mu2. + sqrt(&c&i.sigma2.)*normal(&seed.);
        output;
        lagState = state;
      end;
    end;
  %end;
end;
run;

* trainingTable is sorted by sec and t;
data &trainingTable.;
  set &dgpTable.;
  keep sec t y;
run;

data &libTrainingTable.;
  set &trainingTable.;
run;
%mend;

```

The following macro initializes the clusters:

```

%macro estSection(libTrainingTable,casSectionTable,selectedSec,i,T,
                  evalTable,libModelTable);
  data &casSectionTable.&selectedSec.;
    set &libTrainingTable.;
    if(sec=&selectedSec.);
  run;

  proc hmm data=&casSectionTable.&selectedSec.;
    id section=sec time=t;

```

```

    model y / method=ml type=gaussian nstate=&k.;
    optimize ALGORITHM=interiorpoint printlevel=3 printIterFreq=1;
    score outmodel=&libModelTable.&i.;
run;

proc hmm data=&libTrainingTable.;
    evaluate out=mylib.tempEval;
    score inmodel=&libModelTable.&i.;
run;

data &evalTable.&i.;
    set mylib.tempEval;
    logLikelihood&i. = logLikelihood;
    if(t=&T.) then output;
    keep sec logLikelihood&i.;
run;

proc sort data=&evalTable.&i.; by sec; run;
%mend;

%macro assignSeriesToClusters(evalTable,nClusters);
    data &evalTable.;
        merge &evalTable.1 - &evalTable.&nClusters.;
        by sec;
        retain sumLogLikelihood 0;
        array arr[*] logLikelihood1 - logLikelihood&nClusters.;
        sumLogLikelihood = sumLogLikelihood + max(of arr[*]);
        pCluster = whichn(max(of arr[*]), of arr[*]);
        keep sec pCluster sumLogLikelihood;
    run;
%mend;

* initialize clusters;
* each section has T observations;
%macro ctsInitClusters(libTrainingTable,nClusters,nSections,T,
                      evalTable,libModelTable);
    * as a start, use first section or randomly select a section;
    %let selectedSec = 1;
    %let i = 0;
    %estSection(&libTrainingTable.,mylib.tempSection,
                &selectedSec.,&i.,&T.,
                &evalTable.,&libModelTable.);
    proc sort data=&evalTable.&i.; by descending logLikelihood&i.; run;

    %do i = 1 %to &nClusters.;
        data _null_;
            set &evalTable.0;
            pos = floor(1+(&nSections.-1)*(&i.-1)/(&nClusters.-1)+0.5);
            if(_N_=pos) then
                call symput('selectedSec',trim(left(put(sec,12)))));
        run;
        %estSection(&libTrainingTable.,mylib.tempSection,
                    &selectedSec.,&i.,&T.,
                    &evalTable.,&libModelTable.);
    %end;

```

```

%end;

%assignSeriesToClusters(&evalTable., &nClusters.);
%mend;

```

The following macro repeatedly moves the time series between clusters until no movement is necessary:

```

* estimate HMM for cluster i;
* each section has T observations;
%macro estCluster(libClusteredTable, libTrainingTable, i, T,
                  evalTable, libModelTable);
    data &libClusteredTable.&i.;
    set &libClusteredTable.;
    if(pCluster=&i.);
    run;

    proc hmm data=&libClusteredTable.&i.;
        id section=sec time=t;
        model y / method=ml type=gaussian nstate=&k.;
        optimize ALGORITHM=interiorpoint printlevel=3 printIterFreq=1;
        score outmodel=&libModelTable.&i.;
    run;

    proc hmm data=&libTrainingTable.;
        evaluate out=mylib.tempEval;
        score inmodel=&libModelTable.&i.;
    run;

    data &evalTable.&i.;
        set mylib.tempEval;
        logLikelihood&i. = logLikelihood;
        if(t=&T.) then output;
        keep sec logLikelihood&i.;
    run;

    proc sort data=&evalTable.&i.; by sec; run;
%mend;

* moves time series between clusters based on their likelihoods;
* each section has T observations;
* (1) &libModelTable.&i. has the model estimates for cluster i,
* and (2) &evalTable. has the clustering result;
%macro ctsClustering(trainingTable, libTrainingTable, nClusters, T,
                    libClusteredTable, evalTable, libModelTable);
    * save last clustering result;
    data prev&evalTable.; set &evalTable.; run;

    * clustering data;
    data &libClusteredTable.;
        merge &trainingTable. (in=a) &evalTable. (in=b);
        by sec;
    run;

    %do i = 1 %to &nClusters.;

```

```

    %estCluster(&libClusteredTable., &libTrainingTable., &i., &T.,
               &evalTable., &libModelTable.);
%end;

%assignSeriesToClusters(&evalTable., &nClusters.);

* compare current and previous clustering results;
proc compare data=&evalTable. compare=prev&evalTable.; run;
%let converged=%eval(&sysinfo.=0);
%let nIter=%eval(&nIter.+1);
%let done=%eval(&converged. or &nIter.>=&maxIter.);

%put nIter      = &nIter.;
%put converged = &converged.;
%put done      = &done.;
%mend;

* cluster time series;
* each section has T observations;
%macro ctsLoopClustering(trainingTable, libTrainingTable, nClusters, T,
                        libClusteredTable, evalTable, libModelTable);
%let done = 0;      * terminate clustering;
%do %while(&done.=0);
    %ctsClustering(&trainingTable., &libTrainingTable., &nClusters., &T.,
                  &libClusteredTable., &evalTable., &libModelTable.);
%end;
%mend;

```

Because the DGP is known, you can use the following macro to evaluate the performance of the clustering algorithm:

```

* compare the clustering results with the true clusters in the DGP;
* it is applicable only when the DGP is available;
%macro evaluateClusteringResults(dgpTable, evalTable, nSections, T);
    data true&evalTable.;
        set &dgpTable.;
        if(t=&T.);
            keep sec cluster;
    run;

    data ctsCheck;
        merge true&evalTable. (in=a) &evalTable. (in=b);
        by sec;
        if(cluster=pCluster) then do; correct=1; end;
        else do; correct=0; end;
        if(a=b);
    run;

    data ctsAccuracy;
        set ctsCheck;
        retain count 0 correctCount 0;
        count = count + 1;
        correctCount = correctCount + correct;
        if(count>&nSections.-0.5) then do;

```

```

        accuracy = correctCount / count;
        if (accuracy < 0.5) then accuracy = 1 - accuracy;
        nIterations = &nIter.;
        converged = &converged.;
        output;
    end;
    keep nIterations converged accuracy;
run;

proc print data = ctsAccuracy noobs;
    var nIterations converged accuracy;
    format accuracy 6.4;
run;
%mend;

```

As data become easy to access, the number of sections might become large. In this example, the number of sections is set to 10,000. As a start, the sample size for each time series is set to 200, which is the same as the sample size in Smyth (1997) and Oates, Firoiu, and Cohen (1999). In fact, the time series might have different lengths; if so, you should replace the log likelihood with the log likelihood per observation when you compare the time series.

The following macro calls simulate 10,000-section time series, apply the clustering algorithm, and then evaluate its performance:

```

%let N = 10000;
%let T = 200;
%createClustersTable(&N., &T., ctsDGP, ctsTrain, mylib.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;           * number of states in the HMM for each cluster;
%ctsInitClusters(mylib.ctsTrain, &nClusters., &N., &T.,
    evalCluster, mylib.modelCluster);
%let maxIter = 100; * maximum number of iterations;
%let nIter = 0;     * number of iterations;
%let converged = 0; * converge status;
%ctsLoopClustering(ctsTrain, mylib.ctsTrain, &nClusters., &T.,
    mylib.ctsTrainClustered, evalCluster, mylib.modelCluster);
%evaluateClusteringResults(ctsDGP, evalCluster, &N., &T.);

```

As shown in [Output 20.2.1](#), the accuracy of the clustering algorithm is very high.

Output 20.2.1 Clustering Accuracy ($N = 10,000$, $T = 200$)

Clustering Time Series

nIterations	converged	accuracy
5	1	0.9868

The following statements display the parameter estimates for cluster 1 and cluster 2 in [Output 20.2.2](#) and [Output 20.2.3](#), respectively:

```

* print parameter estimates for cluster 1;
proc hmm data=mylib.ctsTrain;
    score inmodel=mylib.modelCluster1;
run;

```



```

* print parameter estimates for cluster 2;
proc hmm data=mylib.ctsTrain;
  score inmodel=mylib.modelCluster2;
run;

```

As shown in [Output 20.2.2](#) and [Output 20.2.3](#), the parameter estimates of the Gaussian HMMs for the two clusters are very close to the true parameters in the DGP.

Output 20.2.2 Parameter Estimates for Cluster 1 ($N = 10,000$, $T = 200$)

Clustering Time Series

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.598877	0.000977	612.98	<.0001
TPM1_2	0.401123	0.000977	410.57	<.0001
TPM2_1	0.399884	0.000979	408.35	<.0001
TPM2_2	0.600116	0.000979	612.82	<.0001
MU1_1	-0.000368	0.002626	-0.14	0.8885
MU2_1	2.997355	0.002636	1137.11	<.0001
SIGMA1_1_1	0.994514	0.003457	287.69	<.0001
SIGMA2_1_1	1.002381	0.003487	287.43	<.0001

Output 20.2.3 Parameter Estimates for Cluster 2 ($N = 10,000$, $T = 200$)

Clustering Time Series

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.399589	0.001164	343.20	<.0001
TPM1_2	0.600411	0.001164	515.69	<.0001
TPM2_1	0.600027	0.001161	516.61	<.0001
TPM2_2	0.399973	0.001161	344.37	<.0001
MU1_1	-0.000618	0.002597	-0.24	0.8119
MU2_1	2.999476	0.002603	1152.48	<.0001
SIGMA1_1_1	0.995491	0.003433	290.00	<.0001
SIGMA2_1_1	0.999145	0.003447	289.87	<.0001

In the second scenario, the sample size of one time series is increased to 400:

```

%let N = 10000;
%let T = 400;
%createClustersTable(&N., &T., ctsDGP, ctsTrain, mylib.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;           * number of states in the HMM for each cluster;
%ctsInitClusters(mylib.ctsTrain, &nClusters., &N., &T.,
  evalCluster, mylib.modelCluster);
%let maxIter = 100; * maximum number of iterations;
%let nIter = 0;    * number of iterations;

```

```
%let converged = 0; * converge status;
%ctsLoopClustering(ctsTrain,mylib.ctsTrain,&nClusters.,&T.,
  mylib.ctsTrainClustered,evalCluster,mylib.modelCluster);
%evaluateClusteringResults(ctsDGP,evalCluster,&N.,&T.);
```

As the sample size increases, more information becomes available for each time series. As expected, the accuracy increases (to almost 100% in this case). [Output 20.2.4](#) shows the accuracy.

Output 20.2.4 Clustering Accuracy ($N = 10,000$, $T = 400$)

Clustering Time Series

nIterations	converged	accuracy
4	1	0.9989

The following statements print the parameter estimates for both clusters, which also become closer to the true parameter values, as shown in [Output 20.2.5](#) and [Output 20.2.6](#):

```
* print parameter estimates for cluster 1;
proc hmm data=mylib.ctsTrain;
  score inmodel=mylib.modelCluster1;
run;

* print parameter estimates for cluster 2;
proc hmm data=mylib.ctsTrain;
  score inmodel=mylib.modelCluster2;
run;
```

Output 20.2.5 Parameter Estimates for Cluster 1 ($N = 10,000$, $T = 400$)

Clustering Time Series

Parameter Estimates				
Parameter	Standard		t Value	Pr > t
	Estimate	Error		
TPM1_1	0.599184	0.000689	869.96	<.0001
TPM1_2	0.400816	0.000689	581.95	<.0001
TPM2_1	0.400637	0.000690	580.78	<.0001
TPM2_2	0.599363	0.000690	868.85	<.0001
MU1_1	-0.000181	0.001851	-0.10	0.9219
MU2_1	2.999973	0.001855	1617.37	<.0001
SIGMA1_1_1	0.996500	0.002443	407.90	<.0001
SIGMA2_1_1	0.999350	0.002452	407.56	<.0001

Output 20.2.6 Parameter Estimates for Cluster 2 ($N = 10,000$, $T = 400$)**Clustering Time Series**

Parameter	Parameter Estimates			
	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.399302	0.000827	482.79	<.0001
TPM1_2	0.600698	0.000827	726.30	<.0001
TPM2_1	0.600445	0.000827	725.85	<.0001
TPM2_2	0.399555	0.000827	483.00	<.0001
MU1_1	2.998405	0.001852	1619.15	<.0001
MU2_1	-0.000477	0.001852	-0.26	0.7966
SIGMA1_1_1	1.001306	0.002449	408.85	<.0001
SIGMA2_1_1	1.001528	0.002449	408.92	<.0001

In the last scenario, the sample size for each time series decreases to 20 (see the following statements), which is a very small number. This case is common when the sampling frequency cannot be high (for example, yearly tax return data from millions of families). The information in one time series is very limited; this might even hinder the estimation of the hidden Markov models. However, as shown in this example, by clustering the time series, you condense the common information in time series, and then the true DGP is recovered.

```
%let N = 10000;
%let T = 20;
%createClustersTable(&N., &T., ctsDGP, ctsTrain, mylib.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;           * number of states in the HMM for each cluster;
%ctsInitClusters(mylib.ctsTrain, &nClusters., &N., &T.,
    evalCluster, mylib.modelCluster);
%let maxIter = 100; * maximum number of iterations;
%let nIter = 0;    * number of iterations;
%let converged = 0; * converge status;
%ctsLoopClustering(ctsTrain, mylib.ctsTrain, &nClusters., &T.,
    mylib.ctsTrainClustered, evalCluster, mylib.modelCluster);
%evaluateClusteringResults(ctsDGP, evalCluster, &N., &T.);
```

As expected and as shown in [Output 20.2.7](#), the algorithm needs many more iterations to converge, and the accuracy of the clustering algorithm is not high.

Output 20.2.7 Clustering Accuracy ($N = 10,000$, $T = 20$)**Clustering Time Series**

nIterations	converged	accuracy
26	1	0.7552

The following statements print the parameter estimates for both clusters, which are shown in [Output 20.2.8](#) and [Output 20.2.9](#). Although each time series might contain little information, the parameter estimates of the Gaussian HMMs for the two clusters are very close to the true parameters in the DGP.

```

* print parameter estimates for cluster 1;
proc hmm data=mylib.ctsTrain;
  score inmodel=mylib.modelCluster1;
run;

* print parameter estimates for cluster 2;
proc hmm data=mylib.ctsTrain;
  score inmodel=mylib.modelCluster2;
run;

```

Output 20.2.8 Parameter Estimates for Cluster 1 ($N = 10,000$, $T = 20$)**Clustering Time Series**

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.357167	0.003560	100.32	<.0001
TPM1_2	0.642833	0.003560	180.56	<.0001
TPM2_1	0.638448	0.003510	181.91	<.0001
TPM2_2	0.361552	0.003510	103.02	<.0001
MU1_1	-0.002996	0.007409	-0.40	0.6860
MU2_1	2.989537	0.007619	392.36	<.0001
SIGMA1_1_1	0.965620	0.009808	98.45	<.0001
SIGMA2_1_1	1.011633	0.010366	97.59	<.0001

Output 20.2.9 Parameter Estimates for Cluster 2 ($N = 10,000$, $T = 20$)**Clustering Time Series**

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.638344	0.002875	222.00	<.0001
TPM1_2	0.361656	0.002875	125.78	<.0001
TPM2_1	0.365361	0.002894	126.23	<.0001
TPM2_2	0.634639	0.002894	219.26	<.0001
MU1_1	3.010552	0.007647	393.69	<.0001
MU2_1	0.011987	0.007950	1.51	0.1316
SIGMA1_1_1	0.981346	0.010152	96.66	<.0001
SIGMA2_1_1	1.029845	0.010807	95.30	<.0001

Example 20.3: Analysis of the Business Cycle by Using the Regime-Switching Mean-Adjusted Autoregression Model

This example revisits the model in Hamilton (1989) to illustrate how the regime-switching mean-adjusted autoregression model is used to estimate the business cycle with quarterly US real gross national product (GNP) data from 1951 to 1984.

The following statements import the data and calculate the GNP growth rate (in percentage) to which the model applies:

```

title 'Analysis of the Business Cycle';
data gnpHamilton;
    input date: anydtdte. gnp;
    format date yyq6.;
    dgnp = 100*(log(gnp)-log(lag(gnp)));
    if cmiss(of _all_) then delete;
    label date="Date" gnp="GNP" dgnp="Growth Rate of GNP";
datalines;
1951q1  1286.6
1951q2  1320.4
1951q3  1349.8

    ... more lines ...

1982q4  3159.3
1983q1  3190.6
1983q2  3259.3
1983q3  3303.4
1983q4  3357.2
1984q1  3449.4
1984q2  3492.6
1984q3  3510.4
1984q4  3515.6
;

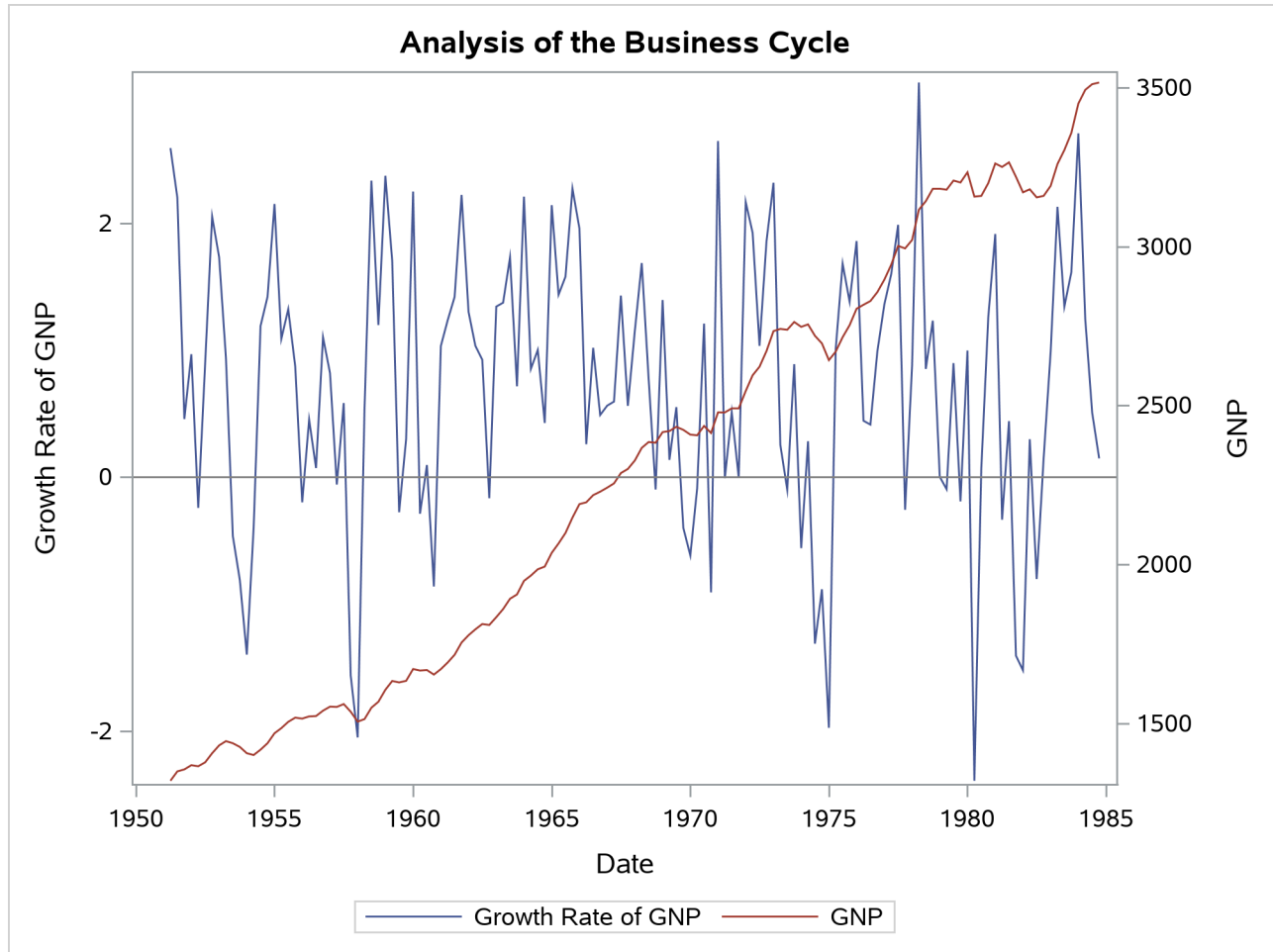
```

The following statements plot the data in [Output 20.3.1](#):

```

proc sgplot data=gnpHamilton;
    series x=date y=dgnp;
    series x=date y=gnp / y2axis;
    refline 0 / axis=y lineattrs=(color=gray thickness=1);
run;

```

Output 20.3.1 GNP and Growth Rate of GNP

Hamilton's model is a bivariate regime-switching mean-adjusted AR(4) model which can be described as

$$y_t = c^{(s_t)} + \sum_{\tau=1}^4 \alpha_{\tau}^{(s_t)} (y_{t-\tau} - c^{(s_{t-\tau})}) + \varepsilon_t, \quad \varepsilon_t \sim N(0, \Sigma^{(s_t)})$$

where $s_t \in \{1, 2\}$ is the unobserved state.

In Hamilton (1989), the autoregressive parameters α 's and variance parameters Σ 's are assumed to be state-independent, which means $\alpha_{\tau}^{(1)} = \alpha_{\tau}^{(2)}$, $\tau = 1, \dots, 4$, and $\Sigma^{(1)} = \Sigma^{(2)}$. The following statements re-create Hamilton's result by setting the initial parameter values as the parameter estimates in Table 1 in Hamilton (1989) and also setting the MAXITER= option in the OPTIMIZE statement to zero. The FILTER statement conducts the probabilities of state on the basis of data available up to the current period. The DECODE statement outputs the best possible path of states on the basis of all observations. The SMOOTH statement finds the probabilities of states on the basis of all observations.

```

data mylib.gnpHamilton; set gnpHamilton; run;
proc hmm data=mylib.gnpHamilton labelswitch=(sort=desc(const));
  id time=date;
  model dgnp / armean=adjusted type=ar ylag=4 method=ml nstate=2
              stateIndependent=(ar cov);
  optimize maxiter=0;
  initial tpm={0.9049 0.0951, 0.2450 0.7550},
    ar={0.014 -0.058 -0.247 -0.213, 0.014 -0.058 -0.247 -0.213},
    const={1.1643, -0.3577},
    cov={0.5914, 0.5914};
  filter out=mylib.filter;
  decode out=mylib.decode;
  smooth out=mylib.smooth;
run;

```

The number of observations and model information are shown in [Output 20.3.2](#).

Output 20.3.2 Number of Observations and Model Information

Analysis of the Business Cycle

Observation Information	
Number of Observations	135
Number of Missing Observations	0

Model Information	
Type of Model	Regime-Switching Mean-adjusted Autoregression Model
Stationary	Yes
Number of States	2
Number of Dependent Variables	1
Number of Exogenous Variables	0

The fit statistics are shown in [Output 20.3.3](#).

Output 20.3.3 Model Information

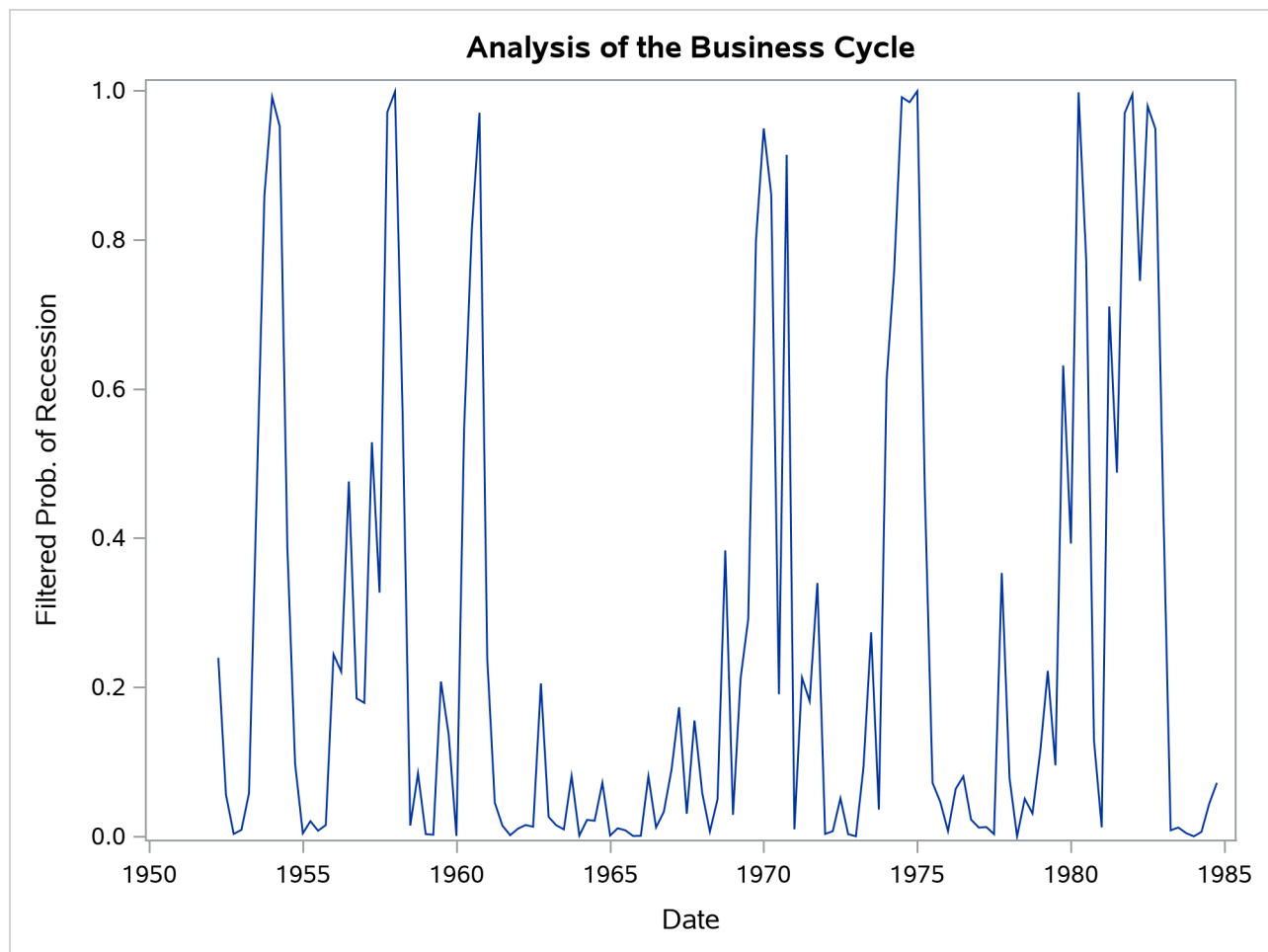
Fit Statistics	
Log Likelihood	-183.1751626
AIC	384.35032521
AICC	385.79032521
BIC	410.49779822
HQC	394.97592522

The following statements plot the filtered probabilities of state 2 in [Output 20.3.4](#), which is comparable to Figure 1 in Hamilton (1989). State 2 corresponds to recession because the intercept (and then the expected mean) in state 2 is negative.

```

data filter; set mylib.filter; run;
proc sort data=filter; by date; run;
proc sgplot data=filter;
    series x=date y=state2;
    yaxis label="Filtered Prob. of Recession";
run;

```

Output 20.3.4 Filter Plot of Economic Recession

In order to show how states correspond to the signs of GNP growth rate, the following statements overlay the decoded states on the GNP growth rate. Recession is from peak to trough, and boom is from trough to peak. As shown in [Output 20.3.5](#), the recession includes most periods that have a negative GNP growth rate.

```

data plot2state;
    merge mylib.decode mylib.gnpHamilton;
    if state=2 then do; recession=1; boom=0; end;
    else if state=1 then do; recession=0; boom=1; end;
    rec = recession*(-3);
    boo = boom*3;
    by date;
    if cmiss(of _all_) then delete;
run;

proc sgplot data=plot2state;

```

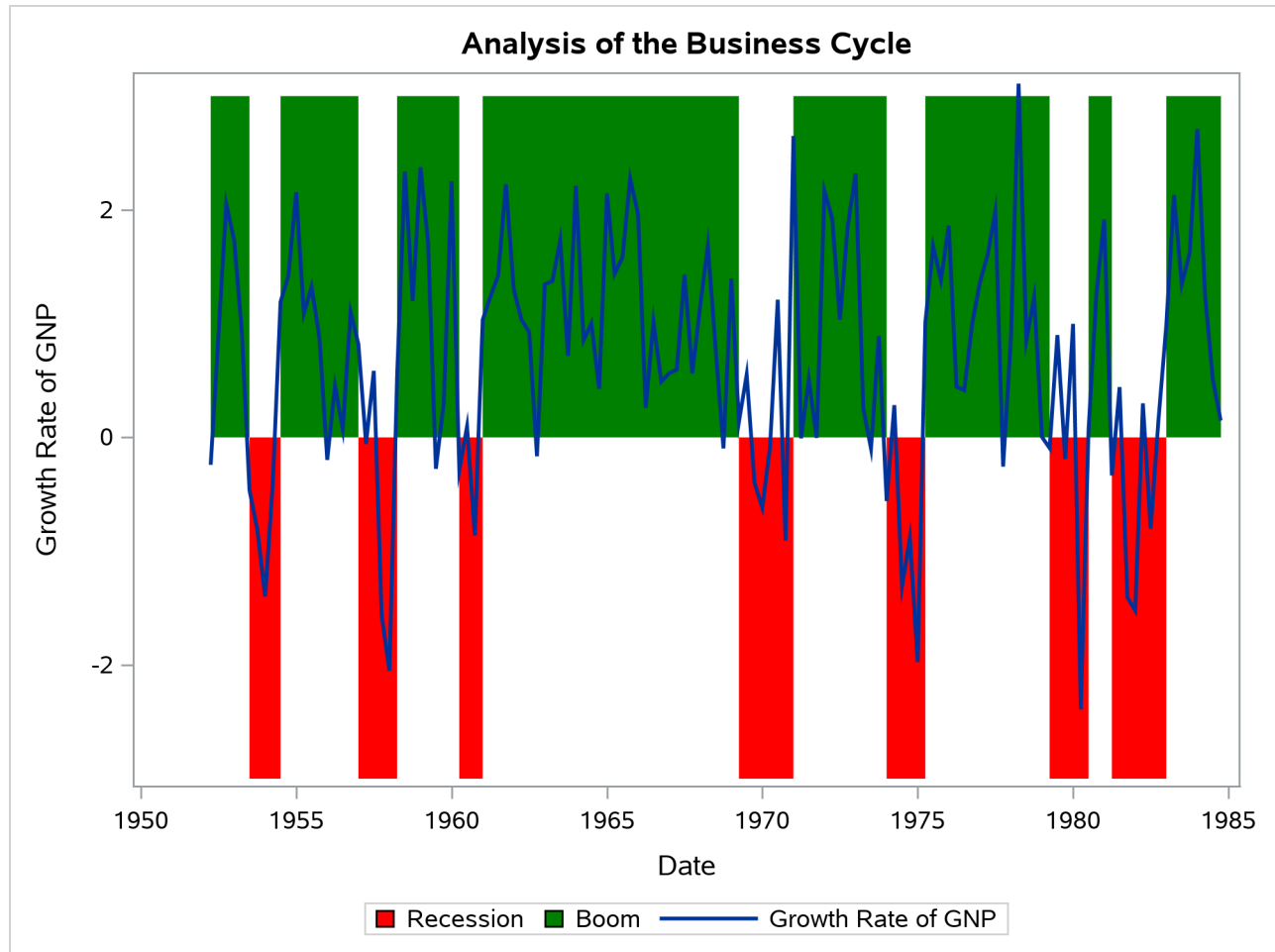


```

band x=date upper=0 lower=rec / legendlabel='Recession' type=step
                                fillattrs=(color=red);
band x=date upper=boo lower=0 / legendlabel='Boom' type=step
                                fillattrs=(color=green);
series x=date y=dgnp / lineattrs=(thickness=2);
run;

```

Output 20.3.5 Decoded Plot of Business Cycle



Hamilton (1989) also compares his choice of business-cycle dates with the historical business-cycle dates that are reported from the National Bureau of Economic Research (NBER). The following data step inputs the business-cycle dates from NBER:

```

data mylib.nber;
  input date: anydtdte. recessionNBER;
  format date yyq6.;
  label date="Date" recessionNBER="Recession Reported from NBER";
datalines;
1951q1  0
1951q2  0
1951q3  0
1951q4  0
1952q1  0

```

```

1952q2  0

... more lines ...

1982q4  1
1983q1  0
1983q2  0
1983q3  0
1983q4  0
1984q1  0
1984q2  0
1984q3  0
1984q4  0
;

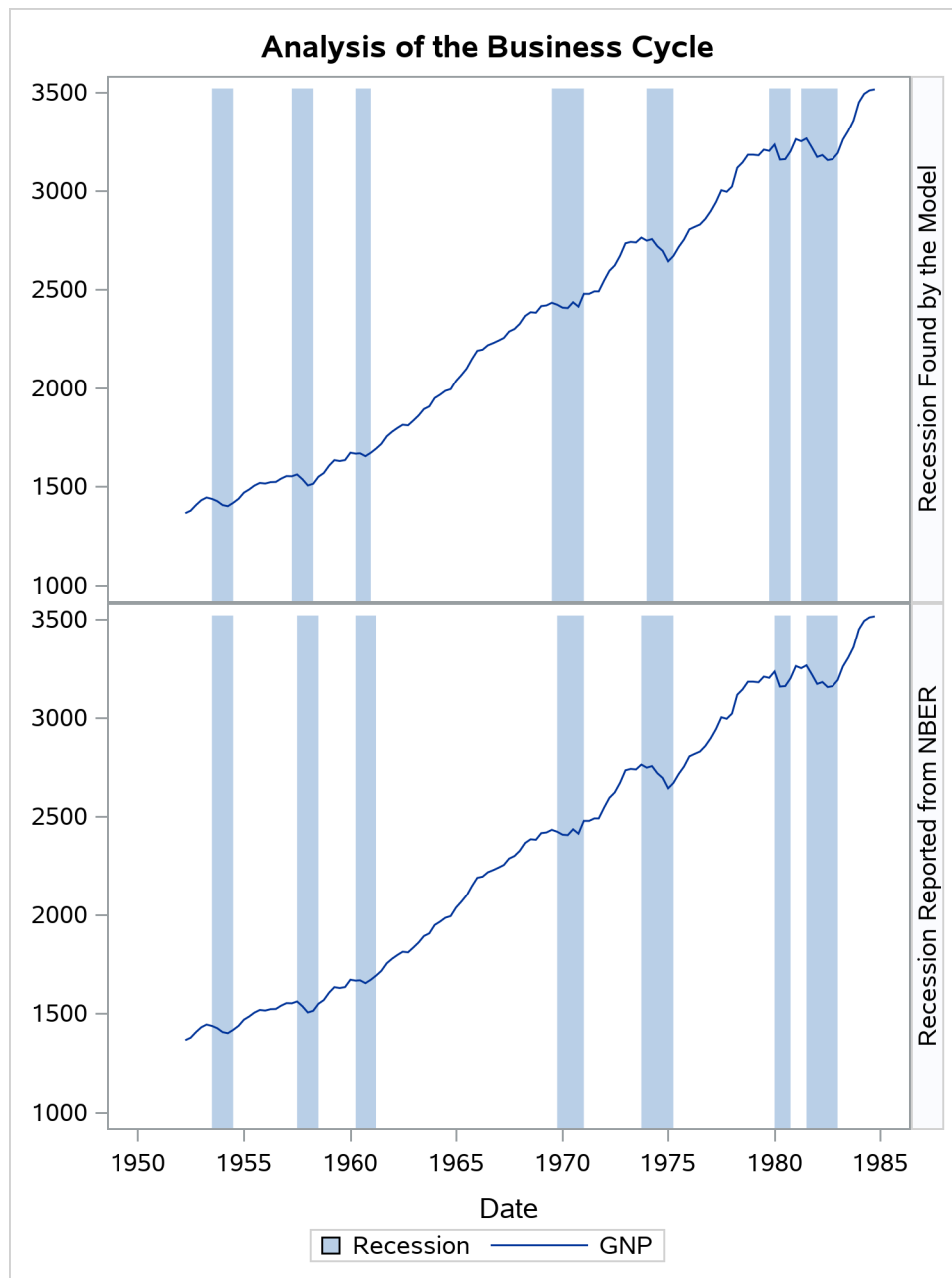
```

The following statements plot together the recession dates that are found by the model and those that are reported from NBER. As shown in [Output 20.3.6](#), the recession dates that are found by Hamilton's model (the upper part) is extremely close to the dates that are reported from NBER (the lower part).

```

data gnpsmooth;
  merge mylib.smooth mylib.nber mylib.gnpHamilton;
  if state2>=0.5 then recession=1; else recession=0;
  by date;
  gnp2=gnp;
  if cmiss(of _all_) then delete;
  label recession="Recession Found by the Model" gnp2="GNP";
run;
proc transpose data=gnpsmooth out=gnpsmooth_t;
  by date;
  var recessionNBER recession;
run;
proc transpose data=gnpsmooth out=gnpsmooth_t2;
  by date;
  var gnp gnp2;
run;
data gnpplot;
  merge gnpsmooth_t(rename=(_name_=name1 _label_=label1 coll=recession))
        gnpsmooth_t2(rename=(_name_=name2 _label_=label2 coll=gnp));
  by date;
  rec=recession*3520;
run;
proc sgpanel data=gnpplot;
  panelby label1 / layout=rowlattice uniscale=column novarname;
  band x=date upper=rec lower=0 / legendlabel='Recession' type=step;
  series x=date y=gnp / legendlabel='GNP';
  rowaxis display=(nolabel) values=(1000 to 3520 by 500);
run;

```

Output 20.3.6 Smoothed Plot of Economic Recession

The following statements use PROC HMM to estimate Hamilton's model while using the STATEINDEPENDENT=(AR COV) option in the MODEL statement so that the autoregression parameter and the variance are fixed across the states:

```
proc hmm data=mylib.gnpHamilton labelswitch=(sort=desc(const));
  id time=date;
  model dgnp / armean=adjusted type=ar ylag=4 method=ml nstate=2
    stateIndependent=(ar cov);
  optimize printIterFreq=1 printlevel=3 ;
  filter out=mylib.filter;
```

```

decode out=mylib.decode;
smooth out=mylib.smooth;
run;

```

The parameter estimates are shown in [Output 20.3.7](#), and the fit statistics are shown in [Output 20.3.8](#). The log likelihood value and the information criterion in [Output 20.3.8](#) show that this solution is better than the one in [Output 20.3.3](#) when MAXITER=0.

Output 20.3.7 Parameter Estimates

Analysis of the Business Cycle

Parameter Estimates				
Parameter	Standard		t Value	Pr > t
	Estimate	Error		
TPM1_1	0.904414	0.037629	24.03	<.0001
TPM1_2	0.095586	0.037629	2.54	0.0123
TPM2_1	0.254962	0.099402	2.56	0.0115
TPM2_2	0.745038	0.099402	7.50	<.0001
CONST1_1	1.160773	0.077552	14.97	<.0001
CONST2_1	-0.392035	0.275951	-1.42	0.1580
AR1_1_1_1	0.016338	0.121529	0.13	0.8933
AR1_2_1_1	-0.088765	0.152015	-0.58	0.5604
AR1_3_1_1	-0.219363	0.114233	-1.92	0.0572
AR1_4_1_1	-0.184025	0.111810	-1.65	0.1024
AR2_1_1_1	0.016338	0.121529	0.13	0.8933
AR2_2_1_1	-0.088765	0.152015	-0.58	0.5604
AR2_3_1_1	-0.219363	0.114233	-1.92	0.0572
AR2_4_1_1	-0.184025	0.111810	-1.65	0.1024
COV1_1_1	0.609788	0.108633	5.61	<.0001
COV2_1_1	0.609788	0.108633	5.61	<.0001

Output 20.3.8 Fit Statistics

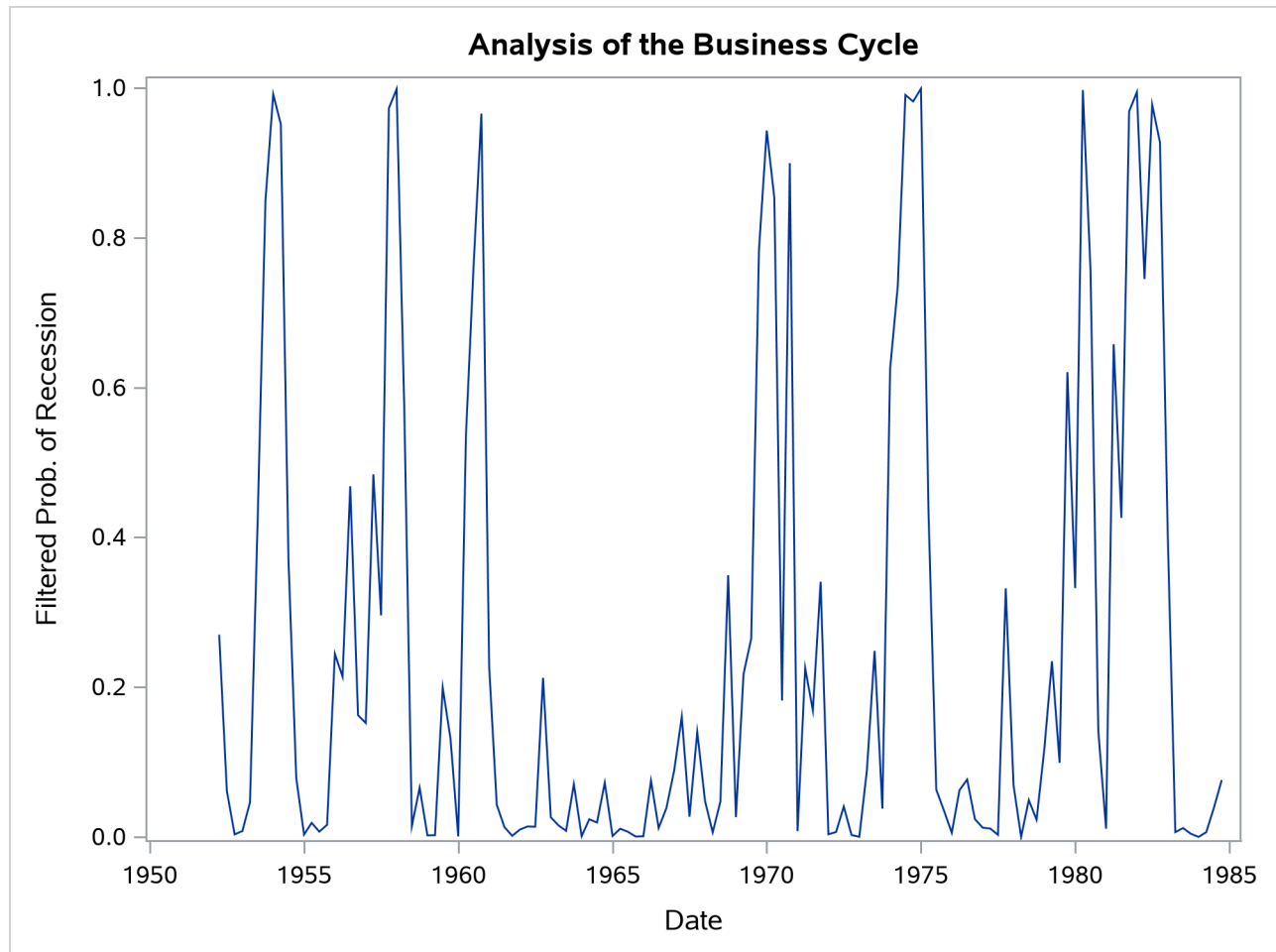
Fit Statistics	
Log Likelihood	-183.0222956
AIC	384.04459127
AICC	385.48459127
BIC	410.19206427
HQC	394.67019127

The following statements plot the filtered probabilities of state 2 (recession). [Output 20.3.9](#) shows the results.

```

data filter; set mylib.filter; run;
proc sort data=filter; by date; run;
proc sgplot data=filter;
  series x=date y=state2;
  yaxis label="Filtered Prob. of Recession";
run;

```

Output 20.3.9 Filter Plot of Economic Recession

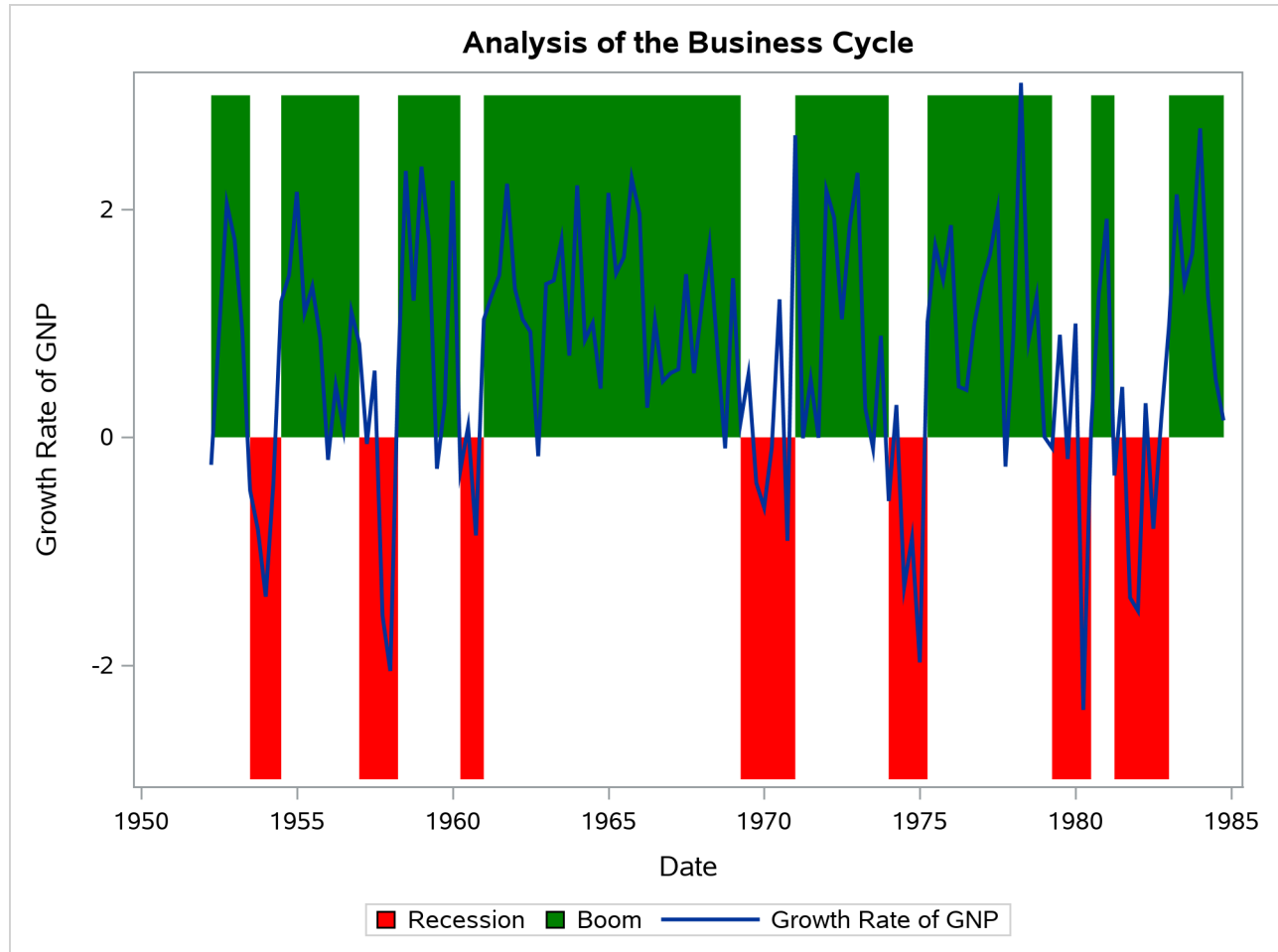
The following statements overlay the decoded states on the GNP growth rate. Recession is from peak to trough, and boom is from trough to peak.

```
data plot2state;
  merge mylib.decode mylib.gnpHamilton;
  if state=2 then do; recession=1; boom=0; end;
  else if state=1 then do; recession=0; boom=1; end;
  rec = recession*(-3);
  boo = boom*3;
  by date;
  if cmiss(of _all_) then delete;
run;

proc sgplot data=plot2state;
  band x=date upper=0 lower=rec / legendlabel='Recession' type=step
                                     fillattrs=(color=red);
  band x=date upper=boo lower=0 / legendlabel='Boom' type=step
                                     fillattrs=(color=green);
  series x=date y=dgnp / lineattrs=(thickness=2);
run;
```

Output 20.3.10 shows that more states switch than are shown in Hamilton's decoded plot. The reason is that the estimates indicate less persistence of each state.

Output 20.3.10 Decoded Plot of Business Cycle



The following statements plot the recession dates that are found by the model and those that are reported from NBER:

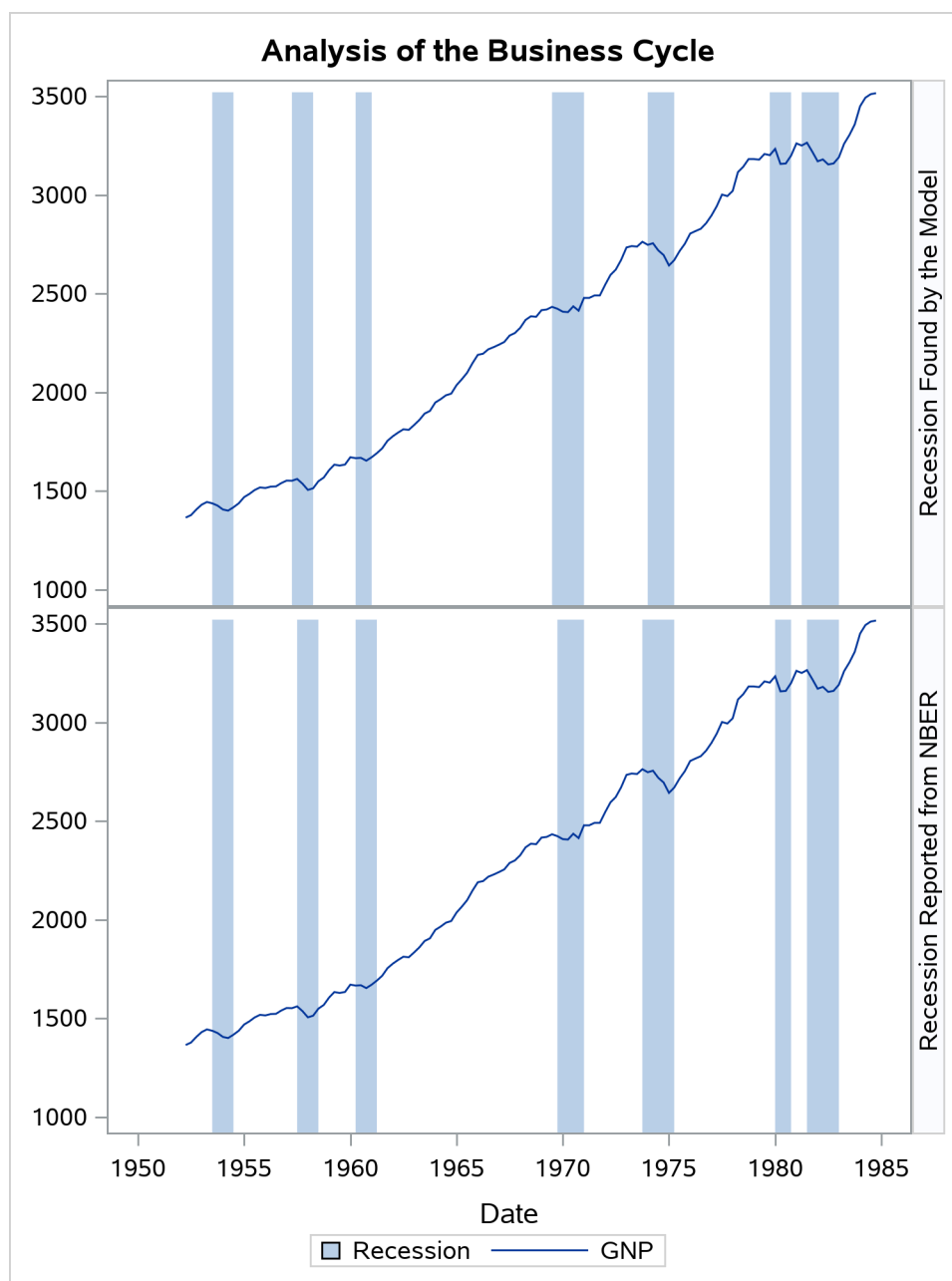
```
data gnpsmooth;
  merge mylib.smooth mylib.nber mylib.gnpHamilton;
  if state2>=0.5 then recession=1; else recession=0;
  by date;
  gnp2=gnp;
  if cmiss(of _all_) then delete;
  label recession="Recession Found by the Model" gnp2="GNP";
run;
proc transpose data=gnpsmooth out=gnpsmooth_t;
  by date;
  var recessionNBER recession;
run;
proc transpose data=gnpsmooth out=gnpsmooth_t2;
  by date;
```

```

var gnp gnp2;
run;
data gnpplot;
  merge gnpsmooth_t(rename=(_name_=name1 _label_=label1 coll=recession))
        gnpsmooth_t2(rename=(_name_=name2 _label_=label2 coll=gnp));
  by date;
  rec=recession*3520;
run;
proc sgpanel data=gnpplot;
  panelby label1 / layout=rowlattice uniscale=column novarname;
  band x=date upper=rec lower=0 / legendlabel='Recession' type=step;
  series x=date y=gnp / legendlabel='GNP';
  rowaxis display=(nolabel) values=(1000 to 3520 by 500);
run;

```

As shown in [Output 20.3.11](#), the recession dates that are found by the model (the upper part) are extremely close to the dates that are reported from NBER (the lower part).

Output 20.3.11 Smoothed Plot of Economic Recession

Example 20.4: Analysis of English Characters by Using a Finite HMM

In natural language processing (NLP), practitioners want to learn the structure of sentences and predict the sequences of words. HMMs have been widely applied for characterizing the statistical features of natural languages; for more information, see Jurafsky and Martin (2019). This example runs the model from Cave and Neuwirth (1980) to illustrate how to use the finite HMM to analyze the English language. The data set used in this example was extracted and converted from the classic novel *Vanity Fair*. To process the raw text, first remove all punctuation and numbers and convert all letters to lowercase.

The following statements create the table for the first 5,000 symbols. The table has two columns: column *t* is the position of the symbol, and column *character* is the value at time *t*.

```
data text;
    input t character $8. ;
datalines;
1    c
2    h
3    a
4    p
5    t
6    e
7    r
8

... more lines ...

4992    c
4993    k
4994
4995    o
4996    n
4997
4998    h
4999    e
5000    r
;
```

When you get data set *text*, you can translate the column value into integers. The following statements relabel the character variables as the unique integers that the finite HMM procedure uses:

```
data tbltemp;
    set text;
    _temp_count_ = 1;
run;

proc sql;
    create table f0 as
        SELECT tbl.character, /* any other columns to be grouped */
               sum(tbl._temp_count_) as count
        FROM tbltemp tbl
        group by tbl.character
        ;
quit;
```

```

data f0Id;
    set f0;
    y = _n_;
    keep character y;
run;

proc print data=f0Id; run;

proc sort data=text out=textSort;
    by character;
run;

PROC SQL;
    CREATE TABLE textLeft AS
    SELECT A.*, B.y
    FROM textSort A LEFT JOIN f0Id B
    ON A.character = B.character
    ORDER BY character;
QUIT;

proc sort data=textLeft out=textLeft;
    by t;
run;

data text2;
    set textLeft;
run;

data mylib.text;;
    set text2;
run;

```

This converts the novel into a sequence of 26 distinct letters and the word space, for a total of 27 symbols. Then, translate the characters to numbers, a to 2, b to 3, ..., z to 27, and word space to 1.

The conversion matching table is shown in [Figure 20.4.1](#).

Output 20.4.1 Matching Table

Obs	character	y
1		1
2	a	2
3	b	3
4	c	4
5	d	5
6	e	6
7	f	7
8	g	8
9	h	9
10	i	10
11	j	11
12	k	12
13	l	13
14	m	14
15	n	15
16	o	16
17	p	17
18	q	18
19	r	19
20	s	20
21	t	21
22	u	22
23	v	23
24	w	24
25	x	25
26	y	26
27	z	27

The following statements print the table after value conversion. The table, which is shown in [Figure 20.4.2](#), contains three columns: column *t* is the position of the symbol, column *character* is the original symbol, and column *y* is the translated value of the symbol at position *t*:

```
proc print data=text2(obs=10) noobs;
run;
```

Output 20.4.2 Converted Data of *Vanity Fair*

t	character	y
1	c	4
2	h	9
3	a	2
4	p	17
5	t	21
6	e	6
7	r	19
8		1
9	i	10
10	c	4

Cave and Neuwirth's model is assumed to have an underlying Markov chain with two hidden states. For each of these hidden states, assume that the 27 symbols appear according to a fixed probability distribution. The following statements estimate a finite HMM in which NSTATE=2 and NCATEGORY=27. There might be a lot of local maximums for the finite HMM; hence, you specify the multistart mode by specifying MULTISTART=1 in the OPTIMIZE statement.

```
data mylib.text;
  set text;
run;
proc hmm data=mylib.text ;
  id time=t;
  model y / type=finite nstate=2 nCategory=27;
  optimize algorithm=interiorpoint multistart=1;
run;
```

The number of observations and model information are shown in [Output 20.4.3](#).

Output 20.4.3 Number of Observations and Model Information

Analysis of English Text

Observation Information	
Number of Observations	5000
Number of Missing Observations	0

Model Information	
Type of Model	Finite HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	1

The parameter estimates are shown in [Output 20.4.4](#), and the fit statistics are shown in [Output 20.4.5](#).

Output 20.4.4 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.308898	0.012144	25.44	<.0001
TPM1_2	0.691102	0.012144	56.91	<.0001
TPM2_1	0.758620	0.016042	47.29	<.0001
TPM2_2	0.241380	0.016042	15.05	<.0001
CPM1_1	0.007426	0.009917	0.75	0.4540
CPM1_2	0.000841	0.004412	0.19	0.8489
CPM1_3	0.024076	0.003013	7.99	<.0001
CPM1_4	0.046623	0.004166	11.19	<.0001
CPM1_5	0.063820	0.004851	13.16	<.0001
CPM1_6	1.473516E-10	0.000000380	0.00	0.9997
CPM1_7	0.034394	0.003591	9.58	<.0001
CPM1_8	0.026369	0.003151	8.37	<.0001
CPM1_9	0.092864	0.005802	16.01	<.0001
CPM1_10	1.425877E-10	0.000000368	0.00	0.9997
CPM1_11	0.008407	0.001788	4.70	<.0001
CPM1_12	0.008347	0.003475	2.40	0.0163
CPM1_13	0.058470	0.004650	12.57	<.0001
CPM1_14	0.055031	0.004515	12.19	<.0001
CPM1_15	0.110443	0.006295	17.54	<.0001
CPM1_16	2.677046E-10	0.000000703	0.00	0.9997
CPM1_17	0.035541	0.003649	9.74	<.0001
CPM1_18	0.004204	0.001266	3.32	0.0009
CPM1_19	0.104711	0.006140	17.05	<.0001
CPM1_20	0.128023	0.006742	18.99	<.0001
CPM1_21	0.111687	0.008254	13.53	<.0001
CPM1_22	6.076342E-10	0.000001795	0.00	0.9997
CPM1_23	0.014140	0.002315	6.11	<.0001
CPM1_24	0.030190	0.003368	8.96	<.0001
CPM1_25	0.003057	0.001080	2.83	0.0047
CPM1_26	0.029808	0.003347	8.91	<.0001
CPM1_27	0.001529	0.000764	2.00	0.0455
CPM2_1	0.332555	0.012576	26.44	<.0001
CPM2_2	0.132087	0.008225	16.06	<.0001
CPM2_3	1.950648E-10	0.000000521	0.00	0.9997
CPM2_4	9.941711E-10	0.000002891	0.00	0.9997
CPM2_5	3.358829E-10	0.000000924	0.00	0.9997
CPM2_6	0.221124	0.009073	24.37	<.0001
CPM2_7	1.189225E-10	0.000000315	0.00	0.9997
CPM2_8	2.0441364E-9	0.000006289	0.00	0.9997
CPM2_9	3.0078461E-9	0.000009440	0.00	0.9997
CPM2_10	0.133430	0.007223	18.47	<.0001
CPM2_11	7.496966E-11	0.000000197	0.00	0.9997
CPM2_12	0.005101	0.003602	1.42	0.1568
CPM2_13	9.85672E-10	0.000002959	0.00	0.9997
CPM2_14	8.918453E-11	0.000000236	0.00	0.9997
CPM2_15	1.481432E-10	0.000000400	0.00	0.9997
CPM2_16	0.123360	0.006965	17.71	<.0001

Output 20.4.4 *continued*

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
CPM2_17	2.135985E-10	0.000000633	0.00	0.9997
CPM2_18	8.173257E-11	0.000000225	0.00	0.9997
CPM2_19	1.372299E-10	0.000000384	0.00	0.9997
CPM2_20	4.145852E-10	0.000001234	0.00	0.9997
CPM2_21	0.011222	0.006957	1.61	0.1068
CPM2_22	0.041120	0.004110	10.00	<.0001
CPM2_23	3.942147E-12	4.8333296E-8	0.00	0.9999
CPM2_24	6.578474E-12	8.3343256E-8	0.00	0.9999
CPM2_25	5.142148E-12	5.9097634E-8	0.00	0.9999
CPM2_26	1.0991882E-9	0.000005356	0.00	0.9998
CPM2_27	4.567458E-12	4.7271039E-8	0.00	0.9999

Output 20.4.5 Fit Statistics

Fit Statistics	
Log Likelihood	-13913.63033
AIC	27939.260663
AICC	27940.552186
BIC	28304.223482
HQC	28067.17439

The most interesting part of this result is the result of the category probability matrix (CPM). In state 1, the probabilities of containing 3, 4, 5, . . . , 26, which correspond to the consonants, are significant. In state 2, the probabilities of containing 1, 2, 6, 10, 16, 22, which correspond to the vowels a, e, i, o, and u and the space, are significant. Without your having made any assumptions about the two hidden states, the CPM shows that one hidden state contains the vowels and the other hidden state contains the consonants. Interestingly, the space is treated more like a vowel in the context of English. The PROC HMM result shows that the distinction between vowels and consonants is a statistically significant feature inherent in the language. And thanks to HMMs, this could easily be deduced by someone who has no background knowledge of the language.

References

- Cappé, O., Moulines, E., and Rydén, T. (2010). *Inference in Hidden Markov Models*. New York: Springer.
- Cave, R. L., and Neuwirth, L. P. (1980). "Hidden Markov Models for English." In *Hidden Markov Models for Speech*, edited by J. D. Ferguson, 16–57. Princeton, NJ: Institute for Defense Analysis.
- Center for Research in Security Prices (2018). "CRSP NYSE/NYSEMKT/Nasdaq/Arca Value-Weighted Market Index." CRSP, University of Chicago Booth School of Business.
- Frühwirth-Schnatter, S. (2006). *Finite Mixture and Markov Switching Models*. New York: Springer.

- Ghassempour, S., Girosi, F., and Maeder, A. (2014). “Clustering Multivariate Time Series Using Hidden Markov Models.” *International Journal of Environmental Research and Public Health* 11:2741–2763. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3968966/>.
- Hamilton, J. D. (1989). “A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle.” *Econometrica* 57:357–384.
- Jurafsky, D., and Martin, J. H. (2019). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Third edition, available as unpublished draft at <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>.
- Krolzig, H.-M. (1997). *Markov-Switching Vector Autoregressions: Modelling, Statistical Inference, and Application to Business Cycle Analysis*. Berlin: Springer.
- Kuester, K., Mittnik, S., and Paolella, M. S. (2006). “Value-at-Risk Prediction: A Comparison of Alternative Strategies.” *Journal of Financial Econometrics* 4:53–89.
- Liao, T. W. (2005). “Clustering of Time Series Data—a Survey.” *Pattern Recognition* 38:1857–1874.
- Oates, T., Firoiu, L., and Cohen, P. R. (1999). “Clustering Time Series with Hidden Markov Models and Dynamic Time Warping.” In *Proceedings of the IJCAI-99 Workshop on Neural, Symbolic, and Reinforcement Learning Methods for Sequence Learning*, 17–21.
- Rabiner, L. R. (1989). “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.” *Proceedings of the IEEE* 77:257–286.
- Rani, S., and Sikka, G. (2012). “Recent Techniques of Clustering of Time Series Data: A Survey.” *International Journal of Computer Applications* 52:1–9.
- Smyth, P. (1997). “Clustering Sequences with Hidden Markov Models.” In *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, edited by M. C. Mozer, T. Petsche, and M. I. Jordan, 648–654. Cambridge, MA: MIT Press.
- Sung, H. G. (2004). “Gaussian Mixture Regression and Classification.” Ph.D. diss., Rice University.

Chapter 21

MKTATTRIBUTION Procedure

Contents

Overview: MKTATTRIBUTION Procedure	1415
Using CAS Sessions and CAS Engine Librefs	1416
Getting Started: MKTATTRIBUTION Procedure	1417
Syntax: MKTATTRIBUTION Procedure	1420
Functional Summary	1420
PROC MKTATTRIBUTION Statement	1421
ID Statement	1422
MODEL Statement	1422
OUTPUT Statement	1423
Details: MKTATTRIBUTION Procedure	1423
Markov Attribution Model	1423
Heuristic Attribution Models	1424
Input Data	1429
Data Table Output	1430
ODS Table Names	1431
Examples: MKTATTRIBUTION Procedure	1431

Overview: MKTATTRIBUTION Procedure

The MKTATTRIBUTION procedure supports different types of market attribution models, which have been used to identify which marketing channels drive customer conversions and to help optimize the investment in those channels. Customers are “converted” when they are persuaded by marketing content to take a particular action, such as buying a product, that a business wants them to take. As the array of media channels where businesses can promote their products and services grows, most customers engage with marketing content through multiple channels, such as in television advertising and in ads on social media. Businesses want to know to what degree each channel contributes to their marketing success. This is called the multichannel attribution problem. The Markov attribution model (MAM), one of the market attribution models, approaches the attribution problem in a probabilistic way by using a Markov chain, a particular stochastic process in which the probability distribution of any next state depends only on what the current state is, regardless of any preceding states. The MAM uses a first-order Markov chain to calculate the probability of interaction between pairs of channels in the customer journey and to evaluate the channel’s contribution to customer conversions through the so-called removal effect. The removal effect would enable a business to give a reliable assessment of the marketing contribution of each channel. In addition, the MKTATTRIBUTION procedure provides a number of heuristic attribution models: the first-touch, last-touch, linear, position-based, and time-decay attribution models.

PROC MKTATTRIBUTION requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following: enables you to run on a cluster of machines that distribute the data and the computations, and exploits all the available cores and concurrent threads.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: MKTATTRIBUTION Procedure

This section provides a brief example of using the Markov attribution model that the MKTATTRIBUTION procedure supports.

Consider a market attribution model that has the following transition probability matrix (TPM) for five states $\{channel1, channel2, channel3, conversion, null\}$:

$$A = \begin{pmatrix} 0.0 & 0.2 & 0.4 & 0.3 & 0.1 \\ 0.1 & 0.0 & 0.4 & 0.5 & 0.0 \\ 0.3 & 0.2 & 0.0 & 0.3 & 0.2 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

It also has the following initial state probability vector (ISPV):

$$\pi = \begin{pmatrix} 0.3 \\ 0.3 \\ 0.4 \\ 0.0 \\ 0.0 \end{pmatrix}$$

The following statements simulate the channel visit history, or the customer journey, for 100 customers from the previous model in order to provide test data for PROC MKTATTRIBUTION. The test data table has three columns: *customerId* is the ID of each customer, *t* is the time period that the customer was in, and *c* is the channel that the customer visited at time *t*. Column *c* takes value *i* for channel *i*, *i* = 1, 2, and 3, and takes a value of 0 when customer conversion occurs. For a customer, if the last column *c* value is not 0, this means that the customer is not converted, which also means that the customer is in the null state. Note that, the channel variable takes the natural numbers sequentially from 1 and can't skip any numbers.

```
%let seed=1234;
%let T=10; *the maxsimustep for each customer;
%let nCustomers=100; *the number of customers;

data simu(keep = customerId t c);

  call streaminit(&seed);

  array p0[5] _temporary_ (0.3, 0.3, 0.4, 0, 0.0);
  array p1[5] _temporary_ (0, 0.2, 0.4, 0.3, 0.1);
  array p2[5] _temporary_ (0.1, 0, 0.4, 0.5, 0.0);
  array p3[5] _temporary_ (0.3, 0.2, 0, 0.3, 0.2);
  do customerId = 1 to &nCustomers;
    c_last=0;
    t=0;
    c=1;
    do while(t < &T. and 0<c<=3);
```

```

c_last=c;
t+1;
if t=1 then
  c = rand("Table", of p0[*]);
else do;
  if c_last=1 then
    c = rand("Table", of p1[*]);
  else if c_last=2 then
    c = rand("Table", of p2[*]);
  else if c_last=3 then
    c = rand("Table", of p3[*]);
  end;
  if c=4 then c=0;
  if c ne 5 then
    output;
  end; *for each customer;
end; *for all simulations;
run;

proc print data=simu (obs=10) noobs; run;

data mylib.marketData; set simu; run;

```

The first 10 observations of the simulated data are shown in [Figure 21.1](#).

Figure 21.1 Simulated Data

customerId	t	c
1	1	3
1	2	0
2	1	1
2	2	3
3	1	1
3	2	3
3	3	0
4	1	1
4	2	3
4	3	1

The following statements estimate the Markov attribution model that has three channels:

```

proc mktattribution data=mylib.marketData;
  id section=customerId time=t;
  model c / nchannel=3;
run;

```

The model information and number of observations are shown in [Figure 21.2](#).

Figure 21.2 Model Information and Number of Observations**The MKTATTRIBUTION Procedure**

Model Information	
Model Type	Markov Attribution
Number of Channels	3
Number of Observations	288
Number of Missing Observations	0

The estimates of the initial state probability vector are shown in [Output 21.3](#).

Figure 21.3 Estimates of Initial State Probability Vector

Initial State Probability Vector	
State	Estimate
Channel 1	0.32000
Channel 2	0.30000
Channel 3	0.38000
Conversion	0.00000
Null	0.00000

The estimated transition probability matrix is shown in [Output 21.4](#).

Figure 21.4 Estimated Transition Probability Matrix

Estimated Transition Probability Matrix					
State	Channel			Conversion	Null
	1	2	3		
Channel 1	0.00000	0.22388	0.47761	0.22388	0.07463
Channel 2	0.10345	0.00000	0.39655	0.50000	0.00000
Channel 3	0.31183	0.13978	0.00000	0.27957	0.26882
Conversion	0.00000	0.00000	0.00000	1.00000	0.00000
Null	0.00000	0.00000	0.00000	0.00000	1.00000

The conversion rates are shown in [Output 21.5](#).

Figure 21.5 Conversion Rates

Conversion Rates	
Method	Rate
Data Based	0.70000
Markov Chain	0.70000

The value in the Data Based row is calculated from the input data. The value in the Markov Chain row is calculated by the estimated TPM. The removal effect of a channel is the decrease in the probability of overall conversion if the channel were removed.

The removal effects are shown in [Output 21.6](#).

Figure 21.6 Removal Effects

Removal Effects	
Channel	Effect
1	0.36540
2	0.40962
3	0.42932

The contributions are the normalized removal effects to indicate each channel's contribution to the conversion. The contributions of each channel through the Markov attribution model and the other three heuristic attribution models are shown in [Output 21.7](#).

Figure 21.7 Channel Contributions

Channel Contributions						
Channel	Markov Chain	First Touch	Last Touch	Linear	Position Based	Time Decay
1	0.30340	0.30000	0.21429	0.27088	0.26352	0.26259
2	0.34012	0.35714	0.41429	0.36942	0.37595	0.37726
3	0.35648	0.34286	0.37143	0.35969	0.36052	0.36015

Syntax: MKTATTRIBUTION Procedure

```
PROC MKTATTRIBUTION DATA=libref.data-table ;
  ID TIME=variable SECTION=variable ;
  MODEL channel-variable / NCHANNEL=number HALFLIFE=number ;
  OUTPUT < options > ;
```

Functional Summary

The statements and options available in the MKTATTRIBUTION procedure are summarized in [Table 21.1](#).

Table 21.1 Functional Summary

Description	Statement	Option
Input Data Table Options		
Specifies the input data table	PROC MKTATTRIBUTION	DATA=
Output Data Table Options		
Writes the contribution estimates to the specified output data table	OUTPUT	OUTCONTRIBUTION=
Writes the removal estimates to the specified output data table	OUTPUT	OUTREMOVAL=
Writes the estimated TPM to the specified output data table	OUTPUT	OUTTPM=

Table 21.1 *continued*

Description	Statement	Option
ID Variable		
Specifies the variable that identifies the section	ID	SECTION=
Specifies the variable that identifies the time or sequence	ID	TIME=
Model Options		
Specifies the half-life of the decaying attribution value	MODEL	HALFLIFE=
Specifies the number of channels	MODEL	NCHANNEL=

PROC MKTATTRIBUTION Statement

PROC MKTATTRIBUTION *DATA=libref.data-table* ;

The PROC MKTATTRIBUTION statement invokes the MKTATTRIBUTION procedure. You must specify the following *option*:

DATA=*libref.data-table*

names the input data table for PROC MKTATTRIBUTION to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1416.

data-table specifies the name of the input data table.

ID Statement

ID *TIME=variable* **SECTION=variable** ;

The ID statement identifies observations in the input data table by specifying two variables for the cross-sectional time series data. That is, for each observation, the combination of the two variables' values must be unique.

You must specify the following options:

TIME=variable

specifies the temporal or sequential order of the observations. The *variable* cannot have missing values.

SECTION=variable

identifies the section or customer ID of each observation. The *variable* cannot have missing values.

MODEL Statement

MODEL *channel-variable* < / *options* > ;

The MODEL statement specifies the channel variable and the number of channels for the market attribution model. Only one MODEL statement is allowed.

You can specify the following *options* after a forward slash (/):

HALFLIFE=number

specifies the half-life the decaying attribution value for the time-decay attribution model. The *number* must be a positive number. By default, the value of the HALFLIFE= option is 1.

NCHANNEL=number

specifies the number of channels for the market attribution model. The *number* must be a positive integer. The *number* must be greater than the value the channel variable takes. If you specify a *number* greater than the maximum value of the channel variable in the input data set, the procedure produces useless information in the TPM. If you specify a *number* less than the maximum value of the channel variable in the input data set, the procedure stops running. By default, the value of the NCHANNEL= option is the maximum value of the channel variable in the input data set.

OUTPUT Statement

OUTPUT < options > ;

The OUTPUT statement saves the estimation results to the specified output data tables. You can specify the following options:

OUTCONTRIBUTION=*libref.data-table*

writes the contribution results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1416.

OUTREMOVAL=*libref.data-table*

writes the removal effects to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1416.

OUTTPM=*libref.data-table*

writes the estimated transition probability matrix to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1416.

Details: MKTATTRIBUTION Procedure

Markov Attribution Model

In the Markov attribution model (MAM), customer journeys are represented as Markov chains. A Markov chain is defined by a set of states $\mathbf{S}_t \in \{1, \dots, K\}$, where t is the time index and K is the number of states; an initial state probability vector (ISPV); and a transition probability matrix (TPM). The ISPV for the MAM can be expressed as a $K \times 1$ vector π , where an element π_i of π denotes the probability of moving from the starting position to the state i ; that is,

$$\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K$$

The TPM for the MAM can be expressed as a $K \times K$ matrix $\mathbf{A}$, where an element a_{ij} of $\mathbf{A}$ denotes the probability of moving from past state i to current state j ; that is,

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

For example, if a business uses three different channels, C1, C2, and C3, in its online marketing strategy, the model would include three states, C1, C2, and C3. Additionally, all Markov chains contain two special states:

a conversion state that represents a successful conversion, and a null state for customer journeys that have not ended in a conversion during the observation period. The full set of states S in this example would therefore look like this: $S = \{C1, C2, C3, conversion, null\}$.

The number of states in the Markov chain is the number of marketing channels plus two states: conversion state and null state. If you assume that the number of marketing channels is N , then $K = N + 2$.

If a customer journey ends in a conversion state, the last channel in the journey is connected to the conversion state; otherwise it leads to the null state. For modeling reasons, the conversion state and the null state always lead to themselves and are considered absorbing states.

The conversion rate is defined as the number of conversions divided by the sum of the number of conversions and the number of null results. The removal effect of a channel is defined as the decrease in probability of reaching the conversion state when you remove the channel from the customer's journey. The normalized removal effect is well suited to measuring the contribution of each channel. The contribution per state as a percentage of the sum of all removal effects is reported (excluding the special conversion and null states).

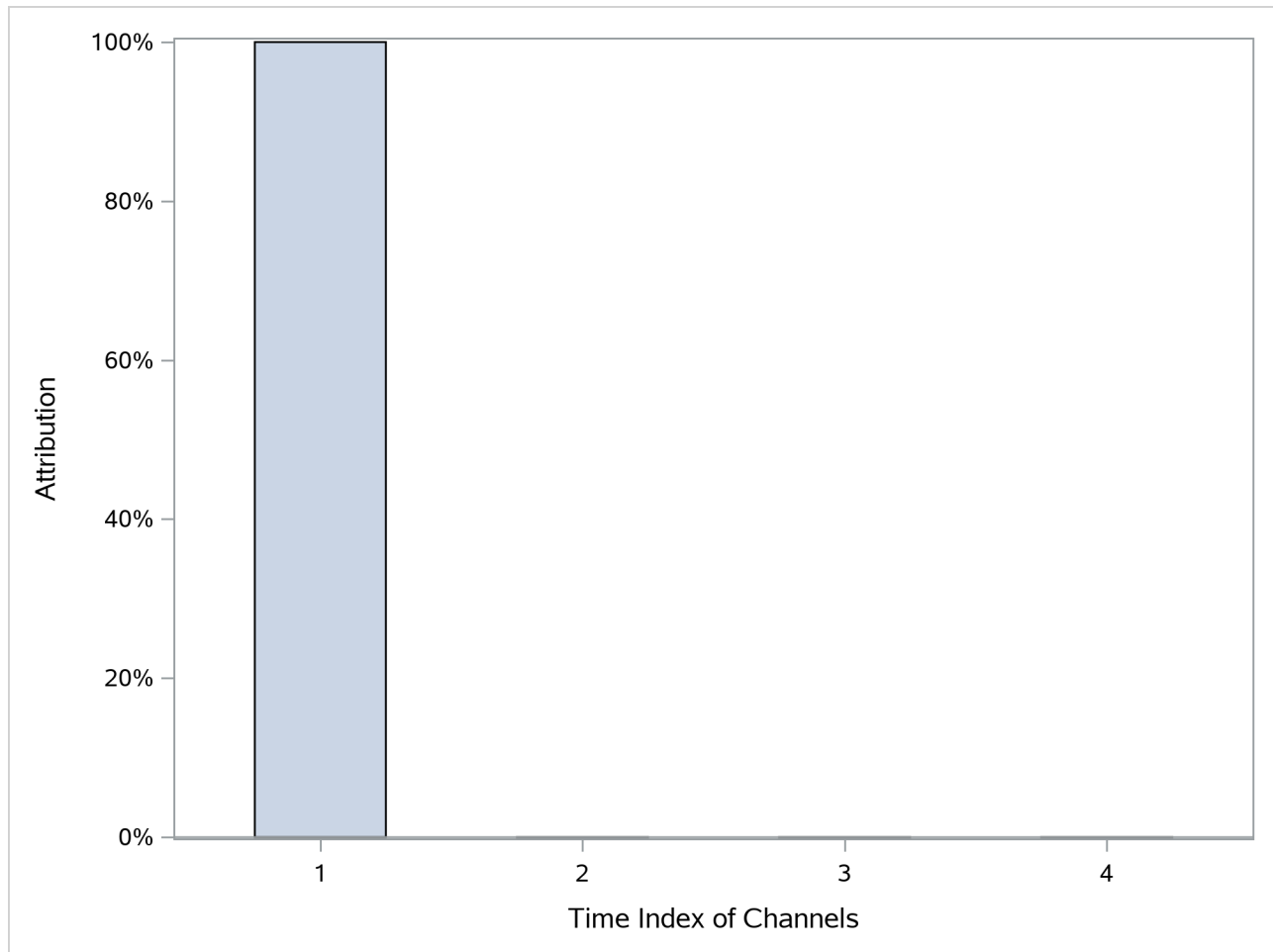
Heuristic Attribution Models

In addition to providing the Markov attribution model, the PROC MKTATTRIBUTION provides several heuristic attribution models: the first-touch, last-touch, linear, position-based, and time-decay attribution models. The first-touch attribution model attributes all credit for the conversion to the first-visited channel in the customer journey. The last-touch attribution model attributes all credit for the conversion to the last-visited channel. The linear attribution model distributes the credit for the conversion equally across all visited channels in the path. The position-based attribution model attributes 40% of credit to the first-visited channel, 40% to the last-visited channel, and the remaining 20% credit equally to the channels in between, if there are more than two visits in the journey. Otherwise, the position-based attribution model is the same as the linear attribution model. The time-decay attribution model attributes the credit increasingly to the channel that is closer to the conversion.

Assuming that there are n customer journeys, the number of conversions is m , where $m \leq n$. $C_j(p)$ is the p th channel in customer j 's journey C_j with length l_j , $p = 1, \dots, l_j$, and $j = 1, \dots, n$. Let $f(C_j(p) = i | conversion) = 1, i = 1, \dots, K$, where K is the number of channels, if $C_j(p) = i$ and conversion occurs at the end of this journey; otherwise it equals 0. The contribution to channel i by the first-touch attribution model is calculated by

$$contribution(i) = \frac{\sum_{j=1}^n f(C_j(1) = i | conversion)}{m}$$

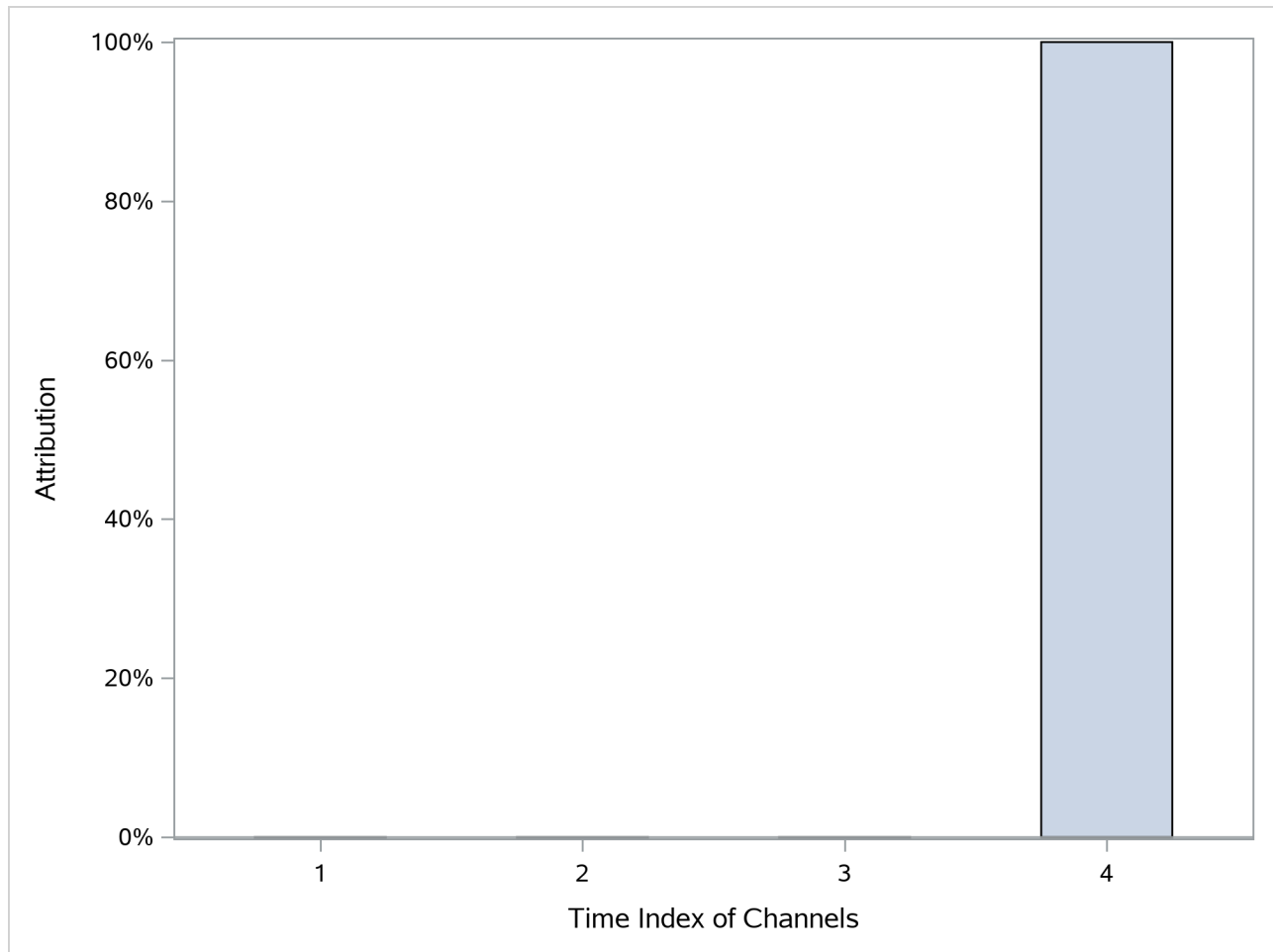
The bar chart in [Output 21.8](#) shows how the first-touch attribution model distributes the contributions for a journey that visits four channels.

Figure 21.8 Bar Chart for the First-Touch Attribution Model

The contribution to channel i by the last-touch attribution model is calculated similarly by

$$contribution(i) = \frac{\sum_{j=1}^n f(C_j(l_j) = i | conversion)}{m}$$

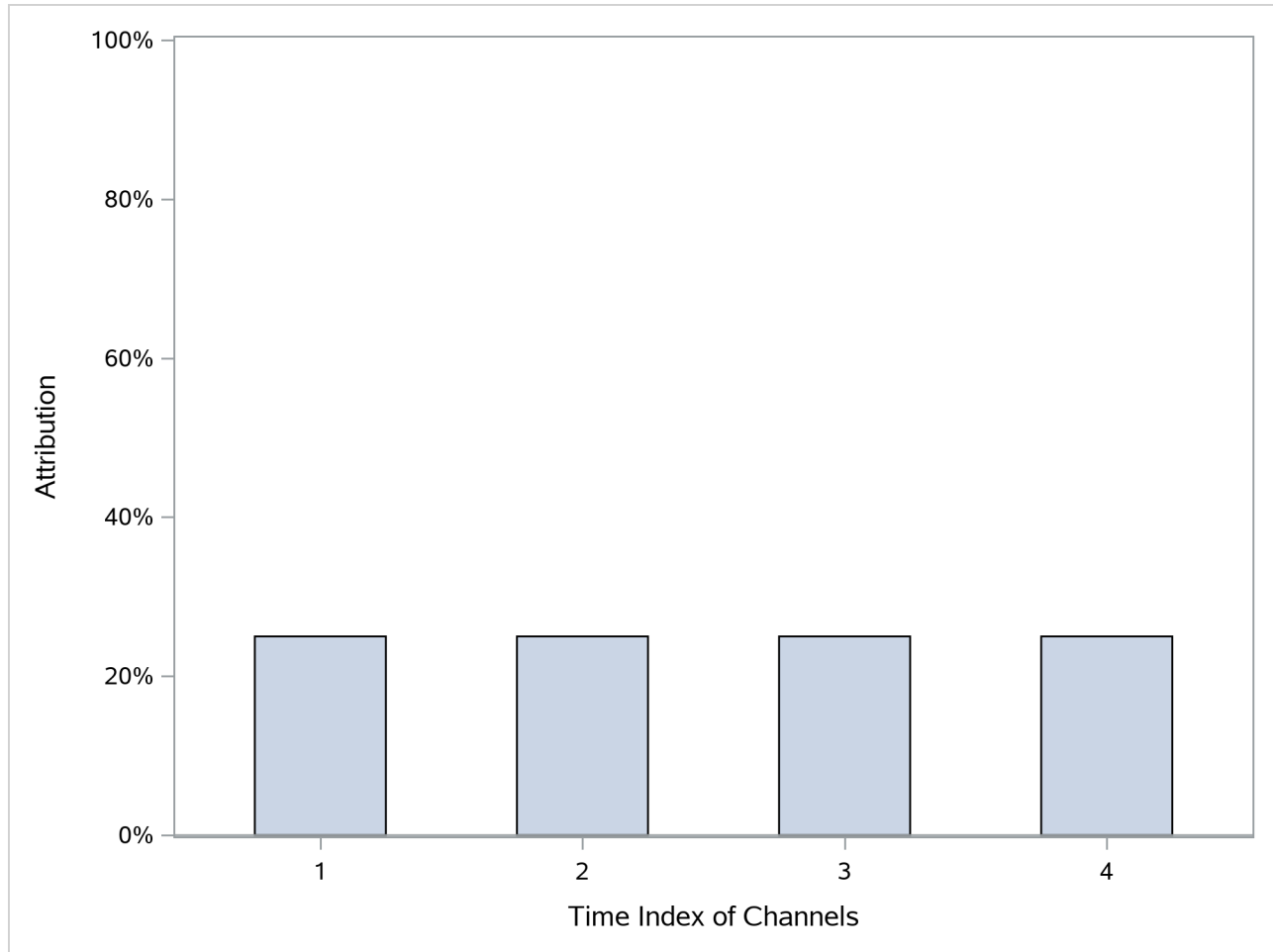
The bar chart in [Output 21.9](#) shows how the last-touch attribution model distributes the contributions for a journey that visits four channels.

Figure 21.9 Bar Chart for the Last-Touch Attribution Model

The contribution to channel i by the linear attribution model is calculated by

$$contribution(i) = \frac{\sum_{j=1}^n \sum_{p=1}^{l_j} f(C_j(p) = i | conversion) / l_j}{m}$$

The bar chart in [Output 21.10](#) shows how the linear attribution model distributes the contributions for a journey that visits four channels.

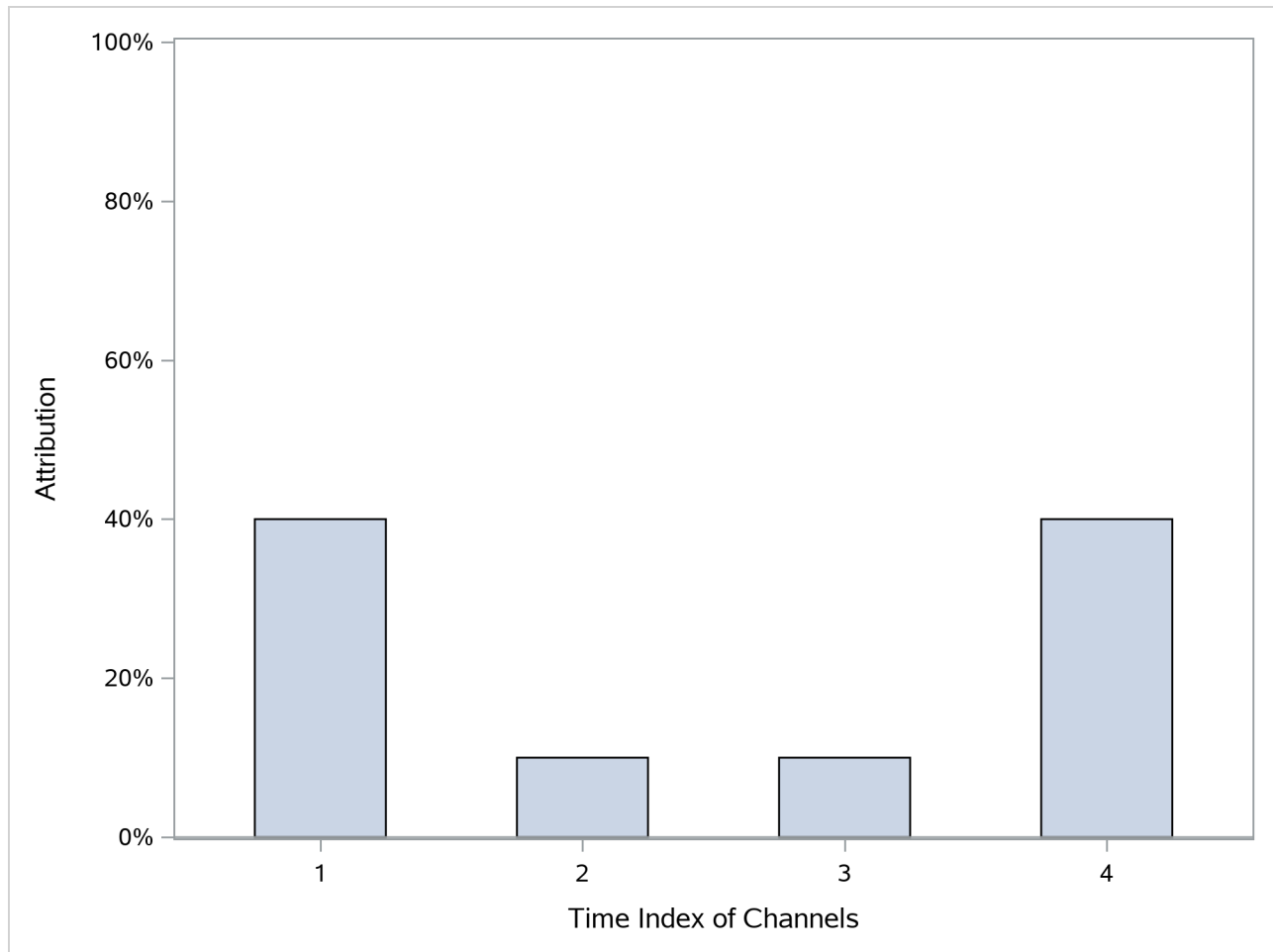
Figure 21.10 Bar Chart for the Linear Attribution Model

If $l_j \leq 2$, the position-based attribution model is the same as the linear attribution model. If $l_j > 2$, then let $h(C_j(p) = i | \text{conversion}) = 0.4$ and $e(C_j(p) = i | \text{conversion}) = 0.2(l_j - 2)^{-1}$ if $C_j(p) = i$ and if conversion occurs at the end of this journey; otherwise it equals 0. The contribution to channel i by the position-based attribution model is calculated by

$$\text{contribution}(i) = \frac{1}{m} \sum_{j=1}^n [\mathbf{A}(l_j \leq 2) + \mathbf{B}(l_j > 2)]$$

where $\mathbf{A}(l_j \leq 2) = \sum_{p=1}^{l_j} f(C_j(p) = i | \text{conversion}) / l_j$, if $l_j \leq 2$, and otherwise it equals 0; and $\mathbf{B}(l_j > 2) = h(C_j(1) = i | \text{conversion}) + h(C_j(l_p) = i | \text{conversion}) + \sum_{p=2}^{l_j-1} e(C_j(p) = i | \text{conversion})$, if $l_j > 2$, and otherwise it equals 0.

The bar chart in [Output 21.11](#) shows how the position-based attribution model distributes the contributions for a journey that visits four channels.

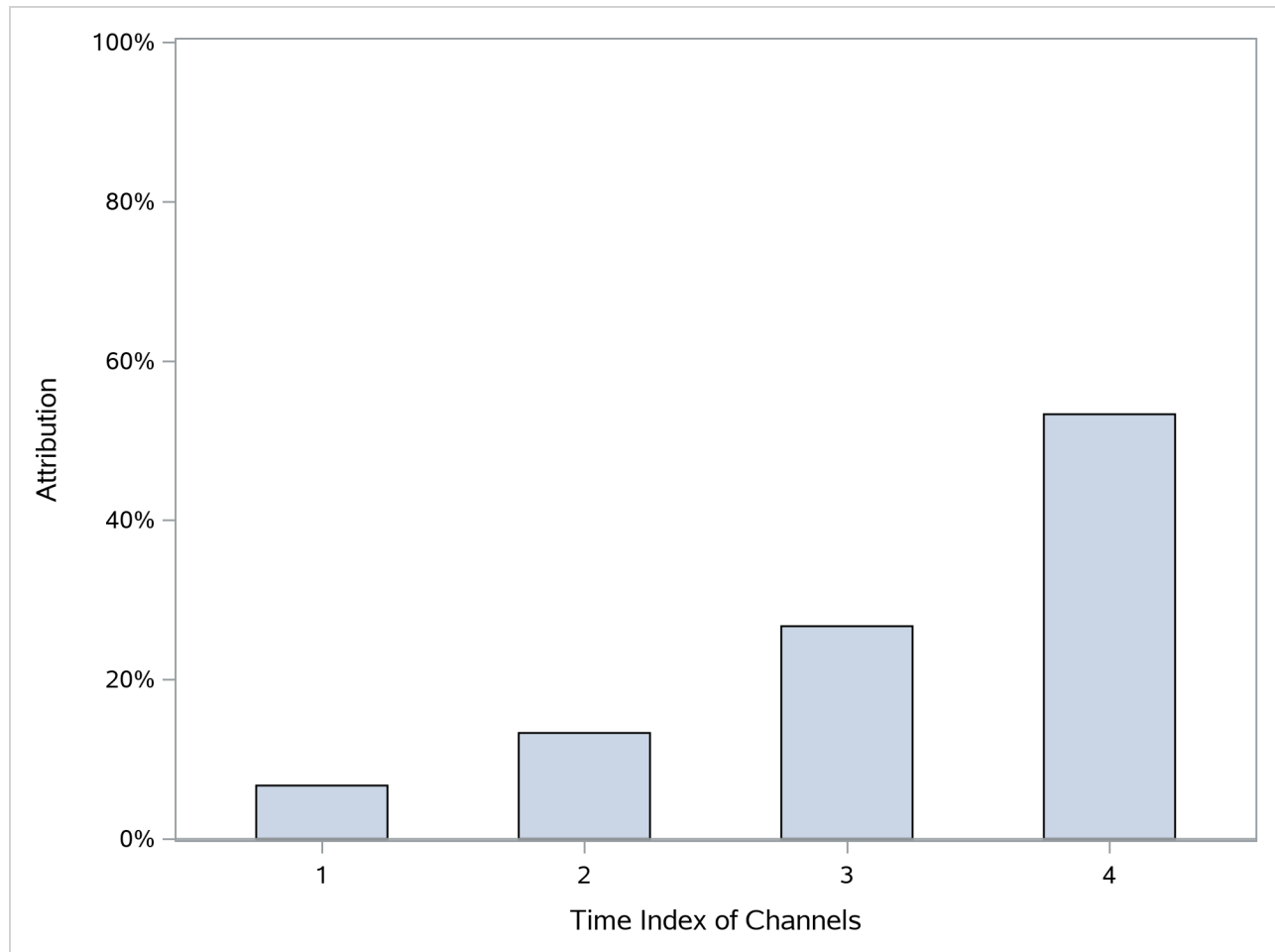
Figure 21.11 Bar Chart for the Position-Based Attribution Model

Let f be the half-life parameter, $g(C_j(p) = i | conversion) = (\frac{1}{2})^{(l_j - p)/f}$ if $C_j(p) = i$ and if conversion occurs at the end of this journey; otherwise it equals 0. The contribution to channel i by the time-decay attribution model is calculated by

$$contribution(i) = \frac{1}{m} \sum_{j=1}^n \left[\sum_{p=1}^{l_j} g(C_j(p) = i | conversion) / D \right]$$

where $D = \sum_{q=1}^{l_j} (\frac{1}{2})^{(l_j - q)/f}$ is the normalizing coefficient for journey j .

The bar chart in [Output 21.12](#) shows how the time-decay attribution model distributes the contributions for a journey that visits four channels.

Figure 21.12 Bar Chart for Time-Decay Attribution Model (HALFLIFE=1)

Input Data

The input data set must have three variables: section ID, time ID, and channel. The channel variable must be a nonnegative integer. The section ID variable identifies each customer. The time ID variable sorts the order of visits of each customer. The combination of section ID and time ID must be unique. The channels can appear in raw data as strings, such as “email”, “online search”, and so on. You need to convert them to the natural numbers and zero, where zero represents the conversion, and save them in the channel variable. So the largest value of the channel variable is the number of distinctive channels in the input data. Any customer journey that does not end with zero ends with a null state. Each customer can have at most only one conversion. The procedure skips records that have a missing channel variable. The procedure skips the records in which the conversion occurs at the beginning or middle of the journey. The procedure skips the records in which consecutive channels occur within the same customer journey, so the estimated TPM always has zero values in the diagonal positions.

Data Table Output

The MKTATTRIBUTION procedure can create the data tables that are specified in the following options in the OUTPUT statement: OUTCONTRIBUTION=, OUTREMOVAL=, and OUTTPM=. The column information that each table contains is described in the following sections.

OUTCONTRIBUTION= Data Table Generated from the OUTPUT Statement

The output data table contains the following variables (columns):

Channel	channel for which to calculate the removal effect
Markov Chain	contribution calculated using the Markov attribution model
First-Touch	contribution calculated using the first-touch model
Last-Touch	contribution calculated using the last-touch model
Linear	contribution calculated using the linear attribution model
Position-Based	contribution calculated using the position-based attribution model
Time-Decay	contribution calculated using the time-decay attribution model

OUTREMOVAL= Data Table Generated from the OUTPUT Statement

The output data table contains the following variables (columns):

Channel	channel for which to calculate the removal effect
Value	removal effect for the corresponding channel

OUTTPM= Data Table Generated from the OUTPUT Statement

The output data table contains the following variables (columns):

State	previous state in the Markov chain
Channel k	probability of transition to channel k , $k = 1, \dots, K$, where K is the number of channels
Conversion	probability of transition to the conversion state
Null	probability of transition to the null state

ODS Table Names

The MKTATTRIBUTION procedure assigns a name to each table that it creates. You can use this name to refer to the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in [Table 21.2](#).

Table 21.2 ODS Tables Produced in the MKTATTRIBUTION Procedure

ODS Table Name	Description	Option
Contributions	Contribution value for each channel calculated for each model	Default
ConversionRates	Conversion rate calculated using input data and from the estimated TPM	Default
ISPV	Initial state probability vector	Default
ModelInfo	Model information	Default
NObs	Observation information	Default
RemovalEffects	Removal effect for each channel	Default
TPM	Transition probability matrix	Default

Examples: MKTATTRIBUTION Procedure

This section provides an example of the model that the MKTATTRIBUTION procedure supports.

The following statements read a data table that contains records of 2 million customer visits. The table has four columns: *sec* is the number to identify the customers, *time* is the time when the customer visits, *c* is a nonnegative integer variable that is converted from the channel variable and is to be used in the model, and *channel* is the channel that the customer visits at time *t*.

```
data marketData;
  format time DATETIME.;
  input sec  time:ANYDTDMM40.  c channel & $ 15.  ;
datalines ;
1   24MAY18:00:00:00    1   Direct Mail
1   05JUN18:23:25:55    4   email
1   05JUN18:23:26:36    0   conversion
2   13JUN19:18:05:25    11  social
2   26OCT19:03:44:08    4   email
3   22AUG19:13:25:49    7   paid social
4   28JUL19:00:52:21    7   paid social

... more lines ...

1169566 15JUN18:11:18:33    5   organic
1169567 05MAY18:00:00:00    1   Direct Mail
1169567 03APR20:05:40:37    5   organic
```

```

1169567 03APR20:12:02:20    0    conversion
1169568 07SEP19:21:50:07    7    paid social
1169569 04MAY19:15:44:22    5    organic
1169570 14AUG18:15:14:56   11    social
1169570 21OCT19:00:00:00    1    Direct Mail
1169571 15MAY19:19:32:36    7    paid social
;

```

The following statements create the table (Output 21.13) to show the one-to-one relationship of channel to c:

```

data channel;
  set marketData(keep=c channel);
run;
proc sort data=channel
  dupout=channel_NoDupkey nodupkey;
  by c;
run;
proc print data=channel noobs; var c channel; run;

```

Figure 21.13 Market Channels

c	channel
0	conversion
1	Direct Mail
2	app
3	display
4	email
5	organic
6	paid search
7	paid social
8	phone
9	print
10	sms
11	social
12	video
13	web

In addition to the conversion, which equals 0, in the c variable, there are 13 distinctive channels shown in Output 21.13.

The following statements estimate the Markov attribution model that has 13 channels:

```

proc sort data=marketData out=marketData nodupkey;
  by sec time;
run;
data mylib.marketData2;
  set marketData;
run;
proc mktattribution data=mylib.marketData2;
  id section=sec time=time;
  model c / nchannel=13;
  output outremoval=mylib.rm outcontribution=mylib.contribution outtpm=mylib.tpm ;
run;

```

The model information and number of observations are shown in [Output 21.14](#).

Figure 21.14 Model Information and Number of Observations

The MKTATTRIBUTION Procedure

Model Information	
Model Type	Markov Attribution
Number of Channels	13
Number of Observations	2045016
Number of Missing Observations	0

The estimates of the initial state probability vector are shown in [Output 21.15](#).

Figure 21.15 Estimates of Initial State Probability Vector

Initial State Probability Vector	
State	Estimate
Channel 1	0.20354
Channel 2	0.00003
Channel 3	0.00706
Channel 4	0.10067
Channel 5	0.20121
Channel 6	0.04194
Channel 7	0.34759
Channel 8	0.00027
Channel 9	0.00159
Channel 10	0.00141
Channel 11	0.07281
Channel 12	0.00330
Channel 13	0.01859
Conversion	0.00000
Null	0.00000

The estimated transition probability matrix are shown in [Output 21.16](#).

Figure 21.16 Estimated Transition Probability Matrix

Estimated Transition Probability Matrix												
State	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7	Channel 8	Channel 9	Channel 10	Channel 11	Channel 12
Channel 1	0.00000	0.00002	0.00896	0.34996	0.23078	0.07311	0.02876	0.00083	0.01499	0.00441	0.00760	0.00112
Channel 2	0.08046	0.00000	0.00000	0.12644	0.45977	0.01149	0.00000	0.00000	0.00000	0.02299	0.00000	0.00000
Channel 3	0.22110	0.00005	0.00000	0.15465	0.12813	0.02383	0.01592	0.00000	0.00070	0.00154	0.00423	0.00035
Channel 4	0.37618	0.00004	0.00918	0.00000	0.20614	0.01010	0.00176	0.00001	0.00038	0.00321	0.00217	0.00016
Channel 5	0.20906	0.00006	0.00776	0.19783	0.00000	0.02327	0.00432	0.00079	0.00284	0.00252	0.00988	0.00053
Channel 6	0.06761	0.00004	0.00680	0.03876	0.20502	0.00000	0.00380	0.00014	0.00236	0.00043	0.00192	0.00068
Channel 7	0.04391	0.00000	0.00372	0.01489	0.02980	0.00188	0.00000	0.00000	0.00002	0.00041	0.01513	0.00004
Channel 8	0.03718	0.00000	0.00000	0.00489	0.29746	0.00489	0.00000	0.00000	0.00000	0.00000	0.00000	0.01272
Channel 9	0.03907	0.00000	0.00108	0.01286	0.17187	0.02150	0.00049	0.00000	0.00000	0.00020	0.00118	0.00010
Channel 10	0.27363	0.00017	0.00670	0.15231	0.23827	0.01039	0.00318	0.00000	0.00034	0.00000	0.00134	0.00017
Channel 11	0.08723	0.00000	0.00374	0.02587	0.06770	0.00275	0.02786	0.00001	0.00015	0.00037	0.00000	0.00002
Channel 12	0.03582	0.00000	0.00361	0.01947	0.12188	0.01803	0.00264	0.00505	0.00096	0.00048	0.00048	0.00000
Channel 13	0.05314	0.00012	0.00258	0.07075	0.26652	0.01966	0.00255	0.00006	0.00317	0.00177	0.00494	0.00112
Conversion	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
Null	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

Estimated Transition Probability Matrix			
State	Channel		
	13	Conversion	Null
Channel 1	0.01082	0.00920	0.25946
Channel 2	0.01149	0.12644	0.16092
Channel 3	0.00473	0.04432	0.40047
Channel 4	0.00488	0.06617	0.31964
Channel 5	0.01339	0.18544	0.34231
Channel 6	0.01079	0.53286	0.12880
Channel 7	0.00043	0.01731	0.87246
Channel 8	0.00391	0.63796	0.00098
Channel 9	0.00982	0.72006	0.02179
Channel 10	0.00419	0.08043	0.22889
Channel 11	0.00180	0.02622	0.75627
Channel 12	0.00505	0.71875	0.06779
Channel 13	0.00000	0.20251	0.37110
Conversion	0.00000	1.00000	0.00000
Null	0.00000	0.00000	1.00000

The conversion rates are shown in [Output 21.17](#).

Figure 21.17 Conversion Rates

Conversion Rates	
Method	Rate
Data Based	0.18771
Markov Chain	0.18771

The removal effects are shown in [Output 21.18](#).

Figure 21.18 Removal Effects

Removal Effects	
Channel	Effect
1	0.07980
2	0.00003
3	0.00379
4	0.06379
5	0.11014
6	0.05682
7	0.01567
8	0.00079
9	0.00833
10	0.00157
11	0.00668
12	0.00335
13	0.01071

The removal effect of a channel is the decrease in the probability of overall conversion if the channel were removed. In [Output 21.18](#), you see that channel 5 (organic) leads to the greatest decrease in the probability of conversion if it is missing.

The contributions are just normalized removal effects to indicate each channel's contribution to the conversion. The contributions of the Markov attribution model and the other three heuristic attribution models are shown side by side in [Output 21.19](#). The last-touch model does not place as much value on channel 1 (direct mail) as other methods do, but it attributes the most value to channel 5 (organic). The first-touch model does not value channel 4 (email) as much as other methods do, but it attributes the most value to channel 1 (direct mail). The Markov and linear attribution models have similar attribution values for most of the channels, except that there is a 9% difference between the two models on the contribution of channel 1 (direct mail). The position-based model and time-decay model both attribute less credit to channel 4 (email), which is the third most important channel according to the Markov model. The reason might be that the email channel always appears in the middle of the customer journey.

Figure 21.19 Channel Contributions

Channel Contributions						
Channel	Markov Chain	Position				Time Decay
		First Touch	Last Touch	Linear	Based	
1	0.22078	0.47248	0.02320	0.23493	0.24179	0.23236
2	0.00008	0.00003	0.00006	0.00005	0.00005	0.00005
3	0.01049	0.00238	0.00486	0.00442	0.00393	0.00408
4	0.17649	0.03489	0.13081	0.08445	0.08369	0.08707
5	0.30470	0.24707	0.43312	0.34435	0.34127	0.34541
6	0.15719	0.15831	0.26382	0.21558	0.21376	0.21484
7	0.04336	0.02774	0.03385	0.03106	0.03094	0.03090
8	0.00217	0.00136	0.00356	0.00255	0.00251	0.00258
9	0.02304	0.00714	0.04001	0.02437	0.02405	0.02434
10	0.00434	0.00092	0.00262	0.00176	0.00177	0.00188
11	0.01847	0.01194	0.01223	0.01237	0.01222	0.01219
12	0.00926	0.01408	0.01631	0.01515	0.01517	0.01521
13	0.02964	0.02166	0.03556	0.02897	0.02884	0.02908

Chapter 22

SASEBEA Interface Engine

Contents

Overview: SASEBEA Interface Engine	1438
Using CAS Sessions and CAS Engine Librefs	1439
Getting Started: SASEBEA Interface Engine	1440
Syntax: SASEBEA Interface Engine	1443
Functional Summary	1443
The LIBNAME <i>libref</i> SASEBEA Statement	1444
Details: SASEBEA Interface Engine	1449
The National Income and Product Accounts	1455
NIPA Underlying Detail National Income and Product Accounts	1458
The Gross Domestic Product by Industry Data	1459
The Underlying Gross Domestic Product by Industry Data	1463
Fixed Assets Data	1464
International Transactions Accounts Data	1469
International Investment Position Data	1515
The Input-Output Statistics Data	1520
International Trade in Services Data	1520
Direct Investment Data and Activities of Multinational Enterprise Data	1527
Regional Data	1544
SAS Output Data Set	1548
Data Elements Reference: SASEBEA Interface Engine	1551
Examples: SASEBEA Interface Engine	1553
Example 22.1: Retrieving Auto and Truck Unit Sales, Production, Inventories, Expenditures, and Price	1553
Example 22.2: Retrieving Table IDs 6 and 25 for the Housing Industry	1557
Example 22.3: Retrieving All Tables for GDP by Industry from the UnderlyingGDP-byIndustry Data Set	1561
Example 22.4: Retrieving One Table from the FixedAssets Data Set	1565
Example 22.5: Retrieving US Direct Investment Position in Canada from the MNE Data Set	1569
Example 22.6: Net Income and Sales for Brazilian, Canadian, and UAE Affiliates of US Parent Enterprises	1571
Example 22.7: Retrieving Available Time Series Names for a Specified Indicator from the ITA Data Set	1576
Example 22.8: Retrieving Time Series for Two Specified Indicators in the ITA Data Set	1580
Example 22.9: Retrieving Quarterly Time Series for Two Types of Investments from the IIP Data Set	1582

Example 22.10: Retrieving Use of Commodities by Industries (Sector Level) from the InputOutput Data Set	1584
Example 22.11: Retrieving Exports and Imports of Advertising Services from the IntlServTrade Data Set	1585
Example 22.12: Retrieving Per Capita Personal Income from the Regional Data Set	1588
Example 22.13: Using the GOCAS= Option	1592
References	1593

Overview: SASEBEA Interface Engine

The SASEBEA interface engine enables SAS users to retrieve time series data from the US Bureau of Economic Analysis (BEA) economic statistics website (www.bea.gov/data). This website offers access to economic statistics data at the industry, regional, national, and international levels.

The industry economic accounts provide a detailed view of the interrelationships between US producers and users. They are used extensively by both policymakers and businesses to understand industry interactions, productivity trends, and the changing structure of the US economy.

The regional economic accounts describe the geographic distribution of US economic activity and growth. The estimates of gross domestic product by state and of state and local personal income, and the accompanying details, help provide a consistent framework for analyzing and comparing individual state and local economies.

The national economic accounts include summary measures such as gross domestic product, corporate profits, personal income and spending, and personal saving.

The international economic accounts provide information about trade in goods and services, such as balance of payments and the balance of trade, investment income, and government and financial flows. These accounts measure the value of US international assets and liabilities and direct investment by multinational enterprises (MNEs). These direct investment data are used to assess the role that business enterprises play in the global economy.

The international transactions data include exports and imports of goods and services, receipts, and payments of income, and international transactions in financial assets and liabilities. The international trade in services data give detailed annual statistics for trade in services and for services provided through affiliates. The US international investment position accounts contain the end-of-period value of accumulated stocks of US financial assets and liabilities.

The input-output accounts focus on the interrelationships between US producers and users.

The SASEBEA interface engine uses the LIBNAME statement to enable you to download the BEA's online economic data from the website at the following root URL:

<https://apps.bea.gov/api/>

It also enables you to specify which time series you want to retrieve, by specifying the data set name and the parameters defined for that data set. You choose the time range of the retrieved data by specifying a list of years. You can then use the SAS DATA step to perform further subsetting, retrieve the data, and store the resulting time series in a SAS data set and in a CAS table (if desired).

You can view the catalog of available BEA databases on the web page at the following URL:

<https://apps.bea.gov/api/data?UserID=<Your-36character-key>&method=GETDATASETLIST>

Every user has a unique user ID key, which the user can obtain by registering at

<https://apps.bea.gov/api/signup/>

The SASEBEA interface engine supports Linux X64 (LAX) and Windows.

Although the SASEBEA engine uses the BEA Data Retrieval API, the interface is not endorsed or certified by the Bureau of Economic Analysis. By using the SASEBEA interface engine, you are agreeing to comply with the terms of service, which are described when you sign up for your user ID key.

To get started using the SASEBEA engine, see the section “[Getting Started: SASEBEA Interface Engine](#)” on page 1440, which shows how to view the National Income and Product Accounts (NIPA), Table T10101, “Percent Change from Preceding Period in Real Gross Domestic Product.” The sample SAS code for accessing the table appears at the end of that section.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: SASEBEA Interface Engine

As described in the preceding section, you need a user ID key to use the BEA economic statistics data web service.

The web page at the following URL describes how to use the BEA's Data Retrieval API:

https://apps.bea.gov/api/_pdf/bea_web_service_api_user_guide.pdf

The statements at the end of this section enable you to access the BEA National Income and Product Accounts (NIPA) Table T10101, which is described as “Table 1.1.1. Percent Change from Preceding Period in Real Gross Domestic Product [Percent] Seasonally Adjusted at Annual Rates.”

The TABLENAME= option specifies which table you want to retrieve the data from in the data set that you specify in the SETNAME= option, which in this sample code is the T10101 table in the NIPA data set.

The OUT= option specifies both the name of the resulting JSON file(s) and the name of the SAS data set.

The range of the retrieved data is determined by the YEAR= option. You specify each year in a quoted list in ‘YYYY’ format.

The JSON data that the BEA's website returns are placed in a file that is named by the OUT= option—in this case, *NIPGDP.json*. **NOTE:** The SASEBEA interface engine appends a numeral to the JSON file name, and the file extension (.json) is excluded from the file name that appears in the OUT= option (or OUTJSON= option). When the SET statement is executed, the JSON data are read (and merged) into a SAS data set named *NIPGDP.sas7bdat*, which resides in the location that is specified inside the string enclosed in double quotation marks in the LIBNAME statement.

The result is named in the DATA step in the SET statement and is shown in [Figure 22.1](#). **NOTE:** It is good practice to use the DATA step with the SET statement, followed by a separate proc print statement that references the output data set named in the DATA step. For more information, see “[SAS Output Data Set](#)” on page 1548.

For another example that uses more LIBNAME statement options, see the section “[Examples: SASEBEA Interface Engine](#)” on page 1553.

```
options validvarname=any;

title 'Retrieve Gross Domestic Product Data for United States, Table T10101';
libname mylib "physical path to the folder where you want the BEA NIPA data";

title1 "National Income and Product Accounts Data";
title2 "Request Table 10101";
libname beanipa sasebea "physical path to the folder where you want the BEA NIPA data"
    setname=NIPA
    TableName="T10101"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' /* access key */
    freq=Q /* quarterly data */
    outjson=NIPGDP
    year='2014,2015,2016,2017'
    ;

data mylib.myGDP;
```

```
    set beanipa.NIPGDP;    /* NIPGDP is specified in the OUT= option */  
run;  
  
title3 "The mylib.myGDP Data Set";  
proc print data=mylib.myGDP; run;
```

Figure 22.1 US Gross Domestic Product in Table T10101

National Income and Product Accounts Data
Request Table 10101
The mylib.myGDP Data Set

Obs	TableName	Date	A006RL	A007RL	A008RL	A009RL	A011RL	A020RL	A021RL	A191RL	A191RP
1	T10101	2014Q1	-4.4	4.1	5.7	14.8	-2.8	-3.8	5.3	-1.4	0.3
2	T10101	2014Q2	18.0	11.6	11.2	14.9	13.4	8.9	9.4	5.2	7.6
3	T10101	2014Q3	8.4	7.9	8.3	1.3	6.0	1.6	0.8	4.7	6.6
4	T10101	2014Q4	0.4	4.7	2.5	6.7	15.0	3.3	10.9	1.8	2.5
5	T10101	2015Q1	14.7	1.1	-0.5	-4.2	8.1	-3.2	5.9	3.3	3.1
6	T10101	2015Q2	1.2	3.5	1.8	3.0	10.4	1.1	3.0	2.3	4.6
7	T10101	2015Q3	-0.8	3.4	1.5	-12.0	11.2	-3.3	3.5	1.3	2.5
8	T10101	2015Q4	-4.7	-1.1	-3.2	-15.1	7.2	-0.6	0.8	0.6	0.5
9	T10101	2016Q1	-1.7	2.4	-0.4	-13.8	12.9	-1.2	-0.6	2.4	2.0
10	T10101	2016Q2	-2.1	2.2	2.6	8.0	0.5	3.1	0.4	1.2	4.1
11	T10101	2016Q3	0.8	3.7	5.3	16.9	-2.0	6.4	2.9	2.4	3.6
12	T10101	2016Q4	10.6	3.1	2.6	6.0	5.1	-2.8	6.1	2.0	4.2
13	T10101	2017Q1	-1.2	6.4	5.2	7.2	10.8	9.9	5.9	1.9	4.0

Obs	A253RL	A255RL	A646RL	A656RL	A822RL	A823RL	A824RL	A825RL	A829RL	DDURRL	DGDSRL
1	-6.8	5.0	3.0	6.4	-1.6	-0.4	-4.9	7.3	-2.4	6.5	2.7
2	10.9	11.7	4.8	-0.8	0.5	-2.3	-2.5	-2.1	2.3	15.2	7.4
3	4.0	1.0	-3.2	-0.3	2.7	4.6	5.2	3.6	1.6	7.5	4.7
4	3.4	10.6	3.3	12.7	-0.2	-5.7	-10.6	2.5	3.4	7.8	5.5
5	-7.5	8.9	6.2	-6.9	2.1	1.7	-1.0	5.9	2.4	7.4	4.9
6	2.8	2.8	-2.2	3.8	3.5	0.7	0.3	1.3	5.2	8.7	4.3
7	-4.1	2.9	-1.8	6.1	1.8	-0.4	-3.8	4.8	3.2	5.5	4.6
8	-2.2	-1.2	2.6	10.2	1.4	3.0	3.2	2.8	0.5	2.3	1.6
9	2.0	-0.2	-6.8	-2.2	4.3	1.0	-0.4	3.1	6.3	5.3	4.1
10	0.0	0.3	9.0	0.9	-0.6	-3.2	-5.9	0.8	1.0	4.0	3.5
11	7.4	2.1	4.4	6.7	2.1	2.5	4.8	-0.8	1.8	10.4	4.0
12	0.3	8.5	-8.3	-3.9	0.8	0.3	-0.4	1.3	1.2	5.7	2.2
13	8.4	4.6	12.7	11.7	-0.8	-2.8	-3.7	-1.5	0.3	2.8	3.5

Obs	DNDGRL	DPCERL	DSERRL	Y001RL	Y033RL
1	0.9	1.5	1.0	2.5	3.5
2	3.9	3.8	2.1	9.5	10.4
3	3.4	4.0	3.7	7.4	12.8
4	4.4	4.5	4.1	8.6	-3.5
5	3.7	2.7	1.7	-2.9	3.3
6	2.2	2.7	2.0	1.4	1.5
7	4.1	3.1	2.4	5.5	6.5
8	1.2	2.0	2.2	9.7	-5.0
9	3.5	3.0	2.5	12.9	-2.1
10	3.3	2.0	1.4	10.0	-4.9
11	0.9	2.4	1.6	7.6	-1.8
12	0.4	2.0	1.9	3.4	0.3
13	3.8	2.9	2.7	7.3	2.4

Figure 22.1 *continued*

**National Income and Product Accounts Data
Request Table 10101
The mylib.myGDP Data Set**

Obs	TableName	Date	A006RL	A007RL	A008RL	A009RL	A011RL	A020RL	A021RL	A191RL	A191RP
14	T10101	2017Q2	7.2	3.8	4.2	0.2	2.6	1.6	4.3	2.3	3.6
15	T10101	2017Q3	6.2	0.5	1.2	-10.6	-1.9	2.7	-0.1	2.9	5.0
16	T10101	2017Q4	4.7	8.1	8.4	4.3	7.0	9.7	10.7	3.8	6.8

Obs	A253RL	A255RL	A646RL	A656RL	A822RL	A823RL	A824RL	A825RL	A829RL	DDURRL	DGDSRL
14	-0.1	3.5	4.8	7.9	1.2	3.4	8.6	-3.7	-0.1	4.4	4.3
15	2.0	-1.0	3.9	3.7	-0.3	-0.1	-1.1	1.3	-0.4	9.7	4.4
16	14.4	15.6	1.5	-8.1	2.8	4.9	5.4	4.1	1.7	15.3	8.2

Obs	DNDGRL	DPCERL	DSERRL	Y001RL	Y033RL
14	4.3	1.9	0.8	1.7	8.5
15	1.8	2.3	1.3	7.6	3.1
16	4.8	4.3	2.5	6.7	12.0

Syntax: SASEBEA Interface Engine

The SASEBEA interface engine uses standard engine syntax to read the observations or data values for one or more time series from BEA's economic statistics data website.

Functional Summary

Table 22.1 summarizes the options that the SASEBEA engine uses.

Table 22.1 Summary of SASEBEA Interface Engine Options

Option	Description
AFFILIATION=	Specifies the affiliation classification of the services transaction
AREAORCOUNTRY=	Specifies the counterparty area or country for which to select BEA time series data
CASLIB=	Specifies the name of the caslib where the in-memory CAS table containing the BEA data is stored
CASOUT=	Specifies the name of the in-memory CAS table, which contains the data that the SASEBEA interface engine returns
CLASSIFICATION=	Specifies the name of the classification by which the data are returned
COMPONENT=	Specifies either the position (Pos) or a component in the change of position (ChgPosTrans) from the previous period
COUNTRY=	Specifies the geographic area code of the selected data to retrieve
DEBUG=	Specifies whether to include diagnostic message logging in the SAS log window
DESCRIPTION=	Specifies whether to use a long or short label
DIRECTIONOFINVESTMENT=	Specifies the direction of investment for multinational enterprise, with respect to United States (inward or outward)

Table 22.1 *continued*

Option	Description
FOOTNOTES=	Specifies whether to include footnotes in the data to retrieve
FREQ=	Specifies the frequency of the data to retrieve
GEOFIPS=	Specifies a list of the federal information processing system (FIPS) codes for state, county, or MSA codes of the data to retrieve
GOCAS=	Specifies whether to generate an in-memory CAS table for the BEA data
INDICATOR=	Specifies a list of series indicators to retrieve
INDUSTRY=	Specifies a list of industries to retrieve
LINECODE=	Specifies the line code in the table to retrieve
NONBANKAFFILIATESONLY=	Specifies whether the selection of data is for nonbank affiliates only (1) or for both bank and nonbank affiliates (0)
OUT=	Specifies the name of the SAS data set and the JSON file, which contains the data that the SASEBEA interface engine returns
OWNERSHIPLEVEL=	Specifies the selection of data for all affiliates (OWNERSHIP=1), or for majority-owned affiliates (OWNERSHIP=0)
SERIESID=	Specifies the series identifier (integer) of the time series data to retrieve
SETNAME=	Specifies the name of the BEA data set from which to retrieve the time series data
STATE=	Specifies the two-digit FIPS code of the state to select the time series data to retrieve
TABLEID=	Specifies the table identifier of the data table to retrieve
TABlename=	Specifies the name of the data table to retrieve
TRADEDIRECTION=	Specifies the trade direction to retrieve data for
TYPEOFINVESTMENT=	Specifies the type of investment to retrieve data for
TYPEOFSERVICE=	Specifies the type of service to retrieve data for
USER=	Specifies the required BEA access key to authenticate the request for the retrieved data
YEAR=	Specifies a list of years to retrieve data for

The LIBNAME *libref* SASEBEA Statement

LIBNAME *libref* **SASEBEA** '*physical-name*' *options* ;

The LIBNAME statement assigns a SAS library reference (*libref*) to the physical path of the directory where the SASEBEA interface engine stores the BEA economic data. The required *physical-name* argument specifies the final location of the folder where your BEA data reside in a SAS data set that is named in the OUT= option. The *physical-name* should end with a backslash (for example, C:*physical_name*\) when you are in a Windows environment or a forward slash (/user/*physical_name*/) when you are in a UNIX environment.

You can specify the following *options*:

AFFILIATION=*bea_affiliation*

specifies the affiliation classification of the services transactions for which to select BEA time series data. See [Table 22.54](#) for a list of valid affiliation classifications.

AREAORCOUNTRY=*bea_areaorcountry*

specifies the counterparty area or country for which to select BEA time series data. The requested data are organized according to the specified counterparty area(s) or countries. See [Table 22.53](#) for a list of available counterparty area or countries for the IntlServTrade data.

CASLIB=*bea_caslib_name*

specifies the name of the CAS library (caslib) where the in-memory CAS table is stored when the JSON data from the BEA website are read into SAS. You can use this option to name your caslib where the in-memory CAS table is stored. The default value is the name of the active caslib. The GOCAS=ON option is required when you specify the CASLIB= option.

CASOUT=*libref.data-table***OUTCAS=***libref.data-table*

specifies the name of the in-memory CAS table that is created when the JSON data from the BEA are read into SAS. You can use this option to name your in-memory CAS table. The default value is the name that you specify in the OUT= option. The in-memory table is created only when you also specify the GOCAS=ON option. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1439.

CLASSIFICATION=*bea_classification***CLASS=***bea_classification*

specifies the name of the classification key (such as COUNTRY or INDUSTRY) with which to group BEA time series data. You can use this option to name one of the key classifications shown in [Table 22.58](#). The requested data are organized according to the specified classification.

COMPONENT=*bea_component*

specifies a position or a component in the change of position from the previous period. You can use this option to select a component or multiple components to retrieve BEA time series data. The requested data are organized according to the specified component(s). See [Table 22.51](#) for a list of available components and their descriptions.

COUNTRY=*bea_country***COUNTRIES=***bea_country*

specifies the geographic area code of the country with which to select BEA time series data. You can use this option to select a geographic area to retrieve BEA time series data from. The requested data are organized according to the specified geographic area code(s). See [Table 22.59](#) for a list of available country geographic area codes for MNE data, and see [Table 22.48](#) for International Transactions Accounts (ITA) data.

DEBUG=ON | OFF

specifies whether or not to include diagnostic message logging in the SAS log window. This information can be useful for troubleshooting a problem. By default, DEBUG=OFF.

DESCRIPTION=LONG | SHORT

For data from the International Trade in Services, specifies whether to use the short BEA time series indicator (using indicators whose names start with the prefix ‘TSI_Ist’) as the time series label, or whether to use the long BEA time series description as the time series label. By default, DESCRIPTION=SHORT. When printing the output data set, it is useful to use the LABEL option on the PROC PRINT statement so that each column heading matches the BEA time series indicator for each time series variable. When DESCRIPTION=LONG and the LABEL option is used on the PROC PRINT statement, each column heading matches the time series description of each time series variable.

DIRECTIONOFINVESTMENT=OUTWARD | INWARD | STATE | PARENT**DIRECT_OF_INVEST=OUTWARD | INWARD | STATE | PARENT**

For data from the Direct Investment (DI) Multinational Enterprise (MNE) data set, specifies whether the direction is outward, as in US direct investment abroad, or inward, as in foreign investment in the United States.

For data from the Activities of Multinational Enterprises (AMNE) data set, in addition to OUTWARD or INWARD, specify STATE when requesting the data on a state level, or specify PARENT for data on US parent enterprises.

The DIRECTIONOFINVESTMENT= option is important to the NONBANKAFFILIATESONLY= option; together these two options influence the range of years of data to be retrieved. When DIRECTIONOFINVESTMENT=OUTWARD and NONBANKAFFILIATESONLY=0, the data that are retrieved range from 2009 to the present. When DIRECTIONOFINVESTMENT=INWARD and NONBANKAFFILIATES=0, the data that are retrieved range from 2007 to the present. When DIRECTIONOFINVESTMENT=OUTWARD and NONBANKAFFILIATESONLY=1, the data that are retrieved are for AMNE “outward” data up to 2008. When DIRECTIONOFINVESTMENT=INWARD and NONBANKAFFILIATESONLY=1, the data that are retrieved are for AMNE “inward” data up to 2006.

FOOTNOTES=YES | NO

specifies whether or not to include footnotes in the retrieved data. By default, FOOTNOTES=NO.

FREQ=A | Q | QSA | QNSA | M**FREQUENCY=A | Q | QSA | QNSA | M****INTERVAL=A | Q | QSA | QNSA | M****PERIOD=A | Q | QSA | QNSA | M**

specifies the selection frequency for the time series. You can specify one of the following frequencies:

A specifies annual frequency.

Q specifies quarterly frequency.

QSA specifies seasonally adjusted quarterly frequency (ITA data set only).

QNSA specifies not seasonally adjusted quarterly frequency (ITA data set only).

M specifies monthly frequency.

GEOFIPS=*bea_geofips*

specifies a list of the federal information processing system (FIPS) codes for state, county, or metropolitan statistical area (MSA) or other codes of the data to retrieve. You can specify *bea_geofips* as a comma-delimited string enclosed in double quotes. Specify GEOFIPS=STATE to select all states, GEOFIPS=COUNTY to select all counties, GEOFIPS=MSA to select all metropolitan statistical areas, GEOFIPS=MIC to select all micropolitan statistical areas, GEOFIPS=DIV to select all metropolitan divisions, GEOFIPS=CSA to select all combined statistical areas, or GEOFIPS=PORT to select all metropolitan and nonmetropolitan portions. The list can contain ANSI state-county codes or metropolitan area codes, which are described at the following web page:

<https://transition.fcc.gov/oet/info/maps/census/fips/fips.txt>

In addition, a comprehensive list of MSAs and their component counties is given at the following web page:

<https://apps.bea.gov/regional/docs/msalist.cfm>

GOCAS=ON | OFF

specifies whether to create (ON) or not create (OFF) an in-memory CAS table of the BEA data (in addition to the SAS data set that is created). When GOCAS=ON, the SASEBEA engine assumes that there is an active CAS session running when it executes the LIBNAME statement. By default, GOCAS=OFF.

INDUSTRY=*bea_industries***INDUSTRIES=***bea_industries*

specifies a list of the desired industries for the time series data selection. You can specify *bea_industries* as a comma-delimited string enclosed in double quotes. You can specify INDUSTRY=ALL to select all available industries. See [Table 22.60](#) for a list of available industry codes.

LINECODE=*bea_linecode*

specifies the line that corresponds to the statistic in a table that is required for the time series data selection. You can specify only one line code. You must use only one GEOFIPS= specification when you specify LINECODE=ALL. GEOFIPS=STATE, COUNTY, MSA, MIC, DIV, CSA, or PORT are treated as multiple GEOFIPS codes so these cannot be used with LINECODE=ALL. When a multiple GEOFIPS code is used, specify only one LINECODE (for example, LINECODE=1). This option is required.

NONBANKAFFILIATESONLY=0 | 1**NONBANKONLY=0 | 1**

specifies whether to return data for nonbank affiliates only (1) or for both bank and nonbank affiliates (0). Specify NONBANKAFFILIATESONLY=0 for AMNE “outward” data from 2009 to present and for “inward” AMNE data from 2007 to present. Specify NONBANKAFFILIATESONLY=1 for AMNE “outward” data up to 2008 and for “inward” AMNE data up to 2006. This option is required when you specify the OWNERSHIPLEVEL= option.

OUT=*bea_jsonfile***OUTJSON=***bea_jsonfile***JSON=***bea_jsonfile*

specifies the name of the JSON file (downloaded) and the name of the SAS data set that is created when the JSON data are read into SAS. You can use the OUT= option to name the JSON data file. It is

stored in the SAS Work library. The SAS data set that is created (when the JSON data are read into SAS) is stored in the folder specified by *physical-name*. You can refer to it by using the myLib libref in the LIBNAME statement.

OWNERSHIPLEVEL=0 | 1

OWNERS=0 | 1

specifies whether to return data for all affiliates (1) or to return data for majority-owned affiliates only (0). This option is used only with AMNE statistics and is required. When DIRECTIONOFINVESTMENT=PARENT, the OWNERSHIPLEVEL= option must be set to 1. You must specify the OWNERSHIPLEVEL= option to retrieve AMNE data.

SERIESID='bea_seriesid'

SERIESIDS='bea_seriesid'

INDICATOR='bea_seriesid'

SERIESINDICATOR='bea_seriesid'

specifies a list of the BEA time series identifiers (or indicators) to retrieve, enclosed in quotes. You can specify *bea_seriesid* as a comma-delimited string enclosed in double quotes. For the ITA data set, you can specify INDICATOR=ALL to select all available indicators. See [Table 22.24](#) through [Table 22.47](#) for a list of available ITA indicator codes. For the MNE data set, you can specify SERIESID='0' to retrieve all the available series IDs whose criteria match the rest of the SASEBEA LIBNAME option specifications. [Table 22.57](#) shows the list of BEA series identifiers for MNE data.

SETNAME=bea_setname

DATASETNAME=bea_setname

DATASET=bea_setname

SETID=bea_setname

specifies the BEA data set name to retrieve the BEA data from. [Table 22.2](#) shows the list of BEA data set names.

STATE=bea_statecode

STATES=bea_statecode

specifies the unique two-digit state FIPS code to select which data are retrieved. [Table 22.61](#) shows the list of state FIPS codes with their corresponding state names. Only one FIPS code is allowed.

TABLEID=bea_tableids

TABLEIDS=bea_tableids

specifies the unique identifier of the table from which the data are retrieved. The available table IDs for the GDPbyIndustry data are listed in [Table 22.17](#). The available table IDs for the UnderlyingGDPbyIndustry data are listed in [Table 22.21](#). The available table IDs for the Input-Output Statistics data are listed in [Table 22.52](#).

TABLENAME=bea_table

TABLE=bea_table

specifies the name of the table from which the data are retrieved. [Table 22.14](#) and [Table 22.15](#) show the list of NIPA and Underlying NIPA table names. Fixed assets table names are shown in [Table 22.22](#). Regional table names are shown in [Table 22.62](#). This is a required option.

TRADEDIRECTION=*bea_tradedirection*

specifies the direction of trade of the data to be retrieved. [Table 22.55](#) shows the list of trade directions and their corresponding descriptions.

TYPEOFINVESTMENT=*bea_typeofinvestment*

specifies the type of investment data to be retrieved. [Table 22.50](#) shows the list of investment types and their corresponding descriptions.

TYPEOFSERVICE=*bea_typeofservice*

specifies the type of service data to be retrieved. [Table 22.56](#) shows the list of service types and their corresponding descriptions.

USER=*'bea_user'***ACCESSKEYID=***'bea_user'*

specifies the unique user access key, enclosed in quotes, that the BEA assigns to the user (subscriber) to allow access to the BEA economic statistics API data service.

YEAR=*bea_years***YEARS=***bea_years*

specifies a list of the desired years for the time series data selection. You can specify *bea_years* as a comma-delimited string enclosed in double quotes. If not specified, the default is set to “LAST5” (last 5 years). You can also specify “LAST10” to select the last ten years, or “ALL” to select all available years.

Details: SASEBEA Interface Engine

The SASEBEA interface engine enables SAS users to access time series data from the BEA data sets shown in [Table 22.2](#). To specify the BEA data set, use one of the supported data set names listed in the table. To access the data set, specify SETNAME=<data set name>, as shown in the SASEBEA LIBNAME statement in [Example 22.1](#). [Table 22.3](#) shows where to find a summary of each data set’s details.

Table 22.2 Supported BEA Data Set Names

Supported Data Set Name	Description and Example Reference
FIXEDASSETS	Fixed Assets; see Example 22.4
GDPBYINDUSTRY	Gross Domestic Product by Industry; see Example 22.2
IIP	International Investment Position (IIP); see Example 22.9
INPUTOUTPUT	Input-Output Statistics; see Example 22.10
INTLSERVTRADE	International Trade in Services; see Example 22.11
ITA	International Transactions Accounts (ITA); see Example 22.7 , Example 22.8
MNE (AMNE)	Activities of Multinational Enterprises (AMNE); see Example 22.6
MNE (DI)	Direct Investment (DI); see Example 22.5
NIPA	National Income and Product Accounts; see Output 22.1
NIPAUNDERLYINGDETAIL	NIPA Underlying Detail; see Example 22.1
REGIONAL	Regional; see Example 22.12
UNDERLYINGGDPBYINDUSTRY	Underlying Gross Domestic Product by Industry; see Example 22.3

Table 22.3 gives a summary of supported options for each set name. See the Section “[Data Elements Reference: SASEBEA Interface Engine](#)” on page 1551 for sample code that shows how to update the information displayed in each option summary table.

Table 22.3 Supported Options Summary for Each BEA Data Set Name

Data Set Name	Option Summary Table
FIXEDASSETS	See Table 22.6
GDPBYINDUSTRY	See Table 22.5
IIP	International Investment Position (IIP); see Table 22.10
INPUTOUTPUT	Input-Output Statistics; see Table 22.11
INTLSERVTRADE	International Trade in Services; see Table 22.12
ITA	International Transactions Accounts (ITA); see Table 22.9
MNE (AMNE)	Activities of Multinational Enterprises (AMNE); see Table 22.8
MNE (DI)	Direct Investment (DI); see Table 22.7
NIPA	See Table 22.4
NIPAUNDERLYINGDETAIL	See Table 22.4
REGIONAL	Regional; see Table 22.13
UNDERLYINGGDPBYINDUSTRY	See Table 22.5

Table 22.4 shows the options specific to the NIPA and NIPAUnderlyingDetail data sets.

Table 22.4 Options for the NIPA and the NIPAUnderlyingDetail Data Sets

Option	Description
FREQ=	Specifies the frequency of the data to retrieve (M, monthly; Q, quarterly; or A, annual)
TABLENAME=	Specifies the name of the data table to retrieve (see Table 22.14 , and Table 22.15)
YEAR=	Specifies a list of years to retrieve data for (ALL for All)

Table 22.5 shows the options specific to the GDPbyIndustry and UnderlyingGDPbyIndustry data sets.

Table 22.5 Options for the GDPbyIndustry and the UnderlyingGDPbyIndustry Data Sets

Option	Description
FREQ=	Specifies the frequency of the data to retrieve (A, annual; or Q, quarterly)
INDUSTRY=	Specifies a list of the industries to retrieve data for (ALL for All) (see Table 22.18 , Table 22.19 , and Table 22.20)
TABLEID=	Specifies the table identifier of the data table to retrieve (ALL for All) (see Table 22.17 , and for underlying data see Table 22.21)
YEAR=	Specifies a list of years to retrieve data for (ALL for All)

Table 22.6 shows the options specific to the FixedAssets data set.

Table 22.6 Options for the FixedAssets Data Set

Option	Description
TABLENAME=	Specifies the name of the data table to retrieve (see Table 22.22)
YEAR=	Specifies a list of years to retrieve data for (ALL for All)

Table 22.7 shows the options specific to the Direct Investment Multinational Enterprises data set.

Table 22.7 Options for the DI Multinational Enterprises Data Set

Option	Description
CLASSIFICATION=	Specifies a data classification based on country and/or industry to retrieve See Table 22.58 for classification names
COUNTRY=	Specifies a list of geographic area codes based on country See Table 22.59 for country codes
DIRECTIONOFINVESTMENT=	Specify 'OUTWARD' for US direct investment abroad, or Specify 'INWARD' for foreign investment in the US

Table 22.7 *continued*

Option	Description
FOOTNOTES=	Specify 'YES' to include footnotes or 'NO' to exclude them
INDUSTRY=	Specifies a list of industry codes based on industry See Table 22.60 for industry codes
SERIESID=	Specifies a list of series IDs to retrieve; optional, 0 means ALL See Table 22.57 for available series IDs
YEAR=	Specifies a list of years to retrieve data for (ALL for All)

Table 22.8 shows the options specific to the Activities of Multinational Enterprises (AMNE) data set.

Table 22.8 Options for the Activities of Multinational Enterprises (AMNE) Data Set

Option	Description
CLASSIFICATION=	Specifies a data classification based on country and/or industry to retrieve. See Table 22.58 for classification names.
COUNTRY=	Specifies a list of geographic area codes based on country. See Table 22.59 for country codes.
DIRECTIONOFINVESTMENT=	Specify 'OUTWARD' for US direct investment abroad, or specify 'INWARD' for foreign investment in the US Specify 'PARENT' for US parent companies. Specify 'STATE' for US affiliates of foreign multinational companies at the state level.
FOOTNOTES=	Specify 'YES' to include footnotes or 'NO' to exclude them.
INDUSTRY=	Specifies a list of industry codes based on industry. See Table 22.60 for industry codes.
NONBANKAFFILIATESONLY=	Specify 0 for Both Bank and Nonbank Affiliates, or 1 for Nonbank Affiliates.
OWNERSHIPLEVEL=	Specify 0 for Majority-Owned Affiliates or 1 for All Affiliates.
SERIESID=	Specifies a list of series IDs to retrieve; optional; 0 means ALL. See Table 22.57 for available series IDs.
STATE=	Specifies a list of years to retrieve data for (ALL for All). See Table 22.61 for state codes.
YEAR=	Specifies a list of years to retrieve data for (ALL for All).

Table 22.9 shows the options specific to the International Transactions (ITA) data set.

Table 22.9 Options for the International Transactions Accounts (ITA) Data Set

Option	Description
INDICATOR=	Specifies the indicator code for the type of transaction requested. See the tables at the following URL: https://www.bea.gov/iTable/iTableHtml.cfm?reqid=62&step=2&isuri=1&6210=1
COUNTRY=	Specifies a list of the counterparty areas or countries of the transactions. See Table 22.48 for ITA country codes.
FREQ=	Specifies a frequency with which to select data (A, annual; QNSA, quarterly not seasonally adjusted; or QSA, quarterly seasonally adjusted).
YEAR=	Specifies a list of years to retrieve data for (ALL for All).

Table 22.10 shows the options specific to the International Investment Position (IIP) data set.

Table 22.10 Options for the International Investment Position (IIP) Data Set

Option	Description
COMPONENT=	Specifies the component or components requested (listed in Table 22.51)
FREQ=	Specifies a frequency at which to select data (A, annual; or QNSA, quarterly not seasonally adjusted). When quarterly data are requested, all available quarters of the specified years are returned.
TYPEOFINVESTMENT=	Specifies the type of investment requested (listed in Table 22.50). For more information, see the tables at the following URL: https://apps.bea.gov/iTable/iTable.cfm?0=1&isuri=1&reqid=62&step=2#0=1&isuri=1&reqid=62&step=2
YEAR=	More than one type of investment can be specified if only one year is specified. If more than one year is specified, only one type of investment can be specified. Specifies a list of years to retrieve data for (ALL for All) If more than one year is requested, then only one type of investment request is allowed.

Table 22.11 shows the options specific to the Input-Output Statistics data set.

Table 22.11 Options for the Input-Output Statistics Data Set

Option	Description
TABLEID=	Specifies the table ID (listed in Table 22.52). For more information, visit the web page at the following URL: https://www.bea.gov/data/industries/input-output-accounts-data
YEAR=	Specifies a list of years to retrieve data for (ALL for All)

Table 22.12 shows the options specific to the International Trade in Services (IntlServTrade) data set.

Table 22.12 Options for the International Trade in Services (IntlServTrade) Data Set

Option	Description
AFFILIATION=	Specifies the affiliation of the data to retrieve (listed in Table 22.54). By default, AFFILIATION=All.
AREAORCOUNTRY=	Specifies the area or country of the data to retrieve (listed in Table 22.53). By default, AREAORCOUNTRY=AllCountries. A list of countries can be specified only if a single type of service is specified.
DESCRIPTION=	Specifies whether to use the short or the long label. The short label matches the BEA time series indicator, and the long label matches the BEA time series description. By default, DESCRIPTION=SHORT.
TRADEDIRECTION=	Specifies the trade direction of the data to retrieve (listed in Table 22.55). By default, TRADEDIRECTION=All.
TYPEOFSERVICE=	Specifies the type of service of the data to retrieve (listed in Table 22.56). By default, TYPEOFSERVICE=All. More than one type of service can be specified only if a single area or country is specified.
YEAR=	Specifies a list of years to retrieve data for (ALL for All). By default, YEAR=All.

Table 22.13 shows the options specific to the Regional data set.

Table 22.13 Options for the Regional Data Set

Option	Description
GEOFIPS=	Specifies the state, county, MSA, or other code for selecting the data to retrieve. It can be a list of codes, separated by commas. “ALL” values can be one of the following: “STATE”, “COUNTY”, “MSA”, “MIC”, “PORT”, “DIV”, “CSA”, or “PO code”. GEOFIPS= is a required option.
LINECODE=	Specifies one line code to select a corresponding statistic of the Regional data to retrieve. LINECODE= is a required option.
TABlename=	Specifies a published table name from the regional accounts that designates the data to retrieve (listed in Table 22.62). TABlename= is a required option.
YEAR=	Specifies a list of years to retrieve data for, or can be one of the following: “ALL” for all available years, “LAST10” for last 10 years, or “LAST5” for last 5 years. By default, YEAR=LAST5. The YEAR= option is not required.

The National Income and Product Accounts

The table names available for the NIPA data set are listed in [Table 22.14](#) and [Table 22.15](#). Each table is identified by a unique string called the table name. The TABLENAME= option is a required option that specifies which table to retrieve. Historically, the NIPA tables were identified by a table ID, but this practice was discontinued in October 2018. For convenience, [Table 22.14](#) and [Table 22.15](#) show how table names correspond to table IDs.

Table 22.14 NIPA Table Names and Corresponding Table IDs

TableName	TableId	TableName	TableId	TableName	TableId	TableName	TableId
T10101	1	T10803	45	T30200	87	T40203	131
T10102	2	T10806	46	T30300	88	T40204	132
T10103	3	T10903	47	T30400	89	T40205	133
T10104	4	T10904	48	T30500	90	T40206	134
T10105	5	T10905	49	T30600	91	T4030A	135
T10106	6	T10906	50	T30700	92	T4030B	136
T10106A	7	T11000	51	T30800	93	T50100	137
T10106B	8	T11100	52	T30901	94	T50203	138
T10106C	9	T11200	53	T30902	95	T50205	139
T10106D	10	T11300	54	T30903	96	T50206	140
T10107	11	T11400	55	T30904	97	T50301	141
T10108	12	T11500	56	T30905	98	T50302	142
T10109	13	T11600	57	T30906	99	T50303	143
T10110	14	T11701	316	T31001	100	T50304	144
T10111	310	T11705	317	T31003	101	T50305	145
T10201	15	T11706	318	T31004	102	T50306	146
T10202	16	T20100	58	T31005	103	T50401	147
T10203	17	T20200A	59	T31006	104	T50402	148
T10204	18	T20200B	60	T31101	105	T50403	149
T10205	19	T20301	61	T31102	326	T50404	150
T10206	20	T20302	62	T31103	106	T50405	151
T10301	21	T20303	63	T31104	107	T50406	152
T10303	22	T20304	64	T31105	108	T50501	153
T10304	23	T20305	65	T31106	109	T50502	154
T10305	24	T20306	66	T31200	110	T50503	155
T10306	25	T20307	67	T31300	111	T50504	156
T10401	26	T20403	68	T31400	112	T50505	157
T10403	27	T20404	69	T31501	113	T50506	158
T10404	28	T20405	70	T31502	114	T50601	327
T10405	29	T20406	71	T31503	115	T50602	328
T10406	30	T20503	72	T31504	116	T50603	329
T10501	31	T20504	73	T31505	117	T50604	330
T10502	32	T20505	74	T31506	118	T50605	331
T10503	33	T20506	75	T31600	119	T50606	332
T10504	34	T20600	76	T31700	120	T50705A	163
T10505	35	T20700A	77	T31800A	121	T50705B	164

Table 22.14 *continued*

TableName	TableId	TableName	TableId	TableName	TableId	TableName	TableId
T10506	36	T20700B	78	T31800B	122	T50706A	165
T10604	37	T20801	79	T31900	123	T50706B	166
T10607	38	T20803	80	T32000	124	T50805A	173
T10608	39	T20804	81	T32100	125	T50805B	174
T10701	40	T20805	82	T32200	126	T50806A	175
T10703	41	T20806	83	T32300	127	T50806B	176
T10704	42	T20807	84	T40100	128	T50809A	333
T10705	43	T20900	85	T40201	129	T50809B	334
T10706	44	T30100	86	T40202	130	T50903A	335

Table 22.15 NIPA Table Names and Corresponding Table IDs

TableName	TableId	TableName	TableId	TableName	TableId
T50903B	336	T61100A	216	T62200B	261
T50904A	337	T61100B	217	T62200C	262
T50904B	338	T61100C	218	T62200D	263
T50905A	339	T61100D	219	T70100	264
T50905B	340	T61200A	220	T70201A	265
T50906B	342	T61200B	221	T70201B	266
T51000	178	T61200C	222	T70203A	267
T51100	343	T61200D	223	T70203B	268
T60100B	179	T61300A	224	T70204A	269
T60100C	180	T61300B	225	T70204B	270
T60100D	181	T61300C	226	T70205A	271
T60200A	182	T61300D	227	T70205B	272
T60200B	183	T61400A	228	T70206B	273
T60200C	184	T61400B	229	T70303	274
T60200D	185	T61400C	230	T70304	275
T60300A	186	T61400D	231	T70305	276
T60300B	187	T61500A	232	T70306	277
T60300C	188	T61500B	233	T70403	278
T60300D	189	T61500C	234	T70404	279
T60400A	190	T61500D	235	T70405	280
T60400B	191	T61600A	236	T70406	281
T60400C	192	T61600B	237	T70500	282
T60400D	193	T61600C	238	T70600	283
T60500A	194	T61600D	239	T70700	284
T60500B	195	T61700A	240	T70800	285
T60500C	196	T61700B	241	T70900	286
T60500D	197	T61700C	242	T71000	287
T60600A	198	T61700D	243	T71100	288
T60600B	199	T61800A	244	T71200	289
T60600C	200	T61800B	245	T71300	290
T60600D	201	T61800C	246	T71400	291

Table 22.15 *continued*

TableName	TableId	TableName	TableId	TableName	TableId
T60700A	202	T61800D	247	T71500	292
T60700B	203	T61900A	248	T71600	293
T60700C	204	T61900B	249	T71700	294
T60700D	205	T61900C	250	T71800	295
T60800A	206	T61900D	251	T71900	297
T60800B	207	T62000A	252	T72000	296
T60800C	208	T62000B	253	T72100	392
T60800D	209	T62000C	254	T72200	393
T60900B	210	T62000D	255	T72300	394
T60900C	211	T62100A	256	T72400	395
T60900D	212	T62100B	257	T72500	397
T61000B	213	T62100C	258		
T61000C	214	T62100D	259		
T61000D	215	T62200A	260		

NIPA Underlying Detail National Income and Product Accounts

The table names available for the NIPA Underlying Detail data set are listed in Table 22.16. Each table is identified by a unique string called the table name. The TABLENAME= option is a required option that specifies which table to retrieve. Historically, the NIPA tables were identified by a table ID, but this practice was discontinued in October 2018. For convenience, Table 22.16 shows how table names correspond to table IDs.

Table 22.16 NIPA Table Names and Corresponding Table IDs

TableName	TableId	TableName	TableId	TableName	TableId	TableName	TableId
U001A	2001	U20305	2014	U50203	2070	U50705BU3	2092
U001A2	2012	U20306	2015	U50205	2071	U50706AM	2093
U001B	2023	U20403	2074	U50206	2072	U50706AU	2094
U001BC	2034	U20404	2016	U50404	2029	U50706BM	2095
U002AU	2045	U20405	2017	U50405	2030	U50706BU	2096
U002AUI	2056	U20406	2018	U50406	2031	U51100	2053
U002BU	2059	U30400	2019	U50504	2032	U70204	2054
U002BUI	2060	U30500	2020	U50505	2033	U70205	2057
U003AU	2061	U20600	2021	U50506	2035	U70205S	2055
U003BU	2002	U30700	2022	U50705AM1	2081	U70206	2058
U004A1	2003	U30800	2024	U50705AM2	2082	U90100	2075
U004A2	2004	U31200	2025	U50705AM3	2083	U90200	2076
U004A3	2005	U31300	2026	U50705AU1	2084	U50300	2077
U004B1	2006	U32400	2027	U50705AU2	2085	U90400	2078
U004B2	2007	U32500	2028	U50705AU3	2086	U90500	2079
U004B3	2008	U40203	2114	U50705BM1	2087		
U00500	2009	U40204	2111	U50705BM2	2088		
U00600	2010	U40205	2113	U50705BM3	2089		
U00700	2011	U40206	2112	U50705BU1	2090		
U20304	2013	U4030B	2073	U50705BU2	2091		

The Gross Domestic Product by Industry Data

The table names available for the GDPbyIndustry data set are listed in Table 22.17. Each table is identified by a unique table ID code.

Table 22.17 GDP Table IDs and Descriptions

TableId	Description
1	Value Added by Industry (A)(Q)
5	Value Added by Industry as a Percentage of Gross Domestic Product (A)(Q)
6	Components of Value Added by Industry (A)
7	Components of Value Added by Industry as a Percentage of Gross Domestic Product (A)
8	Chain-Type Quantity Indexes for Value Added by Industry (A)(Q)
9	Percent Changes in Chain-Type Quantity Indexes for Value Added by Industry (A)(Q)
10	Real Value Added by Industry (A)(Q)
11	Chain-Type Price Indexes for Value Added by Industry (A)(Q)
12	Percent Changes in Chain-Type Price Indexes for Value Added by Industry (A)(Q)
13	Contributions to Percent Change in Real Gross Domestic Product by Industry (A)(Q)
14	Contributions to Percent Change in the Chain-Type Price Indexes for Gross Domestic Product by Industry (A)(Q)
15	Gross Output by Industry (A)(Q)
16	Chain-Type Quantity Indexes for Gross Output by Industry (A)(Q)
17	Percent Changes in Chain-Type Quantity Indexes for Gross Output by Industry (A)(Q)
18	Chain-Type Price Indexes for Gross Output by Industry (A)(Q)
19	Percent Changes in Chain-Type Price Indexes for Gross Output by Industry (A)(Q)
20	Intermediate Inputs by Industry (A)(Q)
21	Chain-Type Quantity Indexes for Intermediate Inputs by Industry (A)(Q)
22	Percent Changes in Chain-Type Quantity Indexes for Intermediate Inputs by Industry (A)(Q)
23	Chain-Type Price Indexes for Intermediate Inputs by Industry (A)(Q)
24	Percent Changes in Chain-Type Price Indexes for Intermediate Inputs by Industry (A)(Q)
25	Composition of Gross Output by Industry (A)
26	Shares of Gross Output by Industry (A)
27	Cost per Unit of Real Gross Output by Industry Group (A)
29	Contributions to Percent Change in the Chain-Type Quantity Indexes for Gross Output by Industry Group(A)
30	Contributions to Percent Change in the Chain-Type Price Indexes for Gross Output by Industry Group(A)

Table 22.17 *continued*

TableId	Description
31	Chain-Type Quantity Indexes for Energy Inputs by Industry (A)
32	Contributions to Percent Change by Industry in the Chain-Type Quantity Index for All Industries Energy Inputs (A)
33	Chain-Type Price Indexes for Energy Inputs by Industry (A)
34	Contributions to Percent Change by Industry in the Chain-Type Price Index for All Industries Energy Inputs (A)
35	Chain-Type Quantity Indexes for Materials Inputs by Industry (A)
36	Contributions to Percent Change by Industry in the Chain-Type Quantity Index for All Industries Material Inputs (A)
37	Chain-Type Price Indexes for Materials Inputs by Industry (A)
38	Contributions to Percent Change by Industry in the Chain-Type Price Index for All Industries Material Inputs (A)
39	Chain-Type Quantity Indexes for Purchased Service Inputs by Industry (A)
40	Contributions to Percent Change by Industry in the Chain-Type Quantity Index for All Industries Purchased Service Inputs (A)
41	Chain-Type Price Indexes for Purchased Service Inputs by Industry (A)
42	Contributions to Percent Change by Industry in the Chain-Type Price Index for All Industries Purchased Service Inputs (A)
208	Real Gross Output by Industry (A)(Q)
209	Real Intermediate Inputs by Industry (A)(Q)

The industries available for the GDPbyIndustry data set are listed in [Table 22.18](#), [Table 22.19](#), and [Table 22.20](#). Each industry is identified by a unique industry code.

Table 22.18 GDP Industries and Descriptions

Industry Code	Description
GDP	Gross domestic product (A,Q)
II	All industries (A,Q)
PVT	Private industries (A,Q)
11	Agriculture, forestry, fishing, and hunting (A,Q)
111CA	Farms (A,Q)
113FF	Forestry, fishing, and related activities (A,Q)
21	Mining (A,Q)
211	Oil and gas extraction (A,Q)
212	Mining, except oil and gas (A,Q)
213	Support activities for mining (A,Q)
22	Utilities (A,Q)
23	Construction (A,Q)
31G	Manufacturing (A,Q)
33DG	Durable goods (A,Q)
321	Wood products (A,Q)
327	Nonmetallic mineral products (A,Q)

Table 22.18 *continued*

Industry Code	Description
331	Primary metals (A,Q)
332	Fabricated metal products (A,Q)
333	Machinery (A,Q)
334	Computer and electronic products (A,Q)
335	Electrical equipment, appliances, and components (A,Q)
3361MV	Motor vehicles, bodies and trailers, and parts (A,Q)
3364OT	Other transportation equipment (A,Q)
337	Furniture and related products (A,Q)
339	Miscellaneous manufacturing (A,Q)
31ND	Nondurable goods (A,Q)
311FT	Food and beverage and tobacco products (A,Q)
313TT	Textile mills and textile product mills (A,Q)
315AL	Apparel and leather and allied products (A,Q)
322	Paper products (A,Q)
323	Printing and related support activities (A,Q)
324	Petroleum and coal products (A,Q)
325	Chemical products (A,Q)
326	Plastics and rubber products (A,Q)
42	Wholesale trade (A,Q)
44RT	Retail trade (A,Q)
441	Motor vehicle and parts dealers (A,Q)
445	Food and beverage stores (A,Q)
452	General merchandise stores (A,Q)
4A0	Other retail (A,Q)
48TW	Transportation and warehousing (A,Q)
481	Air transportation (A,Q)
482	Rail transportation (A,Q)
483	Water transportation (A,Q)
484	Truck transportation (A,Q)

Table 22.19 GDP Industries and Descriptions

Industry Code	Description
485	Transit and ground passenger transportation (A,Q)
486	Pipeline transportation (A,Q)
487OS	Other transportation and support activities (A,Q)
493	Warehousing and storage (A,Q)
51	Information (A,Q)
511	Publishing industries, except internet (includes software) (A,Q)
512	Motion picture and sound recording industries (A,Q)
513	Broadcasting and telecommunications (A,Q)
514	Data processing, internet publishing, and other information services (A,Q)

Table 22.19 *continued*

Industry Code	Description
FIRE	Finance, insurance, real estate, rental, and leasing (A,Q)
52	Finance and insurance (A,Q)
521CI	Federal Reserve banks, credit intermediation, and related activities (A,Q)
523	Securities, commodity contracts, and investments (A,Q)
524	Insurance carriers and related activities (A,Q)
525	Funds, trusts, and other financial vehicles (A,Q)
53	Real estate and rental and leasing (A,Q)
531	Real estate (A,Q)
HS	Housing (A,Q)
ORE	Other real estate (A,Q)
532RL	Rental and leasing services and lessors of intangible assets (A,Q)
PROF	Professional and business services (A,Q)
54	Professional, scientific, and technical services (A,Q)
5411	Legal services (A,Q)
5415	Computer systems design and related services (A,Q)
5412OP	Miscellaneous professional, scientific, and technical services (A,Q)
55	Management of companies and enterprises (A,Q)
56	Administrative and waste management services (A,Q)
561	Administrative and support services (A,Q)
562	Waste management and remediation services (A,Q)
6	Educational services, health care, and social assistance (A,Q)
61	Educational services (A,Q)
62	Health care and social assistance (A,Q)
621	Ambulatory health care services (A,Q)
622	Hospitals (A,Q)
623	Nursing and residential care facilities (A,Q)
624	Social assistance (A,Q)
7	Arts, entertainment, recreation, accommodation, and food services (A,Q)
71	Arts, entertainment, and recreation (A,Q)

Table 22.20 GDP Industries and Descriptions

Industry Code	Description
711AS	Performing arts, spectator sports, museums, and related activities (A,Q)
713	Amusements, gambling, and recreation industries (A,Q)
72	Accommodation and food services (A,Q)
721	Accommodation (A,Q)
722	Food services and drinking places (A,Q)
81	Other services, except government (A,Q)
G	Government (A,Q)
GF	Federal (A,Q)
GFG	General government (A,Q)

Table 22.20 *continued*

Industry Code	Description
GFGD	National defense (A,Q)
GFGN	Nondefense (A,Q)
GFE	Government enterprises (A,Q)
GSL	State and local (A,Q)
GSLG	General government (A,Q)
GSLE	Government enterprises (A,Q)
NABI	Not allocated by industry [1] (A,Q)
PGOOD	Private goods-producing industries [2] (A,Q)
PSERV	Private services-producing industries [3] (A,Q)
ICT	Information-communications-technology-producing industries [4] (A)

The Underlying Gross Domestic Product by Industry Data

The table names available for the UnderlyingGDPbyIndustry data set are listed in [Table 22.21](#). Each table is identified by a unique table ID code.

Table 22.21 Underlying GDP Table IDs and Descriptions

TableId	Description
210	U.Value Added by Industry (A)
211	U.Value Added by Industry as a Percentage of Gross Domestic Product (A)
212	U.Chain-Type Quantity Indexes for Value Added by Industry (A)
213	U.Percent Changes in Chain-Type Quantity Indexes for Value Added by Industry (A)
214	U.Real Value Added by Industry (A)
215	U.Chain-Type Price Indexes for Value Added by Industry (A)
216	U.Percent Changes in Chain-Type Price Indexes for Value Added by Industry (A)
217	U.Contributions to Percent Change in Real Gross Domestic Product by Industry (A)
218	U.Contributions to Percent Change in the Chain-Type Price Index for GDP (A)
219	U.Gross Output by Industry (A)
221	U.Chain-Type Quantity Indexes for Gross Output by Industry (A)
223	U.Percent Changes in Chain-Type Quantity Indexes for Gross Output by Industry (A)
225	U.Chain-Type Price Indexes for Gross Output by Industry (A)
227	U.Percent Changes in Chain-Type Price Indexes for Gross Output by Industry (A)
229	U.Intermediate Inputs by Industry (A)
230	U.Chain-Type Quantity Indexes for Intermediate Inputs by Industry (A)
231	U.Percent Changes in Chain-Type Quantity Indexes for Intermediate Inputs by Industry (A)
232	U.Chain-Type Price Indexes for Intermediate Inputs by Industry (A)
233	U.Percent Changes in Chain-Type Price Indexes for Intermediate Inputs by Industry (A)
234	U.Real Gross Output by Industry (A)
236	U.Real Intermediate Inputs by Industry (A)

Fixed Assets Data

The table names available for the FixedAssets data set are listed in [Table 22.22](#). Each table is identified by a unique table name.

Table 22.22 Fixed Assets Table Names and Descriptions

TableName	Description
FAAt101	Table 1.1. Current-Cost Net Stock of Fixed Assets and Consumer Durable Goods
FAAt102	Table 1.2. Chain-Type Quantity Indexes for Net Stock of Fixed Assets and Consumer Durable Goods
FAAt103	Table 1.3. Current-Cost Depreciation of Fixed Assets and Consumer Durable Goods
FAAt104	Table 1.4. Chain-Type Quantity Indexes for Depreciation of Fixed Assets and Consumer Durable Goods
FAAt105	Table 1.5. Investment in Fixed Assets and Consumer Durable Goods
FAAt106	Table 1.6. Chain-Type Quantity Indexes for Investment in Fixed Assets and Consumer Durable Goods
FAAt107	Table 1.7. Current-Cost Other Changes in Volume of Assets for Fixed Assets and Consumer Durable Goods
FAAt108	Table 1.8. Historical-Cost Other Changes in Volume of Assets for Fixed Assets and Consumer Durable Goods
FAAt109	Table 1.9. Current-Cost Average Age at Year End of Fixed Assets and Consumer Durable Goods
FAAt201	Table 2.1. Current-Cost Net Stock of Private Fixed Assets, Equipment, Structures, and Intellectual Property Products by Type
FAAt202	Table 2.2. Chain-Type Quantity Indexes for Net Stock of Private Fixed Assets, Equipment, Structures, and Intellectual Property Product by Type
FAAt203	Table 2.3. Historical-Cost Net Stock of Private Fixed Assets, Equipment, Structures, and Intellectual Property Product by Type
FAAt204	Table 2.4. Current-Cost Depreciation of Private Fixed Assets, Equipment, Structures, and Intellectual Property Product by Type
FAAt205	Table 2.5. Chain-Type Quantity Indexes for Depreciation of Private Fixed Assets, Equipment, Structures, and Intellectual Property Product by Type
FAAt206	Table 2.6. Historical-Cost Depreciation of Private Fixed Assets, Equipment, Structures, and Intellectual Property Products by Type
FAAt207	Table 2.7. Investment in Private Fixed Assets, Equipment, Structures, and Intellectual Property Products by Type
FAAt208	Table 2.8. Chain-Type Quantity Indexes for Investment in Private Fixed Assets, Equipment, Structures, and Intellectual Property Products by Type
FAAt209	Table 2.9. Current-Cost Average Age at Year End of Private Fixed Assets, Equipment, Structures, and Intellectual Property Products by Type
FAAt210	Table 2.10. Historical-Cost Average Age at Year End of Private Fixed Assets, Equipment, Structures, and Intellectual Property Products by Type
FAAt301E	Table 3.1E. Current-Cost Net Stock of Private Equipment by Industry
FAAt301ESI	Table 3.1ESI. Current-Cost Net Stock of Private Fixed Assets by Industry

Table 22.22 *continued*

TableName	Description
FAAt301I	Table 3.1I. Current-Cost Net Stock of Intellectual Property Products by Industry
FAAt301S	Table 3.1S. Current-Cost Net Stock of Private Structures by Industry
FAAt302E	Table 3.2E. Chain-Type Quantity Indexes for Net Stock of Private Equipment by Industry
FAAt302ESI	Table 3.2ESI. Chain-Type Quantity Indexes for Net Stock of Private Fixed Assets by Industry
FAAt302I	Table 3.2I. Chain-Type Quantity Indexes for Net Stock of Intellectual Property Products by Industry
FAAt302S	Table 3.2S. Chain-Type Quantity Indexes for Net Stock of Private Structures by Industry
FAAt303E	Table 3.3E. Historical-Cost Net Stock of Private Equipment by Industry
FAAt303ESI	Table 3.3ESI. Historical-Cost Net Stock of Private Fixed Assets by Industry
FAAt303I	Table 3.3I. Historical-Cost Net Stock of Private Intellectual Property Products by Industry
FAAt303S	Table 3.3S. Historical-Cost Net Stock of Private Structures by Industry
FAAt304E	Table 3.4E. Current-Cost Depreciation of Private Equipment by Industry
FAAt304ESI	Table 3.4ESI. Current-Cost Depreciation of Private Fixed Assets by Industry
FAAt304I	Table 3.4I. Current-Cost Depreciation of Private Intellectual Property Products by Industry
FAAt304S	Table 3.4S. Current-Cost Depreciation of Private Structures by Industry
FAAt305E	Table 3.5E. Chain-Type Quantity Indexes for Depreciation of Private Equipment by Industry
FAAt305ESI	Table 3.5ESI. Chain-Type Quantity Indexes for Depreciation of Private Fixed Assets by Industry
FAAt305I	Table 3.5I. Chain-Type Quantity Indexes for Depreciation of Private Intellectual Property
FAAt305S	Table 3.5S. Chain-Type Quantity Indexes for Depreciation of Private Structures by Industry
FAAt306E	Table 3.6E. Historical-Cost Depreciation of Private Equipment by Industry
FAAt306ESI	Table 3.6ESI. Historical-Cost Depreciation of Private Fixed Assets by Industry
FAAt306I	Table 3.6I. Historical-Cost Depreciation of Private Intellectual Property Products by Industry
FAAt306S	Table 3.6S. Historical-Cost Depreciation of Private Structures by Industry
FAAt307E	Table 3.7E. Investment in Private Equipment by Industry
FAAt307ESI	Table 3.7ESI. Investment in Private Fixed Assets by Industry
FAAt307I	Table 3.7I. Investment in Private Intellectual Property Products by Industry
FAAt307S	Table 3.7S. Investment in Private Structures by Industry
FAAt308E	Table 3.8E. Chain-Type Quantity Indexes for Investment in Private Equipment by Industry
FAAt308ESI	Table 3.8ESI. Chain-Type Quantity Indexes for

Table 22.22 *continued*

TableName	Description
FAAt308I	Investment in Private Fixed Assets by Industry Table 3.8I. Chain-Type Quantity Indexes for Investment in Private Intellectual Property
FAAt308S	Table 3.8S. Chain-Type Quantity Indexes for Investment in Private Structures by Industry
FAAt309E	Table 3.9E. Current-Cost Average Age at Year End of Private Equipment by Industry
FAAt309ESI	Table 3.9ESI. Current-Cost Average Age at Year End of Private Fixed Assets by Industry
FAAt309I	Table 3.9I. Current-Cost Average Age at Year End of Private Intellectual Property Products by Industry
FAAt309S	Table 3.9S. Current-Cost Average Age at Year End of Private Structures by Industry
FAAt310E	Table 3.10E. Historical-Cost Average Age at Year End of Private Equipment by Industry
FAAt310ESI	Table 3.10ESI. Historical-Cost Average Age at Year End of Private Fixed Assets by Industry
FAAt310I	Table 3.10I. Historical-Cost Average Age at Year End of Private Intellectual Property Products by Industry
FAAt310S	Table 3.10. Historical-Cost Average Age at Year End of Private Structures by Industry
FAAt401	Table 4.1. Current-Cost Net Stock of Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization
FAAt402	Table 4.2. Chain-Type Quantity Indexes for Net Stock of Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization
FAAt403	Table 4.3. Historical-Cost Net Stock of Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization
FAAt404	Table 4.4. Current-Cost Depreciation of Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization
FAAt405	Table 4.5. Chain-Type Quantity Indexes for Depreciation of Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization
FAAt406	Table 4.6. Historical-Cost Depreciation of Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization
FAAt407	Table 4.7. Investment in Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization
FAAt408	Table 4.8. Chain-Type Quantity Indexes for Investment in Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization
FAAt409	Table 4.9. Current-Cost Average Age at Year End of Private Nonresidential Fixed Assets by Industry Group and Legal Form of Organization

Table 22.22 *continued*

TableName	Description
FAAt410	Table 4.10. Historical-Cost Average Age at Year End of Private Nonresidential Fixed Assets by Industry and Legal Form of Organization
FAAt501	Table 5.1. Current-Cost Net Stock of Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt502	Table 5.2. Chain-Type Quantity Indexes for Net Stock of Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt503	Table 5.3. Historical-Cost Net Stock of Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt504	Table 5.4. Current-Cost Depreciation of Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt505	Table 5.5. Chain-Type Quantity Indexes for Depreciation of Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt506	Table 5.6. Historical-Cost Depreciation of Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt507	Table 5.7. Investment in Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt508	Table 5.8. Chain-Type Quantity Indexes for Investment in Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt509	Table 5.9. Current-Cost Average Age at Year End of Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt510	Table 5.10. Historical-Cost Average Age at Year End of Residential Fixed Assets by Type of Owner, Legal Form of Organization, and Tenure Group
FAAt601	Table 6.1. Current-Cost Net Stock of Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt602	Table 6.2. Chain-type Quantity Indexes for Net Stock of Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt603	Table 6.3. Historical-Cost Net Stock of Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt604	Table 6.4. Current-Cost Depreciation of Private Fixed

Table 22.22 *continued*

TableName	Description
	Assets by Industry Group and Legal Form of Organization
FAAt605	Table 6.5. Chain-Type Quantity Indexes for Depreciation of Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt606	Table 6.6. Historical-Cost Depreciation of Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt607	Table 6.7. Investment in Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt608	Table 6.8. Chain-Type Quantity Indexes for Investment in Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt609	Table 6.9. Current-Cost Average Age at Year End of Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt610	Table 6.10. Historical-Cost Average Age at Year End of Private Fixed Assets by Industry Group and Legal Form of Organization
FAAt701	Table 7.1. Current-Cost Net Stock of Government Fixed Assets
FAAt702	Table 7.2. Chain-Type Quantity Indexes for Net Stock of Government Fixed Assets
FAAt703	Table 7.3. Current-Cost Depreciation of Government Fixed Assets
FAAt704	Table 7.4. Chain-Type Quantity Indexes for Depreciation of Government Fixed Assets
FAAt705	Table 7.5. Investment in Government Fixed Assets
FAAt706	Table 7.6. Chain-Type Quantity Indexes for Investment in Government Fixed Assets
FAAt707	Table 7.7. Current-Cost Average Age at Year End of Government Fixed Assets
FAAt801	Table 8.1. Current-Cost Net Stock of Consumer Durable Goods
FAAt802	Table 8.2. Chain-Type Quantity Indexes for Net Stock of Consumer Durable Goods
FAAt803	Table 8.3. Historical-Cost Net Stock of Consumer Durable Goods
FAAt804	Table 8.4. Current-Cost Depreciation of Consumer Durable Goods
FAAt805	Table 8.5. Chain-Type Quantity Indexes for Depreciation of Consumer Durable Goods
FAAt806	Table 8.6. Historical-Cost Depreciation of Consumer Durable Goods
FAAt807	Table 8.7. Investment in Consumer Durable Goods
FAAt808	Table 8.8. Chain-Type Quantity Indexes for Investment

Table 22.22 *continued*

TableName	Description
	in Consumer Durable Goods
FAAt809	Table 8.9. Current-Cost Average Age at Year End of Consumer Durable Goods
FAAt810	Table 8.10. Historical-Cost Average Age at Year End of Consumer Durable Goods
FAAt901	Table 9.1. Real Net Stock of Fixed Assets and Consumer Durable Goods
FAAt902	Table 9.2. Real Depreciation of Fixed Assets and Consumer Durable Goods
FAAt903	Table 9.3. Real Investment in Fixed Assets and Consumer Durable Goods

International Transactions Accounts Data

The indicators available for the International Transactions Accounts (ITA) data set can be found by using the lookup index found in [Table 22.23](#). Click the desired table reference in the lookup table to display a list of indicators. The selected indicator appears in alphabetical order in the list.

There is a one-to-many relationship between the indicator name and the time series names that make up the components of each indicator. Most time series names contain the indicator name and are made up of the following concatenations, in this order: 'TSI_Ita', indicator, area or country code, '_', and frequency code.

Table 22.23 Lookup Index for ITA Series Indicators

Lookup Table	First Indicator – Last Indicator
Table 22.24	BalCapAcct – CurrLiabsCentralBank
Table 22.25	DebtSecAssets – DebtSecLiabsGenGovt
Table 22.26	DebtSecLiabsNonFin – DepResaleAssetsDepTaking
Table 22.27	DiInvAssets – DiInvDebtInstOutwardHoldExcBank
Table 22.28	DiInvDebtInstOutwardMnfctr – DiInvEquityOthThanReinvestEarnAssetsWhlsITrd
Table 22.29	DiInvEquityOthThanReinvestEarnDecAssets – DiInvIntIncOutward
Table 22.30	DiInvIntIncPay – DiInvIntIncOutward
Table 22.31	DiInvReinvestEarnWithoutCurrCostAdjLiabsFinAndIns – DiInvWithoutCurrCostAdjIncOutwardWhlsITrd
Table 22.32	DiInvWithoutCurrCostAdjInward – DiInvWithoutCurrCostAdjOutwardWhlsITrd
Table 22.33	EquityAndInvFundSharesAssets – EquityOthThanInvFundSharesLiabsFoa
Table 22.34	ExpGds – ExpGdsNonmonGoldBopAdj
Table 22.35	ExpGdsNuclearFuelAndElecEnergy – ExpServAudVis
Table 22.36	ExpServChargesForTheUseOfIpNie – ExpServTravel
Table 22.37	ExpServTravelBusiness – ImpGdsJewelryAndCollect
Table 22.38	ImpGdsLiqPetGases – ImpServAudVis
Table 22.39	ImpServChargesForTheUseOfIpNie – ImpServTransportOth
Table 22.40	ImpServTransportSea – LoansResaleAssetsOthFinNonFin
Table 22.41	LtDebtSecAssets – LtDebtSecOthThanFedSponsorAgencyLiabsOthFin
Table 22.42	LtDebtSecStateLocalGovtIncPay – LtTrdCredAndAdvLiabsOthFinNonFin
Table 22.43	LtTrdCredAndAdvLiabsOthFinNonFin – OthInvInsPolHoldtIncRecOthFinNonFin
Table 22.44	OthInvInterbankAssets – PrimIncRec
Table 22.45	ReserveAssets – SecIncRecPrivateFinesPenalties
Table 22.46	SecIncRecUsGovt – StDebtSecOthThanFedSponsorAgencyLiabsOthFin
Table 22.47	StDebtSecTreasIncPay – TSI_ItaDiInvWithoutCurrCostAdjIncOutward

Table 22.24 ITA Series Indicators and Descriptions

Series Indicator	Description
BalCapAcct	Balance on capital account
BalCurrAcct	Balance on current account
BalGds	Balance on goods
BalGdsServ	Balance on goods and services
BalPrimInc	Balance on primary income
BalSecInc	Balance on secondary income (current transfers)
BalServ	Balance on services
CapTransPayAndOthDeb	Capital transfer payments and other debits
CapTransRecAndOthCred	Capital transfer receipts and other credits
CompOfEmplPay	Compensation of employees; payments
CompOfEmplRec	Compensation of employees; receipts
CurrAndDepAssets	Net US acquisition of other investment assets; currency and deposits
CurrAndDepAssetsCentralBank	Net US acquisition of other investment assets; currency and deposits; held by central bank
CurrAndDepAssetsDepTaking	Net US acquisition of other investment assets; currency and deposits; held by deposit-taking institutions except central bank
CurrAndDepAssetsOthFinNonFin	Net US acquisition of other investment assets; currency and deposits; held by non-deposit-taking institutions and nonfinancial institutions except general government
CurrAndDepLiabs	Net US incurrence of other investment liabilities; currency and deposits
CurrAndDepLiabsCentralBank	Net US incurrence of other investment liabilities; currency and deposits; issued by central bank
CurrAndDepLiabsDepTaking	Net US incurrence of other investment liabilities; currency and deposits; issued by deposit-taking institutions except central bank
CurrAndDepLiabsFoa	Net US incurrence of other investment liabilities to foreign official agencies; currency and deposits
CurrAndDepLiabsOthFinNonFin	Net US incurrence of other investment liabilities; currency and deposits; issued by non-deposit-taking institutions and nonfinancial institutions except general government
CurrAndDepReserveAssets	Net US acquisition of reserve assets; other; currency and deposits
CurrAssets	Net US acquisition of other investment assets; other; currency and deposits

Table 22.24 *continued*

Series Indicator	Description
CurrLiabs	Net US acquisition of other investment liabilities; currency (short term)
CurrLiabsCentralBank	Net US acquisition of other investment liabilities; currency (short term); issued by central bank

Table 22.25 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
DebtSecAssets	Net US acquisition of portfolio investment assets; debt securities
DebtSecAssetsDepTaking	Net US acquisition of portfolio investment assets; debt securities; held by deposit-taking institutions except central bank
DebtSecAssetsNonFin	Net US acquisition of portfolio investment assets; debt securities; held by nonfinancial institutions except general government
DebtSecAssetsOthFin	Net US acquisition of portfolio investment assets; debt securities; held by non-deposit-taking financial institutions
DebtSecFedSponsorAgencyIncPay	Portfolio investment income payments; interest on federally sponsored agency securities
DebtSecFedSponsorAgencyLiabs	Net US incurrence of portfolio investment liabilities; federally sponsored agency securities
DebtSecIncPay	Portfolio investment income payments; interest on debt securities
DebtSecIncPayDepTaking	Portfolio investment income payments; interest on debt securities; deposit-taking institutions except central bank
DebtSecIncPayGenGovt	Portfolio investment income payments; interest on debt securities; general government
DebtSecIncPayNonFin	Portfolio investment income payments; interest on debt securities; nonfinancial institutions except general government
DebtSecIncPayOthFin	Portfolio investment income payments; interest on debt securities; non-deposit-taking financial institutions
DebtSecIncRec	Portfolio investment income receipts; interest on debt securities
DebtSecIncRecDepTaking	Portfolio investment income receipts; interest on debt securities; deposit-taking institutions except central bank
DebtSecIncRecNonFin	Portfolio investment income receipts; interest on debt securities; nonfinancial institutions except general government
DebtSecIncRecOthFin	Portfolio investment income receipts; interest on debt securities; non-deposit-taking financial institutions
DebtSecLiabs	Net US incurrence of portfolio investment liabilities; debt securities
DebtSecLiabsDepTaking	Net US incurrence of portfolio investment liabilities; debt securities; issued by deposit-taking institutions except central bank

Table 22.25 *continued*

Series Indicator	Description
DebtSecLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; debt securities
DebtSecLiabsGenGovt	Net US incurrence of portfolio investment liabilities; debt securities; issued by general government

Table 22.26 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
DebtSecLiabsNonFin	Net US incurrence of portfolio investment liabilities; debt securities; issued by nonfinancial institutions except general government
DebtSecLiabsOthFin	Net US incurrence of portfolio investment liabilities; debt securities; issued by non-deposit-taking financial institutions
DebtSecOthThanFedSponsorAgencyIncPayOthFin	Portfolio investment income payments; interest on debt securities other than federally sponsored agency securities; non-deposit-taking financial institutions
DebtSecOthThanFedSponsorAgencyLiabsOthFin	Net US incurrence of portfolio investment liabilities; debt securities other than federally sponsored agency securities; issued by non-deposit-taking financial institutions
DebtSecTreasIncPay	Portfolio investment income payments; interest on US Treasury securities
DebtSecTreasLiabs	Net US incurrence of portfolio investment liabilities; US Treasury securities
DepAssets	Net US acquisition of other investment assets; deposits
DepAssetsCentralBank	Net US acquisition of other investment assets; deposits; held by central bank
DepAssetsDepTaking	Net US acquisition of other investment assets; deposits; held by deposit-taking institutions except central bank
DepAssetsOthFinNonFin	Net US acquisition of other investment assets; deposits; held by non-deposit-taking institutions and nonfinancial institutions except general government
DepLiabs	Net US incurrence of other investment liabilities; deposits

Table 22.26 *continued*

Series Indicator	Description
DepLiabsCentralBank	Net US incurrence of other investment liabilities; deposits; issued by central bank
DepLiabsDepTaking	Net US incurrence of other investment liabilities; deposits; issued by deposit-taking institutions except central bank
DepLiabsOthFinNonFin	Net US incurrence of other investment liabilities; deposits; issued by non-deposit-taking institutions and nonfinancial institutions except general government
DepRepurchaseLiabsDepTaking	Net US incurrence of other investment liabilities; deposits, of which repurchase agreements; issued by deposit-taking institutions except central bank
DepResaleAssetsDepTaking	Net US acquisition of other investment assets; deposits, of which resale agreements; held by deposit-taking institutions except central bank

Table 22.27 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
DiInvAssets	Net US acquisition of direct investment assets, asset/liability basis
DiInvCurrCostAdjAssets	Net US acquisition of direct investment assets; current-cost adjustment
DiInvCurrCostAdjIncPay	Direct investment income on liabilities; current-cost adjustment
DiInvCurrCostAdjIncRec	Direct investment income on assets; current-cost adjustment
DiInvCurrCostAdjLiabs	Net US incurrence of direct investment liabilities; current-cost adjustment
DiInvDebtInstAssets	Net US acquisition of direct investment assets, asset/liability basis; debt instruments
DiInvDebtInstInward	Financial transactions for inward direct investment (foreign direct investment in the United States), directional basis; debt instruments
DiInvDebtInstInwardByInd	Financial transactions for inward direct investment (foreign direct investment in the United States), directional basis; debt instruments; by industry
DiInvDebtInstInwardFinAndIns	Financial transactions for inward direct investment (foreign direct investment in the United States), directional basis; debt instruments; finance (including depository institutions) and insurance
DiInvDebtInstInwardMnfctr	Financial transactions for inward direct investment (foreign direct investment in the United States), directional basis; debt instruments; manufacturing)

Table 22.27 *continued*

Series Indicator	Description
DiInvDebtInstInwardOthInd	Financial transactions for inward direct investment (foreign direct investment in the United States), directional basis; debt instruments; other industries (those not listed under incurrence of liabilities in table 6.1))
DiInvDebtInstInwardWhlsTrd	Financial transactions for inward direct investment (foreign direct investment in the United States), directional basis; debt instruments; wholesale trade)
DiInvDebtInstLiabs	Net US incurrence of direct investment liabilities, asset/liability basis; debt instruments
DiInvDebtInstOutward	Financial transactions for outward direct investment (US direct investment abroad), directional basis; debt instruments
DiInvDebtInstOutwardByInd	Financial transactions for outward direct investment (US direct investment abroad), direct investment abroad), directional basis; debt instruments; by industry
DiInvDebtInstOutwardFinAndIns	Financial transactions for outward direct investment (US direct investment abroad), directional basis; debt instruments; finance (including depository institutions) and insurance
DiInvDebtInstOutwardHoldExcBank	Financial transactions for outward direct investment (US direct investment abroad), directional basis; debt instruments; holding companies except bank holding companies

Table 22.28 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
DiInvDebtInstOutwardMnfctr	Financial transactions for outward direct investment (US direct investment abroad), directional basis; debt instruments; manufacturing
DiInvDebtInstOutwardOthInd	Financial transactions for outward direct investment (US direct investment abroad), directional basis; debt instruments; other industries (those not listed under acquisition of assets in table 6.1)
DiInvDebtInstOutwardWhlsITrd	Financial transactions for outward direct investment (US direct investment abroad), directional basis; debt instruments; wholesale trade
DiInvDebtInstUSAffiliatesClaims	Financial transactions for direct investment; debt instruments; US affiliates' claims
DiInvDebtInstUSAffiliatesLiabs	Financial transactions for direct investment; debt instruments; US affiliates' liabilities
DiInvDebtInstUSParentsClaims	Financial transactions for direct investment; debt instruments; US parents' claims
DiInvDebtInstUSParentsLiabs	Financial transactions for direct investment; debt instruments; US parents' liabilities
DiInvDirectionalBasisAdj	Financial transactions for direct investment; adjustments to convert to directional basis
DiInvDirectionalBasisAdjIncPay	Direct investment income; adjustments to convert to directional basis
DiInvDirectionalBasisAdjIncRec	Direct investment income; adjustments to convert to directional basis
DiInvDivWithdrawIncPay	Direct investment income on liabilities; dividends and withdrawals
DiInvDivWithdrawIncRec	Direct investment income on assets; dividends and withdrawals
DiInvEquityAssets	Net US acquisition of direct investment assets; equity
DiInvEquityIncPay	Direct investment income on liabilities; equity
DiInvEquityIncRec	Direct investment income on assets; equity
DiInvEquityIntIncRecFinAndIns	Direct investment income on outward investment (US direct investment abroad), directional basis; interest, net receipts; finance (including depository institutions) and insurance
DiInvEquityIntIncRecHoldExcBank	Direct investment income on outward investment (US direct investment abroad), directional basis; interest, net receipts; holding companies except bank holding companies

Table 22.28 *continued*

Series Indicator	Description
DiInvEquityIntIncRecMnfctr	Direct investment income on outward investment (US direct investment abroad), directional basis; interest, net receipts; manufacturing
DiInvEquityIntIncRecOthInd	Direct investment income on outward investment (US direct investment abroad), directional basis; interest, net receipts; manufacturing; other industries (those not listed under receipts in table 4.2)
DiInvEquityIntIncRecWhlslTrd	Direct investment income on outward investment (US direct investment abroad), directional basis; interest, net receipts; wholesale trade
DiInvEquityLiabs	Net US incurrence of direct investment liabilities; equity
DiInvEquityOthThanReinvestEarnAssets	Net US acquisition of direct investment assets; equity other than reinvestment of earnings
DiInvEquityOthThanReinvestEarnAssetsFinAndIns	Net US acquisition of direct investment assets; equity other than reinvestment of earnings; finance (including depository institutions) and insurance
DiInvEquityOthThanReinvestEarnAssetsHoldExcBank	Net US acquisition of direct investment assets; equity other than reinvestment of earnings; holding companies except bank holding companies
DiInvEquityOthThanReinvestEarnAssetsMnfctr	Net US acquisition of direct investment assets; equity other than reinvestment of earnings; manufacturing
DiInvEquityOthThanReinvestEarnAssetsOthInd	Net US acquisition of direct investment assets; equity other than reinvestment of earnings; other industries (those not listed under acquisition of assets in table 6.1)
DiInvEquityOthThanReinvestEarnAssetsWhlslTrd	Net US acquisition of direct investment assets; equity other than reinvestment of earnings; wholesale trade

Table 22.29 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
DiInvEquityOthThanReinvestEarnDecAssets	Net US acquisition of direct investment assets; equity other than reinvestment of earnings; decreases
DiInvEquityOthThanReinvestEarnDecLiabs	Net US incurrence of direct investment liabilities; equity other than reinvestment of earnings; decreases
DiInvEquityOthThanReinvestEarnIncAssets	Net US acquisition of direct investment assets; equity other than reinvestment of earnings; increases
DiInvEquityOthThanReinvestEarnIncLiabs	Net US incurrence of direct investment liabilities; equity other than reinvestment of earnings; increases
DiInvEquityOthThanReinvestEarnLiabs	Net US incurrence of direct investment liabilities; equity other than reinvestment of earnings
DiInvEquityOthThanReinvestEarnLiabsFinAndIns	Net US incurrence of direct investment liabilities; equity other than reinvestment of earnings; finance (including depository institutions) and insurance
DiInvEquityOthThanReinvestEarnLiabsHoldExcBank	Net US incurrence of direct investment liabilities; equity other than reinvestment of earnings; other industries (those not listed under incurrence of liabilities in table 6.1)
DiInvEquityOthThanReinvestEarnLiabsMnfctr	Net US incurrence of direct investment liabilities; equity other than reinvestment of earnings; manufacturing
DiInvEquityOthThanReinvestEarnLiabsWhlsITrd	Net US incurrence of direct investment liabilities; equity other than reinvestment of earnings; wholesale trade
DiInvEquityWithoutCurrCostAdjIncPay	Direct investment income without current-cost adjustment on liabilities; equity
DiInvEquityWithoutCurrCostAdjIncPayFinAndIns	Direct investment income without current-cost adjustment on liabilities; equity; finance (including depository institutions) and insurance
DiInvEquityWithoutCurrCostAdjIncPayMnfctr	Direct investment income without current-cost adjustment on liabilities; equity; manufacturing
DiInvEquityWithoutCurrCostAdjIncPayOthInd	Direct investment income without current-cost adjustment on liabilities; equity; other industries (those not listed under receipts in table 4.2)
DiInvEquityWithoutCurrCostAdjIncPayWhlsITrd	Direct investment income without current-cost adjustment on liabilities; equity; wholesale trade
DiInvEquityWithoutCurrCostAdjIncRec	Direct investment income without current-cost adjustment on assets; equity
DiInvEquityWithoutCurrCostAdjIncRecFinAndIns	Direct investment income without current-cost adjustment on assets; equity; finance (including depository institutions) and insurance

Table 22.29 *continued*

Series Indicator	Description
DiInvEquityWithoutCurrCostAdjIncRecHoldExcBank	Direct investment income without current-cost adjustment on assets; equity; holding companies except bank holding companies
DiInvEquityWithoutCurrCostAdjIncRecMnfctr	Direct investment income without current-cost adjustment on assets; equity; manufacturing
DiInvEquityWithoutCurrCostAdjIncRecOthInd	Direct investment income without current-cost adjustment on assets; equity; manufacturing; other industries (those not listed under receipts in table 4.2)
DiInvEquityWithoutCurrCostAdjIncRecWhlsITrd	Direct investment income without current-cost adjustment on assets; equity; wholesale trade
DiInvIncPay	Direct investment income on liabilities, asset/liability basis
DiInvIncRec	Direct investment income on assets, asset/liability basis
DiInvIntIncInward	Direct investment income on inward investment (foreign direct investment in the United States), directional basis; interest, net payments
DiInvIntIncInwardFinAndIns	Direct investment income on inward investment (foreign direct investment in the United States), directional basis; interest, net payments; finance (including depository institutions) and insurance
DiInvIntIncInwardMnfctr	Direct investment income on inward investment (foreign direct investment in the United States), directional basis; interest, net payments; manufacturing
DiInvIntIncInwardOthInd	Direct investment income on inward investment (foreign direct investment in the United States), directional basis; interest, net payments; other industries (those not listed under receipts in table 4.2)
DiInvIntIncInwardWhlsITrd	Direct investment income on inward investment (foreign direct investment in the United States), directional basis; interest, net payments; wholesale trade
DiInvIntIncOutward	Direct investment income on outward investment (US direct investment abroad), directional basis; interest, net receipts

Table 22.30 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
DiInvIntIncPay	Direct investment income on liabilities, asset/liability basis; interest payments
DiInvIntIncRec	Direct investment income on assets, asset/liability basis; interest receipts
DiInvIntUSAffiliatesIncPay	Direct investment income; US affiliates' interest payments
DiInvIntUSAffiliatesIncRec	Direct investment income; US affiliates' interest receipts
DiInvIntUSParentsIncPay	Direct investment income; US parents' interest payments
DiInvIntUSParentsIncRec	Direct investment income; US parents' interest receipts
DiInvInwardDirectionalBasis	Financial transactions for inward direct investment (foreign direct investment in the United States), directional basis
DiInvLiabs	Net US incurrence of direct investment liabilities, asset/liability basis
DiInvOutward	Financial transactions for outward direct investment (US direct investment abroad), directional basis
DiInvReinvestEarnAssets	Net US acquisition of direct investment assets; reinvestment of earnings
DiInvReinvestEarnIncPay	Direct investment income on liabilities; reinvested earnings
DiInvReinvestEarnIncRec	Direct investment income on assets; reinvested earnings
DiInvReinvestEarnLiabs	Net US incurrence of direct investment liabilities; reinvestment of earnings
DiInvReinvestEarnWithoutCurrCostAdjAssets	Net US acquisition of direct investment assets; reinvestment of earnings without current-cost adjustment
DiInvReinvestEarnWithoutCurrCostAdjAssetsFinAndIns	Net US acquisition of direct investment assets; reinvestment of earnings without current-cost adjustment; finance (including depository institutions) and insurance
DiInvReinvestEarnWithoutCurrCostAdjAssetsHoldExcBank	Net US acquisition of direct investment assets; reinvestment of earnings without current-cost adjustment; holding companies except bank holding companies
DiInvReinvestEarnWithoutCurrCostAdjAssetsMnfctr	Net US acquisition of direct investment assets; reinvestment of earnings without current-cost adjustment; manufacturing
DiInvReinvestEarnWithoutCurrCostAdjAssetsOthInd	Net US acquisition of direct investment assets;

Table 22.30 *continued*

Series Indicator	Description
	reinvestment of earnings without current-cost adjustment; other industries (those not listed under acquisition of assets in table 6.1)
DiInvReinvestEarnWithoutCurrCostAdjAssetsWhlslTrd	Net US acquisition of direct investment assets; reinvestment of earnings without current-cost adjustment; wholesale trade
DiInvReinvestEarnWithoutCurrCostAdjIncPay	Direct investment income on liabilities; reinvested earnings without current-cost adjustment
DiInvReinvestEarnWithoutCurrCostAdjIncRec	Direct investment income on assets; reinvested earnings without current-cost adjustment
DiInvReinvestEarnWithoutCurrCostAdjLiabs	Net US incurrence of direct investment liabilities; reinvestment of earnings without current-cost adjustment

Table 22.31 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
DiInvReinvestEarnWithoutCurrCostAdjLiabsFinAndIns	Net US incurrence of direct investment liabilities; reinvestment of earnings without current-cost adjustment; finance (including depository institutions) and insurance
DiInvReinvestEarnWithoutCurrCostAdjLiabsMnfctr	Net US incurrence of direct investment liabilities; reinvestment of earnings without current-cost adjustment; manufacturing
DiInvReinvestEarnWithoutCurrCostAdjLiabsOthInd	Net US incurrence of direct investment liabilities; reinvestment of earnings without current-cost adjustment; other industries (those not listed under incurrence of liabilities in table 6.1)
DiInvReinvestEarnWithoutCurrCostAdjLiabsWhlslTrd	Net US incurrence of direct investment liabilities; reinvestment of earnings without current-cost adjustment; wholesale trade
DiInvWithoutCurrCostAdjIncInward	Direct investment income without current-cost adjustment on inward investment, directional basis
DiInvWithoutCurrCostAdjIncInwardFinAndIns	Direct investment income without current-cost adjustment on inward investment, directional basis; finance (including depository institutions) and insurance
DiInvWithoutCurrCostAdjIncInwardMnfctr	Direct investment income without current-cost adjustment on inward investment, directional basis; manufacturing

Table 22.31 *continued*

Series Indicator	Description
DiInvWithoutCurrCostAdjIncInwardOthInd	Direct investment income without current-cost adjustment on inward investment, directional basis; other industries (those not listed under receipts in table 4.2)
DiInvWithoutCurrCostAdjIncInwardWhlsITrd	Direct investment income without current-cost adjustment on inward investment, directional basis; wholesale trade
DiInvWithoutCurrCostAdjIncOutwardFinAndIns	Direct investment income without current-cost adjustment on outward investment (US direct investment abroad), directional basis; finance (including depository institutions) and insurance
DiInvWithoutCurrCostAdjIncOutwardHoldExcBank	Direct investment income without current-cost adjustment on outward investment (US direct investment abroad), directional basis; holding companies except bank holding companies
DiInvWithoutCurrCostAdjIncOutwardMnfctr	Direct investment income without current-cost adjustment on outward investment (US direct investment abroad), directional basis; manufacturing
DiInvWithoutCurrCostAdjIncOutwardOthInd	Direct investment income without current-cost adjustment on outward investment (US direct investment abroad), directional basis; manufacturing; other industries (those not listed under receipts in table 4.2)
DiInvWithoutCurrCostAdjIncOutwardWhlsITrd	Direct investment income without current-cost adjustment on outward investment (US direct investment abroad), directional basis; wholesale trade

Table 22.32 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
DiInvWithoutCurrCostAdjInward	Financial transactions without current-cost adjustment for inward direct investment (foreign direct investment in the United States), directional basis
DiInvWithoutCurrCostAdjInwardFinAndIns	Financial transactions without current-cost adjustment for inward direct investment (foreign direct investment in the United States), directional basis; finance (including depository institutions) and insurance
DiInvWithoutCurrCostAdjInwardMnfctr	Financial transactions without current-cost adjustment for inward direct investment (foreign direct investment in the United States), directional basis; manufacturing
DiInvWithoutCurrCostAdjInwardOthInd	Financial transactions without current-cost adjustment for inward direct investment (foreign direct investment in the United States), directional basis; other industries (those not listed under incurrence of liabilities in table 6.1)
DiInvWithoutCurrCostAdjInwardWhlsTrd	Financial transactions without current-cost adjustment for inward direct investment (foreign direct investment in the United States), directional basis; wholesale trade
DiInvWithoutCurrCostAdjOutward	Financial transactions without current-cost adjustment for outward direct investment (US direct investment abroad), directional basis
DiInvWithoutCurrCostAdjOutwardFinAndIns	Financial transactions without current-cost adjustment for outward direct investment (US direct investment abroad), directional basis; finance (including depository institutions) and insurance
DiInvWithoutCurrCostAdjOutwardHoldExcBank	Financial transactions without current-cost adjustment for outward direct investment (US direct investment abroad), directional basis; holding companies except bank holding companies
DiInvWithoutCurrCostAdjOutwardMnfctr	Financial transactions without current-cost adjustment for outward direct investment (US direct investment abroad), directional basis; manufacturing

Table 22.32 *continued*

Series Indicator	Description
DiInvWithoutCurrCostAdjOutwardOthInd	Financial transactions without current-cost adjustment for outward direct investment (US direct investment abroad), directional basis; other industries (those not listed under acquisition of assets in table 6.1)
DiInvWithoutCurrCostAdjOutwardWhlslTrd	Financial transactions without current-cost adjustment for outward direct investment (US direct investment abroad), directional basis; other industries (those not listed under acquisition of assets in table 6.1)

Table 22.33 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
EquityAndInvFundSharesAssets	Net US acquisition of portfolio investment assets; equity and investment fund shares
EquityAndInvFundSharesAssetsDepTaking	Net US acquisition of portfolio investment assets; equity and investment fund shares; held by deposit-taking institutions except central bank
EquityAndInvFundSharesAssetsNonFin	Net US acquisition of portfolio investment assets; equity and investment fund shares; held by nonfinancial institutions except general government
EquityAndInvFundSharesAssetsOthFin	Net US acquisition of portfolio investment assets; equity and investment fund shares; held by non-deposit-taking financial institutions
EquityAndInvFundSharesIncPay	Portfolio investment income payments; income on equity and investment fund shares
EquityAndInvFundSharesIncPayDepTaking	Portfolio investment income payments on equity and investment fund shares; deposit-taking institutions except central bank
EquityAndInvFundSharesIncPayNonFin	Portfolio investment income payments on equity and investment fund shares; nonfinancial institutions except general government
EquityAndInvFundSharesIncPayOthFin	Portfolio investment income payments on equity and investment fund shares; non-deposit-taking financial institutions
EquityAndInvFundSharesIncRec	Portfolio investment income receipts; income on equity and investment fund shares
EquityAndInvFundSharesIncRecDepTaking	Portfolio investment income receipts on equity and investment fund shares; deposit-taking institutions except central bank

Table 22.33 *continued*

Series Indicator	Description
EquityAndInvFundSharesIncRecNonFin	Portfolio investment income receipts on equity and investment fund shares; nonfinancial institutions except general government
EquityAndInvFundSharesIncRecOthFin	Portfolio investment income receipts on equity and investment fund shares; non-deposit-taking financial institutions
EquityAndInvFundSharesLiabs	Net US incurrence of portfolio investment liabilities; equity and investment fund share
EquityAndInvFundSharesLiabsDepTaking	Net US incurrence of portfolio investment liabilities; equity and investment fund share; issued by deposit-taking institutions except central bank
EquityAndInvFundSharesLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; equity and investment fund shares
EquityAndInvFundSharesLiabsNonFin	Net US incurrence of portfolio investment liabilities; equity and investment fund shares; issued by nonfinancial institutions except general government
EquityAndInvFundSharesLiabsOthFin	Net US incurrence of portfolio investment liabilities; equity and investment fund shares; issued by non-deposit-taking financial institutions
EquityOthThanInvFundSharesAssets	Net US acquisition of portfolio investment assets; equity other than investment fund shares
EquityOthThanInvFundSharesIncPay	Portfolio investment income payments; dividends on equity other than investment fund shares
EquityOthThanInvFundSharesIncRec	Portfolio investment income receipts; dividends on equity other than investment fund shares
EquityOthThanInvFundSharesLiabs	Net US incurrence of portfolio investment liabilities; equity other than investment fund shares
EquityOthThanInvFundSharesLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; equity excluding investment fund shares

Table 22.34 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
ExpGds	Exports of goods
ExpGdsAgFoodsFeedsAndBevs	Exports of agricultural foods, feeds, and beverages
ExpGdsAgIsM	Exports of agricultural industrial supplies and materials
ExpGdsAppFootAndHouse	Exports of apparel, footwear, and household goods
ExpGdsAutoEngAndEngParts	Exports of automotive engines and engine parts
ExpGdsAutoVehPartsAndEngines	Exports of automotive vehicles, parts, and engines
ExpGdsBauxAndAlum	Exports of bauxite and aluminum
ExpGdsBopAdj	Exports of goods; balance of payments adjustments, net
ExpGdsBuildMatsExcMetals	Exports of building materials except metals
ExpGdsCapGoodsExclAuto	Exports of capital goods except automotive
ExpGdsCensus	Exports of goods, Census basis
ExpGdsChemsExcMeds	Exports of chemicals except medicinals
ExpGdsCivAir	Exports of civilian aircraft, complete, all types
ExpGdsCivAirEngAndParts	Exports of civilian aircraft, engines, and parts
ExpGdsCoalAndRelProds	Exports of coal and related products
ExpGdsComp	Exports of computers
ExpGdsCompAccPeriAndParts	Exports of computer accessories, peripherals, and parts
ExpGdsConsGoodsExcFoodAndAuto	Exports of consumer goods except food and automotive
ExpGdsCopper	Exports of copper
ExpGdsCorn	Exports of corn
ExpGdsCrudePet	Exports of crude petroleum
ExpGdsDistBevAndOthNonAgFoodsFeedsAndBevs	Exports of distilled beverages and other nonagricultural foods, feeds, and beverages
ExpGdsDurCons	Exports of durable consumer goods
ExpGdsElecGenMachElecAppAndParts	Exports of electric-generating machinery, electric apparatus, and parts
ExpGdsEnergyProd	Exports of energy products
ExpGdsEngAndPartsForCivAir	Exports of engines and parts for civilian aircraft
ExpGdsFertPestAndInsect	Exports of fertilizers, pesticides, and insecticides
ExpGdsFishShellfish	Exports of fish and shellfish
ExpGdsFoodsFeedsAndBevs	Exports of foods, feeds, and beverages
ExpGdsFuelOil	Exports of fuel oil
ExpGdsGdsProcPortsBopAdj	Exports of goods; balance of payments adjustments, net; goods procured in US ports by foreign carriers
ExpGdsGemDiamAndOthGem	Exports of gem diamonds and other gemstones
ExpGdsGenMerch	Exports of general merchandise
ExpGdsGrainsPreps	Exports of grains and preparations
ExpGdsHidesSkins	Exports of hides and skins, including fur skins

Table 22.34 *continued*

Series Indicator	Description
ExpGdsHouseAndKitchApp	Exports of household and kitchen appliances
ExpGdsHouseFurnAndRelProds	Exports of household furnishings and related products
ExpGdsHouseKitchAppAndOthHouse	Exports of household and kitchen appliances and other household goods
ExpGdsIndEngPumpsComps	Exports of industrial engines, pump, and compressors
ExpGdsIndInorgChems	Exports of industrial inorganic chemicals
ExpGdsIndOrgChems	Exports of industrial organic chemicals
ExpGdsIronAndSteelProds	Exports of iron and steel products
ExpGdsIsM	Exports of industrial supplies and materials
ExpGdsJewelryAndCollect	Exports of jewelry and collectibles
ExpGdsLiqPetGases	Exports of liquified petroleum gases
ExpGdsMachAndEquipExcCons	Exports of machinery and equipment except consumer-type
ExpGdsMachToolsMetalworkMach	Exports of machine tools and metalworking machinery
ExpGdsMeasTestControlInst	Exports of measuring, testing, and control instruments
ExpGdsMeatProdsPoultry	Exports of meat products and poultry
ExpGdsMedDentAndPharm	Exports of medicinal, dental, and pharmaceutical products
ExpGdsMerchantingBopAdj	Exports of goods; balance of payments adjustments, net; net exports of goods under merchanting
ExpGdsMerchantingNet	Net exports of goods under merchanting
ExpGdsMetalsAndNonmetProds	Exports of metals and nonmetallic products
ExpGdsNaturalGas	Exports of natural gas
ExpGdsNonAgFoodsFeedsAndBevs	Exports of nonagricultural foods, feeds, and beverages
ExpGdsNonAgIsM	Exports of nonagricultural industrial supplies and materials
ExpGdsNondurCons	Exports of nondurable consumer goods
ExpGdsNonferrousMetals	Exports of nonferrous metals
ExpGdsNonmonetaryGold	Exports of nonmonetary gold
ExpGdsNonmonGoldBopAdj	Exports of goods; balance of payments adjustments, net; nonmonetary gold

Table 22.35 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
ExpGdsNuclearFuelAndElecEnergy	Exports of nuclear fuel and electric energy
ExpGdsOilDrillMiningConstMach	Exports of oil-drilling, mining, and construction machinery
ExpGdsOthAgFoodsFeedsAndBevs	Exports of other agricultural foods, feeds, and beverages
ExpGdsOthAgIsm	Exports of other agricultural industrial supplies
ExpGdsOthAutoPartsAndAcc	Exports of other automotive parts and accessories
ExpGdsOthBopAdj	Exports of goods; balance of payments adjustments, net; other adjustments, net
ExpGdsOthChems	Exports of other chemicals
ExpGdsOthDurCons	Exports of other durable consumer goods
ExpGdsOthFeeds	Exports of other feeds
ExpGdsOthGenMerch	Exports of other general merchandise
ExpGdsOthHouseIncCellPhones	Exports of other household goods, including cell phones
ExpGdsOthIndMach	Exports of other industrial machinery
ExpGdsOthMetalsAndNonmetProds	Exports of other metals and nonmetallic products
ExpGdsOthNondurCons	Exports of other nondurable consumer goods
ExpGdsOthNonferrousMetals	Exports of other nonferrous metals
ExpGdsOthNonmetals	Exports of other nonmetals
ExpGdsOthOfficeAndBusMach	Exports of other office and business machines
ExpGdsOthPetProds	Exports of other petroleum products
ExpGdsOthServIndAndAgMach	Exports of other service-industry and agricultural machinery
ExpGdsOthTransEquip	Exports of other transportation equipment
ExpGdsPaperAndPaperBaseStocks	Exports of paper and paper-base stocks
ExpGdsPassCars	Exports of passenger cars, new and used
ExpGdsPetAndProds	Exports of petroleum and products
ExpGdsPlasticMaterials	Exports of plastic materials
ExpGdsPrecMetalsExcNonmonGold	Exports of precious metals except nonmonetary gold
ExpGdsPrivGiftParcelRemitBopAdj	Exports of goods; balance of payments adjustments, net; private gift parcel remittances
ExpGdsRadioAndStereoEquip	Exports of radio and stereo equipment, including recorded media
ExpGdsRawCotton	Exports of raw cotton
ExpGdsRepairEquipBopAdj	Exports of goods; balance of payments adjustments, net; repair of equipment
ExpGdsRiceOthFoodGrains	Exports of rice and other food grains
ExpGdsSciHospAndMedEquipAndParts	Exports of scientific, hospital, and medical equipment and parts
ExpGdsSemiconductors	Exports of semiconductors
ExpGdsServ	Exports of goods and services
ExpGdsServIncRec	Exports of goods and services and income receipts

Table 22.35 *continued*

Series Indicator	Description
	(credits)
ExpGdsSoybeans	Exports of soybeans
ExpGdsSteelmakingMats	Exports of steelmaking materials
ExpGdsTelecomEquip	Exports of telecommunications equipment
ExpGdsTextileSupAndRelMats	Exports of textile supplies and related materials
ExpGdsToilAndCosmet	Exports of toiletries and cosmetics
ExpGdsToysAndSport	Exports of toys and sporting goods, including bicycles
ExpGdsTrucksBusesSpecPurpVeh	Exports of trucks, buses, and special-purpose vehicles
ExpGdsTvsVidRecAndOthVidEquip	Exports of televisions, video receivers, and other video equipment
ExpGdsUnmanufTobacco	Exports of unmanufactured tobacco
ExpGdsUSMilAgencyBopAdj	Exports of goods; balance of payments adjustments, net; exports under US military agency sales contracts
ExpGdsVegFruitNutPreps	Exports of vegetables, fruits, nuts, and preparations
ExpGdsWheat	Exports of wheat
ExpGenMerch	Exports of general merchandise
ExpServ	Exports of services
ExpServArtisticRelated	Exports of artistic related services
ExpServAudVis	Exports of audiovisual services

Table 22.36 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
ExpServChargesForTheUseOfIpNie	Charges for the use of intellectual property, not included elsewhere (n.i.e.); exports
ExpServCipAudVisRelated	Charges for the use of intellectual property n.i.e.; exports; audiovisual and related products
ExpServCipCompSoft	Charges for the use of intellectual property n.i.e.; exports; computer software
ExpServCipIndProcess	Charges for the use of intellectual property n.i.e.; exports; industrial processes
ExpServCipLicensesAudVis	Charges for the use of intellectual property n.i.e.; exports; licenses to reproduce and/or distribute audiovisual products
ExpServCipLicensesCompSoftware	Charges for the use of intellectual property n.i.e.; exports; licenses to reproduce and/or distribute computer software
ExpServCipLicensesFranchisesTrademarks	Charges for the use of intellectual property n.i.e.; exports; franchises and trademarks licensing fees

Table 22.36 *continued*

Series Indicator	Description
ExpServCipLicensesOutcomesResearchAndDev	Charges for the use of intellectual property n.i.e.; exports; licenses for the use of outcomes of research and development
ExpServCipOth	Charges for the use of intellectual property n.i.e.; exports; other intellectual property
ExpServCipTrademarkFranchiseFees	Charges for the use of intellectual property n.i.e.; exports; trademarks and franchise fees
ExpServComp	Exports of computer services
ExpServFinancial	Exports of financial services
ExpServFinCredCardOthCredRelated	Exports of credit card and other credit-related services
ExpServFinFinManFinAdvCust	Exports of financial management, financial advisory, and custody services
ExpServFinSecBrokUwRelated	Exports of securities brokerage, underwriting and related services
ExpServFinSecLendEftOth	Exports of securities lending, electronic funds transfer, and other services
ExpServGovtGoodsAndServicesNie	Exports of government goods and services n.i.e.
ExpServInfo	Exports of information services
ExpServInsurance	Exports of insurance services
ExpServInsuranceAuxIns	Exports of auxiliary insurance services
ExpServInsuranceDirect	Exports of direct insurance services
ExpServInsuranceReIns	Exports of reinsurance services
ExpServMaintenanceAndRepairNie	Exports of maintenance and repair services n.i.e.
ExpServOtherBusiness	Exports of other business services
ExpServProfMgmtConsult	Exports of professional and management consulting services
ExpServResearchAndDev	Exports of research and development services
ExpServTechTradeRelatedOth	Exports of technical, trade-related, and other business services
ExpServTelecom	Exports of telecommunications services
ExpServTelecomCompAndInfo	Exports of telecommunications, computer, and information services
ExpServTransport	Exports of transport services
ExpServTransportAir	Exports of air transport services
ExpServTransportAirFreight	Exports of air freight services
ExpServTransportAirPass	Exports of air passenger services
ExpServTransportAirPort	Exports of airport services
ExpServTransportOth	Exports of transport services; other modes of transport

Table 22.36 *continued*

Series Indicator	Description
ExpServTransportSea	Exports of sea transport services
ExpServTransportSeaFreight	Exports of sea freight services
ExpServTransportSeaPort	Exports of seaport services
ExpServTravel	Exports of travel services (for all purposes, including education)

Table 22.37 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
ExpServTravelBusiness	Exports of business travel services
ExpServTravelBusinessOth	Exports of other business travel services
ExpServTravelEducation	Exports of education-related services
ExpServTravelHealth	Exports of health-related services
ExpServTravelPersonal	Exports of personal travel services
ExpServTravelPersonalOth	Exports of other personal travel services
ExpServTravelShortTermWork	Expenditures in the US by border, seasonal, and other short-term workers
FinAssetsExclFinDeriv	Net US acquisition of financial assets, excluding financial derivatives
FinDeriv	Financial derivatives other than reserves, net transactions
FinDerivReserveAssets	Net US acquisition of reserve assets; other; financial derivatives
FinLiabsExclFinDeriv	Net US incurrence of liabilities, excluding financial derivatives
FinLiabsFoa	Net US incurrence of liabilities to foreign official agencies
GoldReserveAssets	Net US acquisition of reserve assets; monetary gold
ImfReserveAssets	Net US acquisition of reserve assets; reserve position in the International Monetary Fund
ImpGds	Imports of goods
ImpGdsAgFoodsFeedsAndBevs	Imports of agricultural foods, feeds, and beverages
ImpGdsAgIsm	Imports of agricultural industrial supplies and materials
ImpGdsAppFootAndHouse	Imports of apparel, footwear, and household goods
ImpGdsAutoEngAndEngParts	Imports of automotive engines and engine parts
ImpGdsAutoVehPartsAndEngines	Imports of automotive vehicles, parts, and engines
ImpGdsBauxAndAlum	Imports of bauxite and aluminum
ImpGdsBopAdj	Imports of goods; balance of payments adjustments, net
ImpGdsBuildMatsExcMetals	Imports of building materials except metals

Table 22.37 *continued*

Series Indicator	Description
ImpGdsCapGoodsExclAuto	Imports of capital goods except automotive
ImpGdsCensus	Imports of goods, Census basis
ImpGdsChemsExcMeds	Imports of chemicals except medicinals
ImpGdsCivAir	Imports of civilian aircraft, complete, all types
ImpGdsCivAirEngAndParts	Imports of civilian aircraft, complete, all types
ImpGdsCoalAndRelProds	Imports of coal and related products
ImpGdsCocoaAndSugar	Imports of cocoa beans and sugar
ImpGdsComp	Imports of computers
ImpGdsCompAccPeriAndParts	Imports of computer accessories, peripherals, and parts
ImpGdsConsGoodsExcFoodAndAuto	Imports of consumer goods except food and automotive
ImpGdsCrudePet	Imports of crude petroleum
ImpGdsDistBevAndOthNonAgFoodsFeedsAndBevs	Imports of distilled beverages and other nonagricultural foods, feeds, and beverages
ImpGdsDurCons	Imports of durable consumer goods
ImpGdsElecGenMachElecAppAndParts	Imports of electric-generating machinery, electric apparatus and parts
ImpGdsEnergyProds	Imports of energy products
ImpGdsEngAndPartsForCivAir	Imports of engines and parts for civilian aircraft
ImpGdsFertPestAndInsect	Imports of fertilizers, pesticides, and insecticides
ImpGdsFishShellfish	Imports of fish and shellfish
ImpGdsFoodsFeedsAndBevs	Imports of foods, feeds, and beverages
ImpGdsFuelOil	Imports of fuel oil
ImpGdsGdsProcPortsBopAdj	Imports of goods; balance of payments adjustments, net; goods procured in foreign ports
ImpGdsGemDiamAndOthGem	Imports of gem diamonds and other gemstones
ImpGdsGenMerch	Imports of general merchandise
ImpGdsGreenCoffee	Imports of green coffee
ImpGdsHouseAndKitchApp	Imports of household and kitchen appliances
ImpGdsHouseFurnAndRelProds	Imports of household furnishings and related product
ImpGdsHouseKitchAppAndOthHouse	Imports of household and kitchen appliances and other household goods
ImpGdsIndEngPumpsComps	Imports of industrial engines, pumps, and compressors
ImpGdsIndInorgChems	Imports of industrial inorganic chemicals
ImpGdsIndOrgChems	Imports of industrial organic chemicals
ImpGdsInlandFreightCanMexBopAdj	Imports of goods; balance of payments adjustments, net; inland freight in Canada and Mexico
ImpGdsIronAndSteelProds	Imports of iron and steel products
ImpGdsIsm	Imports of industrial supplies and materials
ImpGdsJewelryAndCollect	Imports of jewelry and collectibles

Table 22.38 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
ImpGdsLiqPetGases	Imports of liquified petroleum gases
ImpGdsLocoRailBopAdj	Imports of goods; balance of payments adjustments, net; locomotives and railcars
ImpGdsMachAndEquipExcCons	Imports of machinery and equipment except consumer-type
ImpGdsMachToolsMetalworkMach	Imports of machine tools and metalworking machinery
ImpGdsMeasTestControlInst	Imports of measuring, testing, and control instruments
ImpGdsMeatProdsPoultry	Imports of meat products and poultry
ImpGdsMedDentAndPharm	Imports of medicinal, dental, and pharmaceutical products
ImpGdsMetalsAndNonmetProds	Imports of metals and nonmetallic products
ImpGdsNaturalGas	Imports of natural gas
ImpGdsNonAgFoodsFeedsAndBevs	Imports of nonagricultural foods, feeds, and beverages
ImpGdsNonAgIsm	Imports of nonagricultural industrial supplies and materials
ImpGdsNondurCons	Imports of nondurable consumer goods
ImpGdsNonferrousMetals	Imports of nonferrous metals
ImpGdsNonmonetaryGold	Imports of nonmonetary gold
ImpGdsNonmonGoldBopAdj	Imports of goods; balance of payments adjustments, net; nonmonetary gold
ImpGdsNuclearFuelAndElecEnergy	Imports of nuclear fuel and electric energy
ImpGdsOilDrillMiningConstMach	Imports of oil-drilling, mining, and construction machinery
ImpGdsOthAgFoodsFeedsAndBevs	Imports of other agricultural foods, feeds, and beverages
ImpGdsOthAutoPartsAndAcc	Imports of other automotive parts and accessories
ImpGdsOthBopAdj	Imports of goods; balance of payments adjustments, net; other adjustments, net
ImpGdsOthChems	Imports of other chemicals
ImpGdsOthDurCons	Imports of other durable consumer goods
ImpGdsOthGenMerch	Imports of other general merchandise
ImpGdsOthHouseIncCellPhones	Imports of other household goods, including cell phones
ImpGdsOthIndMach	Imports of other industrial machinery
ImpGdsOthMetalsAndNonmetProds	Imports of other metals and nonmetallic products
ImpGdsOthNondurCons	Imports of other nondurable consumer goods
ImpGdsOthNonferrousMetals	Imports of other nonferrous metals
ImpGdsOthNonmetals	Imports of other nonmetals
ImpGdsOthOfficeAndBusMach	Imports of other office and business machines
ImpGdsOthPetProds	Imports of other petroleum products
ImpGdsOthServIndAndAgMach	Imports of other service-industry and agricultural

Table 22.38 *continued*

Series Indicator	Description
ImpGdsOthTransEquip	machinery Imports of other transportation equipment
ImpGdsPaperAndPaperBaseStocks	Imports of paper and paper-base stocks
ImpGdsPassCars	Imports of passenger cars, new and used
ImpGdsPetAndProds	Imports of petroleum and products
ImpGdsPlasticMaterials	Imports of plastic materials
ImpGdsPrecMetalsExcNonmonGold	Imports of precious metals except nonmonetary gold
ImpGdsRadioAndStereoEquip	Imports of radio and stereo equipment, including recorded media
ImpGdsRepairEquipBopAdj	Imports of goods; balance of payments adjustments, net; repair of equipment
ImpGdsSciHospAndMedEquipAndParts	Imports of scientific, hospital, and medical equipment and parts
ImpGdsSemiconductors	Imports of semiconductors
ImpGdsServ	Imports of goods and services
ImpGdsServIncPay	Imports of goods and services and income payments (debits)
ImpGdsSoftRevalBopAdj	Imports of goods; balance of payments adjustments, net; software revaluation
ImpGdsSteelmakingMats	Imports of steelmaking materials
ImpGdsTelecomEquip	Imports of telecommunications equipment
ImpGdsTextileSupAndRelMats	Imports of textile supplies and related materials
ImpGdsToilAndCosmet	Imports of toiletries and cosmetics
ImpGdsToysAndSport	Imports of toys and sporting goods, including bicycles
ImpGdsTrucksBusesSpecPurpVeh	Imports of trucks, buses, and special purpose vehicles
ImpGdsTvsVidRecAndOthVidEquip	Imports of televisions, video receivers, and other video equipment
ImpGdsUSMilAgencyBopAdj	Imports of goods; balance of payments adjustments, net; imports by US military agencies
ImpGdsVegFruitNutPreps	Imports of vegetables, fruits, nuts, and preparations
ImpGdsWineBeerRelProds	Imports of wine, beer, and related products
ImpServ	Imports of services
ImpServArtisticRelated	Imports of artistic related services
ImpServAudVis	Imports of audiovisual services

Table 22.39 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
ImpServChargesForTheUseOfIpNie	Charges for the use of intellectual property n.i.e.; imports
ImpServCipAudVisRelated	Charges for the use of intellectual property n.i.e.; imports; audiovisual and related property
ImpServCipCompSoft	Charges for the use of intellectual property n.i.e.; imports; computer software
ImpServCipIndProcess	Charges for the use of intellectual property n.i.e.; imports; industrial processes
ImpServCipLicensesAudVis	Charges for the use of intellectual property n.i.e.; imports; licenses to reproduce and/or distribute audiovisual products
ImpServCipLicensesCompSoftware	Charges for the use of intellectual property n.i.e.; imports; licenses to reproduce and/or distribute computer software
ImpServCipLicensesFranchisesTrademarks	Charges for the use of intellectual property n.i.e.; imports; franchises and trademarks licensing fee
ImpServCipLicensesOutcomesResearchAndDev	Charges for the use of intellectual property n.i.e.; imports; licenses for the use of outcomes of research and development
ImpServCipOth	Charges for the use of intellectual property n.i.e.; imports; other intellectual property
ImpServCipTrademarkFranchiseFees	Charges for the use of intellectual property n.i.e.; imports; trademarks and franchise fees
ImpServComp	Imports of computer services
ImpServFinancial	Imports of financial services
ImpServFinCredCardOthCredRelated	Imports of credit card and other credit-related services
ImpServFinFinManFinAdvCust	Imports of financial management, financial advisory, and custody services
ImpServFinSecBrokUwRelated	Imports of securities brokerage, underwriting, and related services
ImpServFinSecLendEftOth	Imports of securities lending, electronic funds transfer, and other services
ImpServGovtGoodsAndServicesNie	Imports of government goods and services n.i.e.
ImpServInfo	Imports of information services
ImpServInsurance	Imports of insurance services
ImpServInsuranceAuxIns	Imports of auxiliary insurance services
ImpServInsuranceDirect	Imports of direct insurance services
ImpServInsuranceReIns	Imports of reinsurance services
ImpServMaintenanceAndRepairNie	Imports of maintenance and repair services n.i.e.
ImpServOtherBusiness	Imports of other business services
ImpServProfMgmtConsult	Imports of professional and management consulting services
ImpServResearchAndDev	Imports of research and development services

Table 22.39 *continued*

Series Indicator	Description
ImpServTechTradeRelatedOth	Imports of technical, trade-related, and other business services
ImpServTelecom	Imports of telecommunications services
ImpServTelecomCompAndInfo	Imports of telecommunications, computer, and information services
ImpServTransport	Imports of transport services
ImpServTransportAir	Imports of air transport services
ImpServTransportAirFreight	Imports of air freight services
ImpServTransportAirPass	Imports of air passenger services
ImpServTransportAirPort	Imports of airport services
ImpServTransportOth	Imports of transport services; other modes of transport

Table 22.40 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
ImpServTransportSea	Imports of sea transport services
ImpServTransportSeaFreight	Imports of sea freight services
ImpServTransportSeaPort	Imports of seaport services
ImpServTravel	Imports of travel services (for all purposes including education)
ImpServTravelBusiness	Imports of business travel services
ImpServTravelBusinessOth	Imports of other business travel services
ImpServTravelEducation	Imports of education-related services
ImpServTravelHealth	Imports of health-related services
ImpServTravelPersonal	Imports of personal travel services
ImpServTravelPersonalOth	Imports of other personal travel services
ImpServTravelShortTermWork	Expenditures abroad by border, seasonal, and other short-term workers
InsLossesPaid	Insurance losses paid
InsLossesRecovered	Insurance losses recovered
InsPremiumsPaid	Insurance premiums paid
InsPremiumsReceived	Insurance premiums received
InsTechReservesAssets	Net US acquisition of other investment assets; insurance technical reserves
InsTechReservesAssetsOthFinNonFin	Net US acquisition of other investment assets; insurance technical reserves; held by non-deposit-taking institutions and nonfinancial institutions except general government
InsTechReservesLiabs	Net US incurrence of other investment liabilities; insurance technical reserves
InsTechReservesLiabsOthFinNonFin	Net US incurrence of other investment liabilities;

Table 22.40 *continued*

Series Indicator	Description
	insurance technical reserves; issued by non-deposit-taking institutions and nonfinancial institutions except general government
InvFundSharesAssets	Net US acquisition of portfolio investment assets; investment fund shares
InvFundSharesIncPay	Portfolio investment income payments; income attributable to investment fund shareholders
InvFundSharesIncRec	Portfolio investment income receipts; income attributable to investment fund shareholders
InvFundSharesLiabs	Net US incurrence of portfolio investment liabilities; investment fund shares
InvFundSharesLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; investment fund shares
InvIncPay	Investment income payments
InvIncRec	Investment income receipts
InvRaIncRec	Reserve asset income receipts
InvRaIntIncRec	Reserve asset income receipts; interest
LoansAssets	Net US acquisition of other investment assets; loans
LoansAssetsDepTaking	Net US acquisition of other investment assets; loans; held by deposit-taking institution

Table 22.40 *continued*

Series Indicator	Description
LoansAssetsGenGovt	Net US acquisition of other investment assets; loans; held by general government
LoansAssetsOthFinNonFin	Net US acquisition of other investment assets; loans; held by non-deposit-taking institutions
LoansLiabs	Net US incurrence of other investment liabilities; loans
LoansLiabsDepTaking	Net US incurrence of other investment liabilities; loans; issued by deposit-taking institutions except central bank
LoansLiabsFoa	Net US incurrence of other investment liabilities to foreign official agencies; loans
LoansLiabsOthFinNonFin	Net US incurrence of other investment liabilities; loans; issued by non-deposit-taking institutions and nonfinancial institutions except general government
LoansRepurchaseLiabsOthFinNonFin	Net US incurrence of other investment liabilities; loans, of which repurchase agreements; issued by non-deposit-taking institutions and nonfinancial institutions except general government
LoansResaleAssetsOthFinNonFin	Net US acquisition of other investment assets; loans, of which resale agreements; held by non-deposit-taking institutions and nonfinancial institutions except general government

Table 22.41 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
LtDebtSecAssets	Net US acquisition of portfolio investment assets; long-term debt securities
LtDebtSecAssetsDepTaking	Net US acquisition of portfolio investment assets; long-term debt securities; held by deposit-taking institutions except central bank
LtDebtSecAssetsNonFin	Net US acquisition of portfolio investment assets; long-term debt securities; held by nonfinancial institutions except general government
LtDebtSecAssetsOthFin	Net US acquisition of portfolio investment assets; long-term debt securities; held by non-deposit-taking financial institutions
LtDebtSecCorpAssets	Net US acquisition of portfolio investment assets; corporate bonds and notes
LtDebtSecCorpLiabs	Net US incurrence of portfolio investment liabilities; corporate bonds and notes

Table 22.41 *continued*

Series Indicator	Description
LtDebtSecCorpLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; corporate bonds and notes
LtDebtSecFedSponsorAgencyIncPay	Portfolio investment income payments; interest on long-term federally sponsored agency securities
LtDebtSecFedSponsorAgencyLiabs	Net US incurrence of portfolio investment liabilities; long-term federally sponsored agency securities
LtDebtSecFedSponsorAgencyLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; long-term federally sponsored agency securities
LtDebtSecGovtAssets	Net US acquisition of portfolio investment assets; long-term government debt securities
LtDebtSecIncPay	Portfolio investment income payments; interest on long-term debt securities
LtDebtSecIncPayDepTaking	Portfolio investment income payments; interest on long-term debt securities; deposit-taking institutions except central bank
LtDebtSecIncPayNonFin	Portfolio investment income payments; interest on long-term debt securities; nonfinancial institutions except general government
LtDebtSecIncRec	Portfolio investment income receipts; interest on long-term debt securities
LtDebtSecIncRecDepTaking	Portfolio investment income receipts; interest on long-term debt securities; deposit-taking institutions except central bank
LtDebtSecIncRecNonFin	Portfolio investment income receipts; interest on long-term debt securities; nonfinancial institutions except general government
LtDebtSecIncRecOthFin	Portfolio investment income receipts; interest on long-term debt securities; non-deposit-taking financial institutions
LtDebtSecLiabs	Net US incurrence of portfolio investment liabilities; long-term debt securities
LtDebtSecLiabsDepTaking	Net US incurrence of portfolio investment liabilities; long-term debt securities; issue
LtDebtSecLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies;

Table 22.41 *continued*

Series Indicator	Description
LtDebtSecLiabsNonFin	Net US incurrence of portfolio investment liabilities; long-term debt securities; issued by nonfinancial institutions except general government
LtDebtSecNegCdAssets	Net US acquisition of portfolio investment assets; long-term negotiable certificates of deposit
LtDebtSecNegCdLiabs	Net US incurrence of portfolio investment liabilities; long-term negotiable certificates of deposit
LtDebtSecNegCdLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; long-term negotiable certificates of deposit
LtDebtSecOthThanFedSponsorAgencyIncPayOthFin	Portfolio investment income payments; interest on long-term debt securities other than federally sponsored agency securities; non-deposit-taking financial institutions
LtDebtSecOthThanFedSponsorAgencyLiabsOthFin	Net US incurrence of portfolio investment liabilities; long-term debt securities other than federally sponsored agency securities; issued by non-deposit-taking financial institutions

Table 22.42 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
LtDebtSecStateLocalGovtIncPay	Portfolio investment income payments; interest on state and local government long-term securities
LtDebtSecStateLocalGovtLiabs	Net US incurrence of portfolio investment liabilities; state and local government long-term securities
LtDebtSecStateLocalGovtLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; state and local government long-term securities
LtDebtSecTreasIncPay	Portfolio investment income payments; interest on long-term US Treasury securities
LtDebtSecTreasLiabs	Net US incurrence of portfolio investment liabilities; Treasury bonds and notes
LtDepAssets	Net US acquisition of other investment assets; long-term deposits
LtDepAssetsDepTaking	Net US acquisition of other investment assets; long-term deposits; held by deposit-taking institutions except central bank

Table 22.42 *continued*

Series Indicator	Description
LtDepAssetsOthFinNonFin	Net US acquisition of other investment assets; long-term deposits; held by non-deposit-taking institutions and nonfinancial institutions except general government
LtDepLiabs	Net US incurrence of other investment liabilities; long-term deposits
LtDepLiabsDepTaking	Net US incurrence of other investment liabilities; long-term deposits; issued by deposit-taking institutions except central bank
LtDepLiabsOthFinNonFin	Net US incurrence of other investment liabilities; long-term deposits; issued by non-deposit-taking institutions and nonfinancial institutions except general government
LtLoansAssets	Net US acquisition of other investment assets; long-term loans
LtLoansAssetsDepTaking	Net US acquisition of other investment assets; long-term loans; held by deposit-taking institutions except central bank
LtLoansAssetsGenGovt	Net US acquisition of other investment assets; long-term loans; held by general government
LtLoansAssetsOthFinNonFin	Net US acquisition of other investment assets; long-term loans; held by non-deposit-taking institutions and nonfinancial institutions except general government
LtLoansLiabs	Net US incurrence of other investment liabilities; long-term loans
LtLoansLiabsDepTaking	Net US incurrence of other investment liabilities; long-term loans; issued by deposit-taking institutions except central bank
LtLoansLiabsOthFinNonFin	Net US incurrence of other investment liabilities; long-term loans; issued by non-deposit-taking institutions and nonfinancial institutions except general government
LtTrdCredAndAdvAssets	Net US acquisition of other investment assets; long-term trade credit and advances

Table 22.42 *continued*

Series Indicator	Description
LtTrdCredAndAdvAssetsOthFinNonFin	Net US acquisition of other investment assets; long-term trade credit and advances; held by non-deposit-taking institutions and nonfinancial institutions except general government
LtTrdCredAndAdvLiabs	Net US incurrence of other investment liabilities; long-term trade credit and advances
LtTrdCredAndAdvLiabsGenGovt	Net US incurrence of other investment liabilities; long-term trade credit and advances; issued by general government
LtTrdCredAndAdvLiabsOthFinNonFin	Net US incurrence of other investment liabilities; long-term trade credit and advances; issued by non-deposit-taking institutions and nonfinancial institutions except general government

Table 22.43 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
NetLendBorrCurrCapAcct	Net lending (+) or net borrowing (-) from current- and capital-account transactions
NetLendBorrFinAcct	Net lending (+) or net borrowing (-) from financial-account transactions
OthCImReserveAssets	Net US acquisition of reserve assets; other; other claims
OthInvAssets	Net US acquisition of other investment assets
OthInvAssetsCentralBank	Net US acquisition of other investment assets; held by central bank
OthInvAssetsDepTaking	Net US acquisition of other investment assets; held by deposit-taking institutions except central bank
OthInvAssetsGenGovt	Net US acquisition of other investment assets; held by general government
OthInvAssetsOthFinNonFin	Net US acquisition of other investment assets; held by non-deposit-taking institutions and nonfinancial institutions except general government
OthInvIncPay	Other investment income payments
OthInvIncPayCentralBank	Other investment income payments; on liabilities issued by central bank
OthInvIncPayDepTaking	Other investment income payments; on liabilities issued by deposit-taking institutions
OthInvIncPayGenGovt	Other investment income payments;

Table 22.43 *continued*

Series Indicator	Description
OthInvIncPayOthFinNonFin	on liabilities issued by general government Other investment income payments on liabilities issued by non-deposit-taking financial institutions and nonfinancial institutions except general government
OthInvIncRec	Other investment income receipts
OthInvIncRecCentralBank	Other investment income receipts; on assets held by central bank
OthInvIncRecDepTaking	Other investment income receipts; on assets held by deposit-taking institutions except central bank
OthInvIncRecGenGovt	Other investment income receipts; on assets held by general government
OthInvIncRecOthFinNonFin	Other investment income receipts; on assets held by non-deposit-taking financial institutions and nonfinancial institutions except general government
OthInvInsPolHoldtIncPay	Other investment income payments; income attributable to insurance policyholders
OthInvInsPolHoldtIncPayOthFinNonFin	Other investment income payments; income attributable to insurance policyholders on liabilities issued by non-deposit-taking institutions and nonfinancial institutions except general government
OthInvInsPolHoldtIncRec	Other investment income receipts; income attributable to insurance policyholders
OthInvInsPolHoldtIncRecOthFinNonFin	Other investment income receipts; income attributable to insurance policyholders; on assets held by non-deposit-taking institutions and nonfinancial institutions except general government

Table 22.44 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
OthInvInterbankAssets	Net US acquisition of other investment assets; interbank transactions
OthInvInterbankLiabsDepTaking	Net US incurrence of other investment liabilities; interbank transactions
OthInvIntIncPay	Other investment income payments; interest
OthInvIntIncPayCentralBank	Other investment income payments; interest; on liabilities issued by central bank
OthInvIntIncPayDepTaking	Other investment income payments; interest; on liabilities issued by deposit-taking institutions

Table 22.44 *continued*

Series Indicator	Description
OthInvIntIncPayGenGovt	except central bank Other investment income payments; interest on special drawing rights allocations
OthInvIntIncPayOthFinNonFin	Other investment income payments; interest; on liabilities issued by non-deposit-taking financial institutions and nonfinancial institutions except general government
OthInvIntIncRec	Other investment income receipts; interest
OthInvIntIncRecCentralBank	Other investment income receipts; interest; on assets held by central bank
OthInvIntIncRecDepTaking	Other investment income receipts; interest; on assets held by deposit-taking institutions
OthInvIntIncRecGenGovt	Other investment income receipts; interest; on assets held by general government
OthInvIntIncRecOthFinNonFin	Other investment income receipts; interest; on assets held by non-deposit-taking financial institutions and nonfinancial institutions except general government
OthInvLiabs	Net US incurrence of other investment liabilities
OthInvLiabsCentralBank	Net US incurrence of other investment liabilities; issued by central bank
OthInvLiabsDepTaking	Net US incurrence of other investment liabilities; issued by deposit-taking institutions
OthInvLiabsFoa	Net US incurrence of other investment liabilities to foreign official agencies
OthInvLiabsGenGovt	Net US incurrence of other investment liabilities; issued by general government
OthInvLiabsOthFinNonFin	Net US incurrence of other investment liabilities; issued by non-deposit-taking institutions and nonfinancial institutions except general government
OthReserveAssets	Net US acquisition of reserve assets; other
PfInvAssets	Net US acquisition of portfolio investment assets
PfInvAssetsDepTaking	Net US acquisition of portfolio investment assets; held by deposit-taking institutions
PfInvAssetsNonFin	Net US acquisition of portfolio investment assets; held by nonfinancial institutions
PfInvAssetsOthFin	Net US acquisition of portfolio investment assets; held by non-deposit-taking financial
PfInvIncPay	Portfolio investment income payments
PfInvIncPayDepTaking	Portfolio investment income payments; deposit-taking institutions except central bank
PfInvIncPayGenGovt	Portfolio investment income payments; general government

Table 22.44 *continued*

Series Indicator	Description
PfInvIncPayNonFin	Portfolio investment income payments; nonfinancial institutions except general government
PfInvIncPayOthFin	Portfolio investment income payments; non-deposit-taking financial institutions
PfInvIncRec	Portfolio investment income receipts
PfInvIncRecDepTaking	Portfolio investment income receipts; deposit-taking institutions except central bank
PfInvIncRecNonFin	Portfolio investment income receipts; nonfinancial institutions except general government
PfInvIncRecOthFin	Portfolio investment income receipts; non-deposit-taking financial institutions
PfInvLiabs	Net US incurrence of portfolio investment liabilities
PfInvLiabsDepTaking	Net US incurrence of portfolio investment liabilities; issued by deposit-taking institutions except central bank
PfInvLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies
PfInvLiabsGenGovt	Net US incurrence of portfolio investment liabilities; issued by general government
PfInvLiabsNonFin	Net US incurrence of portfolio investment liabilities; issued by nonfinancial institutions except general government
PfInvLiabsOthFin	Net US incurrence of portfolio investment liabilities; issued by non-deposit-taking financial institutions
PrimIncPay	Primary income payments
PrimIncRec	Primary income receipts

Table 22.45 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
ReserveAssets	Net US acquisition of reserve assets
SdrAllocLiabs	Net US incurrence of other investment liabilities; special drawing rights allocations
SdrAllocLiabsFoa	Net US incurrence of other investment liabilities to foreign official agencies; special drawing rights allocations
SdrReserveAssets	Net US acquisition of reserve assets; special drawing rights
SeasAdjDisc	Seasonal adjustment discrepancy

Table 22.45 *continued*

Series Indicator	Description
SecIncPay	Secondary income (current transfer) payments; private transfers; transfers to foreign students
SecIncPayCharitableDonations	Secondary income (current transfer) payments; private transfers; charitable donations
SecIncPayForeignStudents	Secondary income (current transfer) payments; private transfers; transfers to foreign students
SecIncPayGenGovt	Secondary income (current transfer) payments; general government transfers
SecIncPayGenGovtContribIntOrg	Secondary income (current transfer) payments; general government transfers; contributions to international organizations
SecIncPayGenGovtIntCoop	Secondary income (current transfer) payments; general government transfers; international cooperation
SecIncPayGenGovtOth	Secondary income (current transfer) payments; other general government transfers
SecIncPayGenGovtSocialBenefits	Secondary income (current transfer) payments; general government transfers; social benefits
SecIncPayInsuranceRelated	Secondary income (current transfer) payments; private transfers; insurance-related transfers
SecIncPayOthPrivate	Secondary income (current transfer) payments; private transfers other than personal transfers
SecIncPayOthPrivateTransfer	Secondary income (current transfer) payments; other private transfers
SecIncPayPersonal	Secondary income (current transfer) payments; personal transfers
SecIncPayPrivate	Secondary income (current transfer) payments; private transfers
SecIncPayPrivateFinesPenalties	Secondary income (current transfer) payments; private transfers; fines and penalties
SecIncPayTaxesIncomeWealth	Secondary income (current transfer) payments; private transfers; taxes on income, wealth, etc.
SecIncPayUSGovt	Secondary income (current transfer) payments; US government transfers
SecIncPayUSGovtGrants	Secondary income (current transfer) payments; US government grants
SecIncPayUSGovtPensionsAndOth	Secondary income (current transfer) payments; US government pensions and other transfer
SecIncRec	Secondary income (current transfer) receipts
SecIncRecGenGovt	Secondary income (current transfer) receipts; general government transfer
SecIncRecGenGovtFinesPenalties	Secondary income (current transfer) receipts; general government transfers; fines and penalties
SecIncRecGenGovtIntCoop	Secondary income (current transfer) receipts;

Table 22.45 *continued*

Series Indicator	Description
SecIncRecGenGovtOth	general government transfers; international cooperation Secondary income (current transfer) receipts; other general government transfers
SecIncRecGenGovtTaxesIncomeWealth	Secondary income (current transfer) receipts; general government transfers; taxes on income, wealth, etc.
SecIncRecInsuranceRelated	Secondary income (current transfer) receipts; insurance-related transfers
SecIncRecOthPrivateTransfer	Secondary income (current transfer) receipts; other private transfers
SecIncRecPrivate	Secondary income (current transfer) receipts; private transfers
SecIncRecPrivateFinesPenalties	Secondary income (current transfer) receipts; fines and penalties

Table 22.46 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
SecIncRecUSGovt	Secondary income (current transfer) receipts; US government transfers
SecReserveAssets	Net US acquisition of reserve assets; other; securities
StatDisc	Statistical discrepancy
StDebtSecAssets	Net US acquisition of portfolio investment assets; short-term debt securities
StDebtSecAssetsDepTaking	Net US acquisition of portfolio investment assets; short-term debt securities; held by deposit-taking institutions except central bank
StDebtSecAssetsNonFin	Net US acquisition of portfolio investment assets; short-term debt securities; held by nonfinancial institutions except general government
StDebtSecAssetsOthFin	Net US acquisition of portfolio investment assets; short-term debt securities; held by non-deposit-taking financial institutions
StDebtSecCommPaperAndOthLiabs	Net US incurrence of portfolio investment liabilities; commercial paper and other short-term debt securities (those not listed in table 7.1)
StDebtSecCommPaperAndOthLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; commercial paper and other short-term debt securities (those not listed in table 9.1)

Table 22.46 *continued*

Series Indicator	Description
StDebtSecCommPaperAssets	Net US acquisition of portfolio investment assets; short-term commercial paper
StDebtSecFedSponsorAgencyIncPay	Portfolio investment income payments; interest on short-term federally sponsored agency securities
StDebtSecFedSponsorAgencyLiabs	Net US incurrence of portfolio investment liabilities; short-term federally sponsored agency securities
StDebtSecFedSponsorAgencyLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; short-term federally sponsored agency securities
StDebtSecIncPay	Portfolio investment income payments; interest on short-term debt securities
StDebtSecIncPayDepTaking	Portfolio investment income payments; interest on short-term debt securities; deposit-taking institutions except central bank
StDebtSecIncPayNonFin	Portfolio investment income payments; interest on short-term debt securities; nonfinancial institutions except general government
StDebtSecIncRec	Portfolio investment income receipts; interest on short-term debt securities
StDebtSecIncRecDepTaking	Portfolio investment income receipts; interest on short-term debt securities; deposit-taking institutions except central bank
StDebtSecIncRecNonFin	Portfolio investment income receipts; interest on short-term debt securities; nonfinancial institutions except general government
StDebtSecIncRecOthFin	Portfolio investment income receipts; interest on short-term debt securities; non-deposit-taking financial institutions
StDebtSecLiabs	Net US incurrence of portfolio investment liabilities; short-term debt securities
StDebtSecLiabsDepTaking	Net US incurrence of portfolio investment liabilities; short-term debt securities; issued by deposit-taking institutions except central bank
StDebtSecLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; short-term debt securities
StDebtSecLiabsNonFin	Net US incurrence of portfolio investment liabilities; short-term debt securities; issued by nonfinancial institutions except general government

Table 22.46 *continued*

Series Indicator	Description
StDebtSecNegCdAssets	Net US acquisition of portfolio investment assets; short-term negotiable certificates of deposit
StDebtSecNegCdLiabs	Net US incurrence of portfolio investment liabilities; short-term negotiable certificates of deposit
StDebtSecNegCdLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; short-term negotiable certificates of deposit
StDebtSecOthAssets	Net US acquisition of portfolio investment assets; short-term debt securities other than negotiable certificates of deposit and commercial paper
StDebtSecOthThanFedSponsorAgencyIncPayOthFin	Portfolio investment income payments; interest on short-term debt securities other than federally sponsored agency securities; non-deposit-taking financial institutions
StDebtSecOthThanFedSponsorAgencyLiabsOthFin	Net US incurrence of portfolio investment liabilities; short-term debt securities other than federally sponsored agency securities; issued by non-deposit-taking financial institutions

Table 22.47 ITA Series Indicators and Descriptions (*continued*)

Series Indicator	Description
StDebtSecTreasIncPay	Portfolio investment income payments; interest on short-term US Treasury securities
StDebtSecTreasLiabs	Net US incurrence of portfolio investment liabilities; Treasury bills and certificates
StDebtSecTreasLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; Treasury bills and certificates
StDepAssets	Net US acquisition of other investment assets; short-term deposits
StDepAssetsCentralBank	Net US acquisition of other investment assets; short-term deposits; held by central bank
StDepAssetsDepTaking	Net US acquisition of other investment assets; short-term deposits; held by deposit-taking institutions except central bank
StDepAssetsOthFinNonFin	Net US acquisition of other investment assets; short-term deposits; held by non-deposit-taking institutions and nonfinancial institutions except general government
StDepLiabs	Net US incurrence of other investment liabilities; short-term deposits

Table 22.47 *continued*

Series Indicator	Description
StDepLiabsCentralBank	Net US incurrence of other investment liabilities; short-term deposits; issued by central bank
StDepLiabsDepTaking	Net US incurrence of other investment liabilities; short-term deposits; issued by deposit-taking institutions except central bank
StDepLiabsOthFinNonFin	Net US incurrence of other investment liabilities; short-term deposits; issued by non-deposit-taking institutions and nonfinancial institutions except general government
StLoansAssets	Net US acquisition of other investment assets; short-term loans
StLoansAssetsDepTaking	Net US acquisition of other investment assets; short-term loans; held by deposit-taking institutions except central bank
StLoansAssetsOthFinNonFin	Net US acquisition of other investment assets; short-term loans; held by non-deposit-taking institutions and nonfinancial institutions except general government
StLoansLiabs	Net US incurrence of other investment liabilities; short-term loans
StLoansLiabsDepTaking	Net US incurrence of other investment liabilities; short-term loans; issued by deposit-taking institutions except central bank
StLoansLiabsOthFinNonFin	Net US incurrence of other investment liabilities; short-term loans; issued by non-deposit-taking institutions and nonfinancial institutions except general government
StTrdCredAndAdvAssets	Net US acquisition of other investment assets; short-term trade credit and advances
StTrdCredAndAdvAssetsOthFinNonFin	Net US acquisition of other investment assets; short-term trade credit and advances; held by non-deposit-taking institutions and nonfinancial institutions except general government
StTrdCredAndAdvLiabs	Net US incurrence of other investment liabilities; short-term trade credit and advances
StTrdCredAndAdvLiabsOthFinNonFin	Net US incurrence of other investment liabilities; short-term trade credit and advances; issued by non-deposit-taking institutions and nonfinancial institutions except general government
TrdCredAndAdvAssets	Net US acquisition of other investment assets; trade credit and advances
TrdCredAndAdvAssetsOthFinNonFin	Net US acquisition of other investment assets;

Table 22.47 *continued*

Series Indicator	Description
TrdCredAndAdvLiabs	trade credit and advances; held by non-deposit-taking institutions and nonfinancial institutions except general government Net US incurrence of other investment liabilities; trade credit and advances
TrdCredAndAdvLiabsFoa	Net US incurrence of other investment liabilities to foreign official agencies; trade credit and advances
TrdCredAndAdvLiabsGenGovt	Net US incurrence of other investment liabilities; trade credit and advances; issued by general government
TrdCredAndAdvLiabsOthFinNonFin	Net US incurrence of other investment liabilities; trade credit and advances; issued by non-deposit-taking institutions and nonfinancial institutions except general government
TreasBondsAndNotesLiabsFoa	Net US incurrence of portfolio investment liabilities to foreign official agencies; Treasury bonds and notes
TSI_ItaDiInvIncInward	Direct investment income on inward investment (foreign direct investment in the United States)
TSI_ItaDiInvIncOutward	Direct investment income on outward investment (US direct investment abroad), directional basis
TSI_ItaDiInvWithoutCurrCostAdjIncOutward	Direct investment income without current-cost adjustment on outward investment (US direct investment abroad), directional basis

The counterparty area or country codes available for the International Transactions Accounts (ITA) data set are listed in Table 22.48 and Table 22.49.

For more information about using area or country codes, visit the web page at the following URL: <https://apps.bea.gov/international/factsheet/docs/Country%20Facts%20Data%20Availability.pdf>

Table 22.48 ITA Counterparty Area or Country Codes

Country Code	Country Code	Country Code
Africa	CostaRica	IntOrgAndUnalloc
AfricaOthGdsNsaDetail	Croatia	Ireland
Algeria	Cyprus	Israel
AllCountries	CzechRep	Italy
AllOthSeas	Denmark	Japan
AllOthThanCanada	DominicanRep	Jordan
Argentina	ElSalvador	LatAmAndOthWestHem
AsiaAndPac	Estonia	Latvia
AsiaAndPacOthNsaDetail	EU	Lithuania
Australia	EuroArea	Luxembourg
Austria	Europe	Malaysia
Bahrain	EuropeOthNsaDetail	Malta
Belgium	Finland	Mexico
Bermuda	France	MiddleEast
Brazil	Germany	MiddleEastOthGdsNsaDetail
Brunei	Greece	Morocco
Bulgaria	Guatemala	Netherlands
CaftaDrCountries	Honduras	NewZealand
Canada	HongKong	Nicaragua
Chile	Hungary	Nigeria
China	India	Norway
Colombia	Indonesia	Oman

Table 22.49 ITA Counterparty Area or Country Codes

Country Code	Country Code
OthAfricaExcl1	ResidualSeas
OthAfricaExcl3	Romania
OthAfricaIst	Russia
OthAsiaAndPacExcl15	SaudiArabia
OthAsiaAndPacExcl8	Singapore
OthAsiaAndPacIst	Slovakia
OthEuropeExcl32	Slovenia
OthEuropeExcl7	SouthAfrica
OthEuropeIst	SouthAndCenAm
OthMiddleEastExcl5	SouthAndCenAmOthNsaDetail

Table 22.49 *continued*

Country Code	Country Code
OthMiddleEastIst	SouthKorea
OthSouthAndCenAmExcl13	Spain
OthSouthAndCenAmExcl4	Sweden
OthSouthAndCenAmIst	Switzerland
OthWestHem	Taiwan
OthWestHemOthExcl3	Thailand
OthWestHemOthIst	Turkey
Panama	UkIslandsCarib
Peru	UnitedKingdom
Philippines	Venezuela
Poland	Vietnam
Portugal	

International Investment Position Data

The available investment types and components for the International Investment Position (IIP) data set are listed in Table 22.50 and Table 22.51.

For more details, visit the web page at the following URL: <https://www.bea.gov/sites/default/files/methodologies/ONE%20PDF%20-%20IEA%20Concepts%20Methods.pdf>

Table 22.50 International Investment Position (IIP) Investment Types

Type of Investment	Investment Description
CurrAndDepAssets	US assets; other investment; currency and deposits
CurrAndDepLiabs	US liabilities; other investment; currency and deposits
CurrAndDepLiabsFoa	US liabilities to foreign official agencies; other investment; currency and deposits
CurrAndDepReserveAssets	US assets; other reserve assets; currency and deposits
DebtSecAssets	US assets; portfolio investment; debt securities
DebtSecLiabs	US liabilities; portfolio investment; debt securities
DebtSecLiabsFoa	US liabilities to foreign official agencies; portfolio investment; debt securities
DiInvAssets	US assets; direct investment at market value, asset/liability basis
DiInvAssetsCurrCost	US assets; direct investment at current cost, asset/liability basis
DiInvAssetsHistCostToMarketValueAdj	US assets; direct investment; adjustment to revalue equity from historical cost to market value
DiInvDebtInstAssets	US assets; direct investment, asset/liability basis; debt instruments
DiInvDebtInstInward	Inward direct investment (foreign direct investment in the United States), directional basis; debt instruments

Table 22.50 *continued*

Type of Investment	Investment Description
DiInvDebtInstLiabs	US liabilities; direct investment, asset/liability basis; debt instruments
DiInvDebtInstOutward	Outward direct investment (US direct investment abroad), directional basis; debt instruments
DiInvDebtInstUsAffiliatesClaims	Direct investment; debt instruments; US affiliates' claims
DiInvDebtInstUsAffiliatesLiabs	Direct investment; debt instruments; US affiliates' liabilities
DiInvDebtInstUsParentsClaims	Direct investment; debt instruments; US parents' claims
DiInvDebtInstUsParentsLiabs	Direct investment; debt instruments; US parents' liabilities
DiInvDirectionalBasisAdj	Direct investment; adjustments to convert to directional basis
DiInvEquityAssets	US assets; direct investment at market value; equity
DiInvEquityAssetsCurrCost	US assets; direct investment at current cost; equity
DiInvEquityAssetsHistCost	US assets; direct investment at historical cost; equity
DiInvEquityInward	US liabilities; direct investment at market value; equity
DiInvEquityLiabs	US liabilities; direct investment at market value; equity
DiInvEquityLiabsCurrCost	US liabilities; direct investment at current cost; equity
DiInvEquityLiabsHistCost	US liabilities; direct investment at historical cost; equity
DiInvInwardCurrCost	Inward direct investment (foreign direct investment in the United States) at current cost, directional basis
DiInvInwardHistCost	Inward direct investment (foreign direct investment in the United States) at historical cost, directional basis
DiInvInwardMarketValue	Inward direct investment (foreign direct investment in the United States) at market value, directional basis
DiInvLiabs	US liabilities; direct investment at market value, asset/liability basis
DiInvLiabsCurrCost	US liabilities; direct investment at current cost, asset/liability basis
DiInvLiabsHistCostToMarketValueAdj	US liabilities; direct investment; adjustment to revalue equity from historical cost to market value
DiInvOutwardCurrCost	Outward direct investment (US direct investment abroad) at current cost, directional basis
DiInvOutwardHistCost	Outward direct investment (US direct investment abroad) at historical cost, directional basis
DiInvOutwardMarketValue	Outward direct investment (US direct investment abroad) at market value, directional basis
EquityAndInvFundSharesAssets	US assets; portfolio investment; equity and investment fund shares
EquityAndInvFundSharesLiabs	US liabilities; portfolio investment; equity and investment fund shares
EquityAndInvFundSharesLiabsFoa	US liabilities to foreign official agencies; portfolio investment; equity and investment fund shares
FinAssets	US assets
FinAssetsExclFinDeriv	US assets excluding financial derivatives
FinDerivAssets	US assets; financial derivatives other than reserves, gross positive fair value
FinDerivExchTradedAssets	US assets; financial derivatives other than reserves, gross positive fair value; exchange-traded contracts

Table 22.50 *continued*

Type of Investment	Investment Description
FinDerivExchTradedLiabs	US liabilities; financial derivatives other than reserves, gross negative fair value; exchange-traded contracts
FinDerivForExAssets	US assets; financial derivatives other than reserves, gross positive fair value; over-the-counter contracts; foreign exchange contracts
FinDerivForExLiabs	US liabilities; financial derivatives other than reserves, gross negative fair value; over-the-counter contracts; foreign exchange contracts
FinDerivLiabs	US liabilities; financial derivatives other than reserves, gross negative fair value
FinDerivNet	US net international investment position; financial derivatives other than reserves
FinDerivOtcAssets	US assets; financial derivatives other than reserves, gross positive fair value; over-the-counter contracts
FinDerivOtcLiabs	US liabilities; financial derivatives other than reserves, gross negative fair value; over-the-counter contracts
FinDerivOthAssets	US assets; financial derivatives other than reserves, gross positive fair value; other over-the-counter contracts
FinDerivOthLiabs	US liabilities; financial derivatives other than reserves, gross negative fair value; other over-the-counter contracts
FinDerivReserveAssets	US assets; other reserve assets; financial derivatives
FinDerivSingleCurrAssets	US assets; financial derivatives other than reserves, gross positive fair value; over-the-counter contracts; single-currency interest rate contracts
FinDerivSingleCurrLiabs	US liabilities; financial derivatives other than reserves, gross negative fair value; over-the-counter contracts; single-currency interest rate contracts
FinLiabs	US liabilities
FinLiabsExclFinDeriv	US liabilities excluding financial derivatives
FinLiabsFoa	US liabilities to foreign official agencies
GoldReserveAssets	US assets; reserve assets; monetary gold
ImfReserveAssets	US assets; reserve assets; reserve position in the International Monetary Fund
InsTechReservesAssets	US assets; other investment; insurance technical reserves
InsTechReservesLiabs	US liabilities; other investment; insurance technical reserves
LoansAssets	US assets; other investment; loans
LoansLiabs	US liabilities; other investment; loans
LoansLiabsFoa	US liabilities to foreign official agencies; other investment; loans
LtDebtSecAssets	US assets; portfolio investment; long-term debt securities
LtDebtSecLiabs	US liabilities; portfolio investment; long-term debt securities
LtDebtSecLiabsFoa	US liabilities to foreign official agencies; portfolio investment; long-term debt securities
LtDebtSecTreasLiabsFoa	US liabilities to foreign official agencies; portfolio investment; long-term debt securities
Net	US net international investment position

Table 22.50 *continued*

Type of Investment	Investment Description
NetExclFinDeriv	US net international investment position excluding financial derivatives
OthCImReserveAssets	US assets; other reserve assets; other claims
OthInvAssets	US assets; other investment
OthInvLiabs	US liabilities; other investment
OthInvLiabsFoa	US liabilities to foreign official agencies; other investment
OthLtDebtSecLiabs	US liabilities; portfolio investment; long-term debt securities excluding Treasury bonds and notes
OthLtDebtSecLiabsFoa	US liabilities to foreign official agencies; portfolio investment; long-term debt securities excluding Treasury bonds and notes
OthReserveAssets	US assets; other reserve assets
OthStDebtSecLiabs	US liabilities; portfolio investment; short-term debt securities excluding Treasury bills and certificates
OthStDebtSecLiabsFoa	US liabilities to foreign official agencies; portfolio investment; short-term debt securities excluding Treasury bills and certificates
PfInvAssets	US assets; portfolio investment
PfInvLiabs	US liabilities; portfolio investment
PfInvLiabsFoa	US liabilities to foreign official agencies; portfolio investment
ReserveAssets	US assets; reserve assets
SdrAllocLiabs	US liabilities; other investment; special drawing rights allocations
SdrAllocLiabsFoa	US liabilities to foreign official agencies; other investment; special drawing rights allocations
SdrReserveAssets	US assets; reserve assets; special drawing rights
SecReserveAssets	US assets; other reserve assets; securities
StDebtSecAssets	US assets; portfolio investment; short-term debt securities
StDebtSecLiabs	US liabilities; portfolio investment; short-term debt securities
StDebtSecLiabsFoa	US liabilities to foreign official agencies; portfolio investment; short-term debt securities
StDebtSecTreasLiabsFoa	US liabilities to foreign official agencies; portfolio investment; short-term debt securities; Treasury bills and certificates
TrdCredAndAdvAssets	US assets; other investment; trade credit and advances
TrdCredAndAdvLiabs	US liabilities; other investment; trade credit and advances
TrdCredAndAdvLiabsFoa	US liabilities to foreign official agencies; other investment; trade credit and advances
TreasBillsAndCertsLiabs	US liabilities; portfolio investment; short-term debt securities; Treasury bills and certificates
TreasBondsAndNotesLiabs	US liabilities; portfolio investment; long-term debt securities; Treasury bonds and notes

Table 22.51 International Investment Position (IIP) Component Values

Component	Component Description
ChgPos	Change in position
ChgPosNie	Change in position attributable to changes in volume and valuation n.i.e.
ChgPosOth	Change in position not attributable to financial-account transactions
ChgPosPrice	Change in position attributable to price changes
ChgPosTrans	Change in position attributable to financial-account transactions
ChgPosXRate	Change in position attributable to exchange-rate changes
Pos	Position

The Input-Output Statistics Data

The table names available for the InputOutput Statistics data set are listed in [Table 22.52](#). Each table is identified by a unique table ID code.

Table 22.52 InputOutput Statistics Table IDs and Descriptions

TableId	Description
56	Industry-by-Commodity Total Requirements, After Redefinitions (Sector Level)
57	Industry-by-Commodity Total Requirements, After Redefinitions (Summary Level)
58	Commodity-by-Commodity Total Requirements, After Redefinitions (Sector Level)
59	Commodity-by-Commodity Total Requirements, After Redefinitions (Summary Level)
60	Industry-by-Industry Total Requirements, After Redefinitions (Sector Level)
61	Industry-by-Industry Total Requirements, After Redefinitions (Summary Level)
258	Use of Commodities by Industries (Sector Level)
259	Use of Commodities by Industries (Summary Level)
261	Domestic Supply of Commodities by Industries (Sector Level)
262	Domestic Supply of Commodities by Industries (Summary Level)

International Trade in Services Data

The counterparty country or area names available for the International Trade in Services (IntlServTrade) data set are listed in [Table 22.53](#).

Table 22.53 IntlServTrade Data Set Counterparty Country or Area Names

Country or Area Name	Country or Area Description
Africa	Africa
AllCountries	All countries
Argentina	Argentina
AsiaAndPac	Asia and Pacific
Australia	Australia
Austria	Austria
Bahrain	Bahrain
Belgium	Belgium
Bermuda	Bermuda

Table 22.53 *continued*

Country or Area Name	Country or Area Description
Brazil	Brazil
Brunei	Brunei
Bulgaria	Bulgaria
CaftaDrCountries	Central America–Dominican Republic Free Trade Agreement countries
Canada	Canada
Chile	Chile
China	China
Colombia	Colombia
CostaRica	Costa Rica
Croatia	Croatia
Cyprus	Cyprus
CzechRep	Czech Republic
Denmark	Denmark
DominicanRep	Dominican Republic
ElSalvador	El Salvador
Estonia	Estonia
EU	European Union
EuroArea	Euro area
Europe	Europe
Finland	Finland
France	France
Germany	Germany
Greece	Greece
Guatemala	Guatemala
Honduras	Honduras
HongKong	Hong Kong
Hungary	Hungary
India	India
Indonesia	Indonesia
IntOrgAndUnalloc	International organizations and unallocated areas
Ireland	Ireland
Israel	Israel
Italy	Italy
Japan	Japan
Jordan	Jordan
KoreaRepOf	Republic of Korea
LatAmAndOthWestHem	Latin America and other Western Hemisphere countries
Latvia	Latvia
Lithuania	Lithuania
Luxembourg	Luxembourg
Malaysia	Malaysia
Malta	Malta
Mexico	Mexico

Table 22.53 *continued*

Country or Area Name	Country or Area Description
MiddleEast	Middle East
Morocco	Morocco
NaftaCountries	North American Free Trade Agreement countries
Netherlands	Netherlands
NewZealand	New Zealand
Nicaragua	Nicaragua
Nigeria	Nigeria
Norway	Norway
Oman	Oman
OthAfricaIst	Other Africa (International Trade in Services)
OthAsiaAndPacIst	Other Asia and Pacific (International Trade in Services)
OthEuropeIst	Other Europe (International Trade in Services)
OthMiddleEastIst	Other Middle East (International Trade in Services)
OthSouthAndCenAmIst	Other South and Central America (International Trade in Services)
OthWestHem	Other Western Hemisphere
OthWestHemOthIst	Other Western Hemisphere, Other (International Trade in Services)
Panama	Panama
Peru	Peru
Philippines	Philippines
Poland	Poland
Portugal	Portugal
Romania	Romania
Russia	Russia
SaudiArabia	Saudi Arabia
Singapore	Singapore
Slovakia	Slovakia
Slovenia	Slovenia
SouthAfrica	South Africa
SouthAndCenAm	South America and Central America
Spain	Spain
Sweden	Sweden
Switzerland	Switzerland
Taiwan	Taiwan
Thailand	Thailand
Turkey	Turkey
UkIslandsCarib	United Kingdom Islands, Caribbean
UnitedKingdom	United Kingdom
Venezuela	Venezuela
Vietnam	Vietnam

The affiliation classifications available for the International Trade in Services (IntlServTrade) data set are listed in [Table 22.54](#).

Table 22.54 IntlServTrade Affiliation Classifications

Affiliation	Affiliation Description
Affiliated	Affiliated trade
AllAffiliations	The total for all trade, whether affiliated or unaffiliated
Unaffiliated	Unaffiliated trade
USAffiliates	United States affiliates trade with their foreign parent groups
USParents	United States parents trade with their foreign affiliates

The trade direction classifications available for the International Trade in Services (IntlServTrade) data set are listed in [Table 22.55](#).

Table 22.55 IntlServTrade Trade Direction Classifications

Trade Direction	Description
Exports	Exports
Imports	Imports
Balance	Balance (exports less imports)
SupplementalIns	Supplemental detail on insurance transactions
All	All

The types of services available for the International Trade in Services (IntlServTrade) data set are listed in [Table 22.56](#).

Table 22.56 IntlServTrade Types of Services

Types of Services	Description
All	All
AccountAuditBookkeep	Accounting, auditing, and bookkeeping services
Advertising	Advertising services
AdvertisingAndRelated	Advertising and related services
AgForAndFish	Services incidental to agriculture, forestry, and fishing
AllTypesOfService	All types of service
ArchEngSciAndOthTech	Architectural, engineering, scientific, and other technical services
Architectural	Architectural services
ArtisticRelated	Artistic-related services
AudioVisual	Audiovisual services
AudVisOriginals	Audiovisual originals
AudVisOriginalsBooksAndSound	Books and sound recordings
AudVisOriginalsMoviesAndTV	Movies and television programming
AudVisProduction	Audiovisual production services
AudVisRightsToUse	Rights to use audiovisual products
AudVisRightsToUseBooksAndSound	Books and sound recordings
AudVisRightsToUseMoviesAndTV	Movies and television programming
BusMgmtConsPubRel	Business and management consulting and public relations services

Table 22.56 *continued*

Types of Services	Description
ChargesForTheUseOfIpNie	Charges for the use of intellectual property n.i.e.
CipCompSoftIct	Charges for the use of intellectual property n.i.e.; licenses to reproduce and/or distribute computer software (backcasted for ICT services)
CipLicensesAudVis	Charges for the use of intellectual property n.i.e.; licenses to reproduce and/or distribute audiovisual and related products
CipLicensesBooksSoundRecord	Charges for the use of intellectual property n.i.e.; books and sound recordings
CipLicensesBroadcastLiveRecord	Charges for the use of intellectual property n.i.e.; broadcasting and recording of live events
CipLicensesCompSoftware	Charges for the use of intellectual property n.i.e.; licenses to reproduce and/or distribute computer software
CipLicensesFranchiseFees	Charges for the use of intellectual property n.i.e.; franchise fees
CipLicensesFranchisesTrademarks	Charges for the use of intellectual property n.i.e.; franchises and trademarks licensing
CipLicensesMoviesTV	Charges for the use of intellectual property n.i.e.; movies and television programming
CipLicensesOutcomesResearchAndDev	Charges for the use of intellectual property n.i.e.; licenses for the use of outcomes of research and development
CipLicensesTrademarks	Charges for the use of intellectual property n.i.e.; trademarks
CloudCompAndDataStor	Cloud computing and data storage services
Comp	Computer services
Const	Construction services
ConstAbroadUS	Construction services abroad or in the United States
ConstExpend	Construction services; contractor's expenditures
DatabaseAndOthInfo	Database and other information services
Education	Education services
Engineering	Engineering services
FinAdvCust	Financial advisory and custody services
Financial	Financial services
FinCredCardOthCredRelated	Credit card and other credit-related services
FinExplicitAndOth	Explicitly charged and other financial services
FinFinMan	Financial management services
FinFisim	Financial intermediation services indirectly measured
FinSecBrokAndMM	Brokerage and market-making services
FinSecLendEftOth	Securities lending, electronic funds transfer, and other services
FinUwAndPP	Underwriting and private placement services
GovtGoodsAndServicesNie	Government goods and services n.i.e.
Health	Health services
HeritageAndRec	Heritage and recreational services
IctServ	ICT services
Info	Information services
InsLossesPaid	Insurance losses paid
InsLossesPaidDirIns	Insurance losses paid direct insurance

Table 22.56 *continued*

Types of Services	Description
InsLossesPaidReins	Insurance losses paid reinsurance
InsLossesRecovered	Insurance losses recovered
InsLossesRecoveredDirIns	Insurance losses recovered direct insurance
InsLossesRecoveredReins	Insurance losses recovered reinsurance
InsPremiumsPaid	Insurance premiums paid
InsPremiumsPaidDirIns	Insurance premiums paid direct insurance
InsPremiumsPaidReins	Insurance premiums paid reinsurance
InsPremiumsReceived	Insurance premiums received
InsPremiumsReceivedDirIns	Insurance premiums received direct insurance
InsPremiumsReceivedReins	Insurance premiums received reinsurance
Insurance	Insurance services
InsuranceAuxIns	Auxiliary insurance services
InsuranceDirect	Direct insurance services
InsuranceReIns	Reinsurance services
Legal	Legal services
LegalAccountMgmtConsAndPubRel	Legal, accounting, management consulting, and public relations services
MaintenanceAndRepairNie	Maintenance and repair services n.i.e.
Manufacturing	Manufacturing services on physical inputs owned by others
MarResAndPubOpinPoll	Market research and public opinion polling services
Mining	Mining services
NewsAgency	News agency services
OperatingLeasing	Operating leasing services
OthBusinessNie	Other business services n.i.e.
OthComp	Other computer services
OtherBusiness	Other business services
OthPersonalCulturalAndRecreational	Other personal, cultural, and recreational services
PersonalCulturalAndRecreational	Personal, cultural, and recreational services
PotIctEnServ	Potentially ICT-enabled services
PotIctEnServOthBusServ	Potentially ICT-enabled services within other business services
PotIctEnServPerCultRec	Potentially ICT-enabled services within personal, cultural, and recreational services
PotIctEnServTechTradeRelatedOth	Potentially ICT-enabled services within technical, trade-related, and other business services
ProfMgmtConsult	Professional and management consulting services
RDOthRD	Other research and development services
RDProvision	Provision of customized and non-customized research and development services
RDSaleRights	Sale of proprietary rights arising from research and development
RDStockKnowledge	Work undertaken on a systematic basis to increase the stock of knowledge
ResearchAndDev	Research and development services
SciAndOthTech	Scientific and other technical services
TechTradeRelatedOth	Technical, trade-related, and other business services

Table 22.56 *continued*

Types of Services	Description
Telecom	Telecommunications services
TelecomCompAndInfo	Telecommunications, computer, and information services
TradeExhAndSalesConv	Trade exhibition and sales convention services
TradeRelated	Trade-related services
Transport	Transport services
TransportAir	Air transport services
TransportAirFreight	Air freight services
TransportAirPass	Air passenger services
TransportAirPort	Airport services
TransportOth	Transport services; other modes of transport
TransportPostal	Transport services; other modes of transport; postal
TransportRoadAndOth	Transport services; other modes of transport; road and other transport
TransportSea	Sea transport services
TransportSeaFreight	Sea freight services
TransportSeaPort	Seaport services
Travel	Travel services
TravelBusiness	Business travel services
TravelBusinessOth	Other business travel services
TravelBusinessPersonalOth	Expenditures by border, seasonal, and other short-term workers
TravelEducation	Education-related services
TravelHealth	Health-related services
TravelPersonal	Personal travel services
TravelPersonalOth	Other personal travel services
TravelShortTermWork	Expenditures by border, seasonal, and other short-term workers
WasteTreatAndDePol	Waste treatment and de-pollution services
WasteTreatAndDePolAgAndMining	Waste treatment and de-pollution, agricultural, and mining services

Direct Investment Data and Activities of Multinational Enterprise Data

The series IDs available for the Direct Investment (DI) Multinational Enterprise (MNE) data set and the Activities of Multinational Enterprise (AMNE) data set are listed in [Table 22.57](#). Each series is identified by a unique code, which is an integer enclosed in quotes. The time series names are made up of the following concatenations in this order: 'SeriesID_', series ID, 'ROW_', row position, 'COL_', and column code.

SeriesID="0" is the first entry in [Table 22.57](#) and is used to retrieve a list of series IDs whose data match the criteria that are set by the options specified in the SASEBEA LIBNAME statement. The remaining series IDs are used to request data by subject, and they have integer codes that must be enclosed in quotes. They are listed with their descriptions in [Table 22.57](#).

Table 22.57 MNE Series IDs and Descriptions

Series ID	Description
0	All
16	Balance Sheet - Balance sheet data present items typically found on corporate balance sheets, including assets and liabilities.
69	Balance Sheet of New Affiliates - Balance sheet data present items typically found on corporate balance sheets, including assets and liabilities.
3	Capital Expenditures - The expenditures made by a firm to acquire, to add to, or to improve property, plant, and equipment (PPE).
7	Compensation of Employees - The income received from an employer as remuneration for work.
26	Debt Instruments Inflows - Transactions in debt instruments (previously called intercompany debt flows) are the result of changes in net outstanding loans between foreign parent companies and their US affiliates, including loans by parents to affiliates and loans by affiliates to parents.
33	Debt Instruments Outflows - Transactions in debt instruments (previously called intercompany debt flows) are the result of changes in net outstanding loans between US parent companies and their foreign affiliates, including loans by parents to affiliates and loans by affiliates to parents.
27	Direct Investment Income Without Current-Cost Adjustment - Income is the return on the direct investment position. It consists of earnings, which include the parents' shares in the net income from the operations of their affiliates and net interest received by parents from affiliates from outstanding loans and trade accounts.
8	Employment - The number of full-time and part-time employees on the payroll at year end.
9	Employment - Manufacturing - Consists of the employees on the payrolls of manufacturing plants of affiliates. It includes employees of manufacturing plants of affiliates that are not classified in manufacturing, and it excludes employees of nonmanufacturing plants of affiliates that are classified in manufacturing.
64	Employment of New Affiliates - The number of full-time and part-time employees on the payroll at year end.
24	Equity Inflows Other than Reinvestment of Earnings - Equity investment other than reinvestment of earnings is the net of equity increases and decreases other than those due to reinvestment of earnings. Equity increases consist of payments by parents for the purchase of capital stock or other equity interests when they acquire an

Table 22.57 *continued*

Series ID	Description
	existing business, payments made by parents to acquire additional ownership interests in their affiliates, and capital contributions to their affiliates. Equity decreases are the funds parents receive when they reduce their equity interest in their affiliates.
32	Equity Outflows Other than Reinvestment of Earnings - Equity investment other than reinvestment of earnings is the net equity increases and decreases other than those due to reinvestment of earnings. Equity increases consist of payments by parents for the purchase of capital stock or other equity interests when they acquire an existing business, payments made by parents to acquire additional ownership interests in their affiliates, and capital contributions to their affiliates. Equity decreases are the funds parents receive when they reduce their equity interest in their affiliates.
23	Financial Inflow Transactions Without Current-Cost Adjustment - Direct investment financial inflow transactions (previously called financial flows) measure the funds that foreign parent companies provide to their US affiliates, net of funds that affiliates provide to their parents, in a given time period. Direct investment financial transactions consist of equity investment other than reinvestment of earnings, and transactions in debt instruments.
31	Financial Outflow Transactions Without Current-Cost Adjustment - Direct investment financial outflow transactions (previously called financial flows) measure the funds that US parent companies provide to their foreign affiliates, net of funds that affiliates provide to their parents, in a given time period. Direct investment financial transactions consist of equity investment other than reinvestment of earnings, reinvestment of earnings, and transactions in debt instruments.
22	Foreign Direct Investment Position in the United States on a Historical-Cost Basis - The foreign direct investment position in the United States is the value of foreign direct investors' equity in their US affiliates plus the value of net outstanding loans to their US affiliates. It may be viewed as the direct investors' net financial claims on their affiliates.
58	Goods Supplied - For most industries, goods supplied are generally defined as sales of outputs that are tangible. For wholesalers and retailers, goods supplied include only the value of goods resold; BEA estimates the value of the distributive services provided by selling or arranging for the sale of goods and includes it in services supplied.
2	Gross Property, Plant, and Equipment - Property, Plant, and Equipment (PPE) consists of land, mineral rights, buildings, structures, machinery, and equipment (production, office, and transportation equipment). This measure records the value of PPE before the deduction of accumulated depreciation and depletion.
62	Investment Expenditures - Investment Expenditures
20	Liabilities - The total debts or obligations of the firm.
5	Net Income - The profits that a firm earns in a given time period. It equals total sales or gross operating revenues and other income less total expenses. It is net of, that is, after deduction of, income taxes and includes income from equity investments.

Table 22.57 *continued*

Series ID	Description
68	Net Income of New Affiliates - The profits that a firm earns in a given time period. It equals total sales or gross operating revenues and other income less total expenses. It is net of, that is, after deduction of, income taxes and includes income from equity investments.
54	Net Property, Plant, and Equipment - Property, Plant, and Equipment (PPE) consists of land, mineral rights, buildings, structures, machinery, and equipment (production, office, and transportation equipment). This measure records the value of PPE after the deduction of accumulated depreciation and depletion.
44	Other Private Services, Net Payments - Other Private Services, Net Payments represent net payments between US affiliates and their foreign parents for services provided to one another. They consist of service charges, including management fees and allocated expenses and rentals for tangible property.
35	Other Private Services, US Parents' Receipts - Other Private Services, US Parents' Receipts - Other Private Services, US Parents' Receipts represent receipts by US parents from their foreign affiliates for services provided. They consist of service charges, including management fees and allocated expenses and rentals for tangible property.
29	Other Private Services, US Affiliates' Payments - Other Private Services, US Affiliates' Payments represent payments by US affiliates to their foreign parents for services provided. They consist of service charges, including management fees and allocated expenses and rentals for tangible property.
42	Other Private Services, US Affiliates' Receipts - Other Private Services, US Affiliates' Receipts represent receipts by US affiliates from their foreign parents for services provided. They consist of service charges, including management fees and allocated expenses and rentals for tangible property.
40	Other Private Services, US Parents' Payments - Other Private Services, US Parents' Payments represent payments by US parents to their foreign affiliates for services provided. They consist of service charges, including management fees and allocated expenses and rentals for tangible property.
21	Owners Equity - Total assets of the firm less total liabilities.
53	Property, Plant, and Equipment Expenditures - The expenditures made by a firm to acquire, to add to, or to improve property, plant, and equipment (PPE). PPE includes land, timber, mineral and like-rights owned; structures, machinery, equipment, special tools, and other depreciable property; construction in progress; and tangible and intangible exploration and development costs. It excludes the changes in PPE that are due to changes in the entity—such as mergers, acquisitions, and divestitures—or that are due to changes in accounting methods. The expenditures are measured on a gross basis; sales and other dispositions of fixed assets are not netted against them.
25	Reinvestment of Earnings Without Current-Cost Adjustment - Reinvestment of earnings (a component of direct investment financial transactions) are the parents' claim on the current period undistributed earnings of their affiliates.
14	Research and Development Expenditures - Includes expenditures for the following: the planned, systematic pursuit of new knowledge or understanding toward general application (basic research); the acquisition of knowledge or understanding to meet a specific, recognized need (applied research); and the application of knowledge or understanding

Table 22.57 *continued*

Series ID	Description
	toward the production or improvement of a product, service, process, or method (development). It excludes quality control, routine product testing, market research, sales promotion, sales service and other nontechnical activities; routine technical services; geological and geophysical exploration activities, and advertising programs to promote or demonstrate new products or processes. Also excluded are capital expenditures, expenditures for tests and evaluations once a prototype becomes a production model, patent expenses, and income taxes and interest.
43	Royalties and License Fees Net of Withholding Taxes, Net Payments - Royalties and License Fees Net of Withholding Taxes, Net Payments represent receipts and payments between US affiliates and their foreign parents for the use or sale of intangible property or rights (including patents, trademarks, and copyrights).
28	Royalties and License Fees Net of Withholding Taxes, US Affiliates' Payments - Royalties and License Fees Net of Withholding Taxes, US Affiliates' Payments represent payments by US affiliates from their foreign parents for the use or sale of intangible property or rights (including patents, trademarks, and copyrights).
41	Royalties and License Fees Net of Withholding Taxes, US Affiliates' Receipts - Royalties and License Fees Net of Withholding Taxes, US Affiliates' Receipts represent receipts by US affiliates from their foreign parents for the use or sale of intangible property or rights (including patents, trademarks, and copyrights).
39	Royalties and License Fees Net of Withholding Taxes, US Parents' Payments - Royalties and License Fees Net of Withholding Taxes, US Parents' Payments represent payments by US parents from their foreign affiliates for the use or sale of intangible property or rights (including patents, trademarks, and copyrights).
34	Royalties and License Fees Net of Withholding Taxes, US Parents' Receipts - Royalties and License Fees Net of Withholding Taxes, US Parents' Receipts represent receipts of US parents from their foreign affiliates for the use or sale of intangible property or rights (including patents, trademarks, and copyrights).
56	Sales of Goods - For purposes of disaggregating sales into goods, services, and investment income, sales of goods are defined as sales that are typical of establishments in any of the following six NAICS sectors: Agriculture, forestry, fishing and hunting (except support activities for agriculture and forestry); mining (except support activities); construction; manufacturing; wholesale trade; and retail trade. (A parent or affiliate need not be classified in any of these sectors to have sales of goods.) Sales of services are defined as sales that are typical of establishments in the 14 other NAICS sectors and the support activities for agriculture, forestry, fishing and hunting and for mining. (A parent or affiliate need not be classified in one of these sectors to have sales of services.)
55	Sales of Services - For purposes of disaggregating sales into goods, services, and investment income, sales of goods are defined as sales that are typical of establishments in any of the following six NAICS sectors: Agriculture, forestry, fishing and hunting (except support activities for agriculture and forestry); mining (except support activities); construction; manufacturing; wholesale trade; and retail trade. (A parent or affiliate need not be classified in any of these sectors to have sales of goods.) Sales of

Table 22.57 *continued*

Series ID	Description
	services are defined as sales that are typical of establishments in the 14 other NAICS sectors and the support activities for agriculture, forestry, fishing and hunting and for mining. (A parent or affiliate need not be classified in one of these sectors to have sales of services.)
60	Services Supplied - For most industries, services supplied generally are defined as sales of outputs that are intangible. For insurance, services supplied consists of BEA's estimate of the portion of premiums remaining after provision for expected or normal losses and a measure of premium supplements, which represent income earned on funds insurers hold on policyholders' behalf. For banks, it includes explicit fees and commissions and an estimate of the value of implicit services provided by banks. For wholesalers and retailers, services supplied includes an estimate of the distributive services provided by selling or arranging for the sales of goods.
61	Services Supplied to Foreigners - Services Supplied to Foreigners - Sales of outputs that are intangible to foreign persons. Sales may be to other foreign affiliates or unaffiliated foreigners, and the foreign persons may be local or in another foreign country.
1	Total Assets - Any owned physical object or intangible right having economic value to a firm. It includes items such as cash, inventories, receivables, and property, plant, and equipment. It also includes equity investments in unconsolidated domestic and foreign businesses.
52	Total Assets - Affiliates - Any owned physical object or intangible right
51	Total Assets - US Parent - Any owned physical object or intangible right
4	Total Sales - The value of goods and services sold and, for financial firms, also includes investment income. It is net of returns, allowances, and discounts and excludes sale or consumption taxes levied directly on the consumer and excise taxes levied on manufacturers, wholesalers, and retailers.
66	Total Sales of New Affiliates - The value of goods and services sold and, for financial firms, also includes investment income. It is net of returns, allowances, and discounts and excludes sale or consumption taxes levied directly on the consumer and excise taxes levied on manufacturers, wholesalers, and retailers.
30	US Direct Investment Position Abroad on a Historical-Cost Basis - The US direct investment position abroad is the value of US direct investors' equity in their foreign affiliates plus the value of net outstanding loans to their affiliates. It may be viewed as the direct investors' net financial claims on their affiliates
10	US Exports of Goods - Exports are valued on an f.a.s. (free alongside ship) basis; they exclude transit costs, such as the costs of shipping and insurance.
57	US Exports of Goods Shipped by US Parents to Foreign Affiliates - Exports are valued on an f.a.s. (free alongside ship) basis; they exclude transit costs, such as the costs of shipping and insurance.
46	US Exports of Goods Shipped by US Parents to Foreigners Other Than Foreign Affiliates - Exports are valued on an f.a.s. (free alongside ship) basis; they exclude transit costs, such as the costs of shipping and insurance.
11	US Exports of Goods to the Foreign Parent Group - US Exports of Goods to the Foreign Parent Group represents total exports of goods from US affiliates to their foreign parent groups
12	US Imports of Goods - Imports are valued on an f.a.s. (free alongside ship) basis; they exclude

Table 22.57 *continued*

Series ID	Description
	transit costs, such as the costs of shipping and insurance.
13	US Imports of Goods from the Foreign Parent Group - US Imports of Goods from the Foreign Parent Group represents total imports of goods by US parent enterprises from their foreign affiliates
47	US Imports of Goods Shipped to US Parents by Foreign Affiliates - Imports are valued on an f.a.s. (free alongside ship) basis; they exclude transit costs, such as the costs of shipping and insurance.
48	US Imports of Goods Shipped to US Parents by Foreigners Other Than Foreign Affiliates - Imports are valued on an f.a.s. (free alongside ship) basis; they exclude transit costs, such as the costs of shipping and insurance.
50	US Trade in Goods of Affiliates; US Trade in Goods of Affiliates represents US exports and imports of goods associated with a US affiliate or foreign affiliate
19	US Trade in Goods of Majority-Owned Affiliates - US Trade in Goods of Majority-Owned Affiliates represents US exports and imports of goods associated with a majority-owned US affiliate or majority-owned foreign affiliate
6	Value Added (Gross Product) - The portion of the goods and services sold or added to inventory or fixed investment by a firm that reflects the production of the firm itself. It represents the firm's contribution to a country's gross domestic product, which is the value of goods and services produced by labor and property located in the country. Compared to sales, value added is a preferable measure of production because it indicates the extent to which a firm's sales result from their own production rather than from production that originates elsewhere, whereas sales data do not distinguish between these two sources of production. Value added can be measured as gross output minus intermediate inputs; alternatively, it can be measured as the sum of the costs incurred (except for intermediate inputs) and the profits earned in production. The value-added estimates presented here were prepared by summing the costs and profits data.

The classification names available for the DirectInvestment (DI) Multinational Enterprise (MNE) data set and the Activities of Multinational Enterprise (AMNE) data set are listed in [Table 22.58](#). Each classification has a unique name.

Table 22.58 MNE Classifications and their Descriptions

Classification Name	Description
Country	Set the classification to country to obtain data for all countries for which data are available for the given series, ownership, and year
CountryByDestination	Country by destination returns values for country broken out by destination for goods supplied and services supplied.
CountryByIndustry	Country by industry provides data for a large set of countries broken down by major industry.
CountrybyType	Country by type of investment returns data on new

Table 22.58 *continued*

Classification Name	Description
	foreign direct investment in the United States broken out by country of ultimate beneficial owner (UBO) and the type of investment (acquisition, establishment, or expansion).
CountryofUBO	Presents data on foreign direct investment in the United States by the country of the ultimate beneficial owner (UBO). The ultimate beneficial owner (UBO) is the entity, proceeding up the foreign ownership chain, which is not more than 50 percent owned by another entity. The UBO is the entity that ultimately owns or controls an affiliate and thus ultimately derives the benefits and assumes the risks from owning or controlling an affiliate.
CountryofUBObyDestination	Presents data on services supplied by US affiliates of foreign MNEs, broken down by the country of ultimate beneficial owner (UBO) and the destination (US or foreign) of the service supplied
CountryofUBObyIndustry	Presents data on foreign direct investment in the United States broken down by the country of the ultimate beneficial owner (UBO) and the industry of the affiliate.
CountryofUBObyType	Country of UBO by type of investment returns data on new foreign direct investment in the United States broken out by the country of the ultimate beneficial owner (UBO) and the type of investment (acquisition, establishment, or expansion).
Industry	Set the classification equal to 'industry' to obtain data for all industries for which data are available for the given series, ownership, and year.
IndustryByCountry	Industry by country provides data for a large set of industries broken down by major country.
IndustrybyCountryofUBO	Presents foreign direct investment data broken down by industry of the US affiliate and the country of ultimate beneficial owner (UBO)
IndustryByDestination	Industry by destination returns values for industry broken out by destination for goods supplied and services supplied.
IndustrybyType	Industry by type of investment returns data on new foreign direct investment in the United States broken out by industry of US business and the type of investment (acquisition, establishment, or expansion).
IndustryofSales	Industry of sales provides data on employment and sales for majority-owned US affiliates of foreign MNEs. Each majority-owned US affiliate above a minimum size threshold was required to distribute its sales and its employment among the four-digit industry in which it had sales. These data approximate an establishment-based distribution.
IndustryofUSParent	Industry of US parent provides data for foreign affiliates of US

Table 22.58 *continued*

Classification Name	Description
IndustryofUSParentAllIndustries	parents, broken down by the industry of the US parent, which may differ from the industry of the affiliate. Industry of US parent, all industries, provides data for foreign affiliates of US parents, broken down by the most detailed industry of the US parent, which may differ from the industry of the affiliate.
IndustryofUSParentByCountry	Industry of US parent by country provides data for foreign affiliates of US parents, broken down by the industry of the US parent, which may differ from the industry of the affiliate, and broken and broken by major country.
StatebyCountryofUBO	Presents selected activities of multinational enterprises data on US affiliates, broken down by US state and country of UBO.
StatebyType	State by type of investment returns data on new foreign direct investment in the United States broken out by destination state and the type of investment (acquisition, establishment, or expansion).
TypeofExpenditure	Type of expenditure returns values on investment expenditure for new foreign direct investment in the United States, broken out by type of expenditure.

The geographic area codes available for the DirectInvestment (DI) Multinational Enterprise (MNE) data set and the Activities of Multinational Enterprise (AMNE) data set are listed in [Table 22.59](#). Each geographic area (or country) has a unique code.

Table 22.59 MNE Geographic Area Codes and Descriptions

Geographic Area Code	Country Name	Geographic Area Code	Country Name
all	All	316	Luxembourg
000	All Countries Total	617	Malaysia
499	Africa	213	Mexico
699	Asia and Pacific	599	Middle East
601	Australia	319	Netherlands
898	Australia, New Zealand, and South Africa	259	Netherlands Antilles
301	Austria	620	New Zealand
250	Bahamas	320	Norway
302	Belgium	899	OPEC
252	Bermuda	899	Other Africa, Asia, and Pacific
202	Brazil	394	Other EC
100	Canada	298	Other Western Hemisphere
650	China	215	Panama
275	Curacao	623	Philippines
305	Denmark	511	Saudi Arabia
390	Denmark, Greece, and Ireland	625	Singapore
399	Europe	436	South Africa
398	European Union	296	South and Central America
306	Finland	323	Spain
307	France	324	Sweden
308	Germany	325	Switzerland
611	Hong Kong	628	Taiwan
612	India	000	Total 1
999	International	000	Total 2
313	Ireland	000	Total 3
504	Israel	000	Total 4
314	Italy	699	Total Asia and Pacific
614	Japan	399	Total Europe
626	Korea, Republic of	299	Total LA and OWH
506	Kuwait	513	United Arab Emirates
299	Latin America and Other Western Hemisphere	327	United Kingdom
299	Latin America and Other Western Hemisphere2	266	United Kingdom Islands, Caribbean
507	Lebanon	199	United States
315	Liechtenstein	219	Venezuela

The industry codes available for the DirectInvestment (DI) Multinational Enterprise (MNE) data set and the Activities of Multinational Enterprise (AMNE) data set are listed in [Table 22.60](#). Each industry has a unique code.

Table 22.60 MNE Industry Codes and Descriptions

Industry Code	Description or Industry Name
all	All
0000	All Industries Total
7210	Accommodation
7200	Accommodation and food services
5412	Accounting, tax preparation, bookkeeping, and payroll services
5223	Activities related to credit intermediation
5600	Administration, support, and waste management
5600	Administrative and support services
5418	Advertising and related services
3364	Aerospace products and parts
5242	Agencies, brokerages, and other insurance-related activities
3331	Agriculture, construction, and mining machinery
1100	Agriculture, forestry, fishing, and hunting
4810	Air transportation
3313	Alumina and aluminum production and processing
6210	Ambulatory health care services
7130	Amusement, gambling, and recreation industries
3111	Animal foods
1120	Animal production
3116	Animal slaughtering and processing
3150	Apparel
4243	Apparel, piece goods, and notions
3323	Architectural and structural metals
5413	Architectural, engineering, and related services
7100	Arts, entertainment, and recreation
3244	Asphalt and other petroleum and coal products
3343	Audio and video equipment
5321	Automotive equipment rental and leasing
3118	Bakeries and tortillas
5221	Banks
3251	Basic chemicals
4248	Beer, wine, and distilled alcoholic beverages
3121	Beverages
3120	Beverages and tobacco products
3324	Boilers, tanks, and shipping containers
5229	Branches and agencies
5150	Broadcasting (except internet)
5149	Broadcasting (except internet) and telecommunications
5148	Broadcasting and telecommunications
5147	Broadcasting, cable networks, and program distribution
4440	Building material and garden equipment and supplies dealers
2330	Building, developing, and general contracting
5614	Business support services
5175	Cable and other program distribution

Table 22.60 *continued*

Industry Code	Description or Industry Name
5152	Cable and other subscription programming
5176	Cable networks and program distribution
3273	Cement and concrete products
4246	Chemical and allied products
3250	Chemicals
3271	Clay products and refractories
4480	Clothing and clothing accessories stores
2121	Coal mining
3328	Coating, engraving, heat treating, and allied activities
3333	Commercial and service industry machinery
3342	Communications equipment
5415	Computer systems design and related services
3340	Computers and electronic products
3341	Computers and peripheral equipment
2300	Construction
2360	Construction of buildings
3222	Converted paper products
2126	Copper, nickel, lead, and zinc ores
5513	Corporate, subsidiary, and regional management offices
4920	Couriers and messengers
1110	Crop production
3322	Cutlery and handtools
3115	Dairy products
5142	Data processing services
5182	Data processing, hosting, and related services
5221	Depository credit intermediation (banking)
5225	Depository institutions
5221	Depository institutions (banking)
4242	Drugs and druggists' sundries
6110	Educational services
3351	Electric lighting equipment
2211	Electric power generation, transmission, and distribution
3353	Electrical equipment
3350	Electrical equipment, appliances, and components
4236	Electrical goods
4431	Electronics and appliance stores
5613	Employment services
3336	Engines, turbines, and power transmission equipment
3320	Fabricated metal products
5612	Facilities support services
4245	Farm product raw materials
5220	Finance (except depository institutions) and insurance
5200	Finance and insurance
5220	Finance, except depository institutions

Table 22.60 *continued*

Industry Code	Description or Industry Name
1140	Fishing, hunting, and trapping
3110	Food
4450	Food and beverage stores
7220	Food services and drinking places
1130	Forestry and logging
3321	Forging and stamping
3315	Foundries
3114	Fruit and vegetable preserving and specialty foods
5252	Funds, trusts, and other financial vehicles
4232	Furniture and home furnishings
4420	Furniture and home furnishings stores
3370	Furniture and related products
4471	Gasoline stations
4520	General merchandise stores
3272	Glass and glass products
2125	Gold and silver ores
3112	Grain and oilseed milling
4244	Groceries and related products
3325	Hardware
4237	Hardware, and plumbing and heating equipment and supplies
4461	Health and personal care stores
6200	Health care and social assistance
2370	Heavy and civil engineering construction
2340	Heavy construction
5512	Holding companies (nonbank)
5513	Holding companies (nonbank), excluding management offices
5512	Holding companies, except bank holding companies
6220	Hospitals
3352	Household appliances
3332	Industrial machinery
5100	Information
5141	Information services
5188	Information services and data processing services
5240	Insurance carriers and related activities
5243	Insurance carriers, except life insurance carriers
3242	Integrated petroleum refining and extraction
5161	Internet publishing and broadcasting
5181	Internet service providers and web search portals
5180	Internet service providers, web search portals, and data processing services
5189	Internet, data processing, and other information services
5616	Investigation and security services
3311	Iron and steel mills and ferroalloys
2124	Iron ores
3160	Leather and allied products

Table 22.60 *continued*

Industry Code	Description or Industry Name
5411	Legal services
5331	Lessors of nonfinancial intangible assets (except copyrighted works)
5249	Life insurance carriers
3274	Lime and gypsum products
4233	Lumber and other construction materials
3327	Machine shops, turned products, and screws, nuts, and bolts
3330	Machinery
4238	Machinery, equipment, and supplies
4238	Machinery, equipment, and supplies merchant wholesalers
3346	Magnetic and optical media
5512	Management of nonbank companies and enterprises
5416	Management, scientific, and technical consulting
3000	Manufacturing
3116	Meat products
3391	Medical equipment and supplies
2128	Metal ore mining
4235	Metals and minerals (except petroleum)
3335	Metalworking machinery
2100	Mining
2199	Mining (except oil and gas)
4239	Miscellaneous durable goods
3390	Miscellaneous manufacturing
4249	Miscellaneous nondurable goods
6000	Miscellaneous services
4530	Miscellaneous store retailers
5120	Motion picture and sound recording industries
5121	Motion picture and video industries
4410	Motor vehicle and parts dealers
3362	Motor vehicle bodies and trailers
3363	Motor vehicle parts
3361	Motor vehicles
4231	Motor vehicles and motor vehicle parts and supplies
3368	Motor vehicles, bodies and trailers, and parts
7121	Museums, historical sites, and similar institutions
2212	Natural gas distribution
3345	Navigational, measuring, and other instruments
5111	Newspaper, periodical, book, and database publishers
5224	Non-depository credit intermediation
9100	Nonbusiness entities, except government
5224	Nondepository credit intermediation
5224	Nondepository credit intermediation and related services
3314	Nonferrous metal (except aluminum) production and processing
2123	Nonmetallic mineral mining and quarrying
3270	Nonmetallic mineral products

Table 22.60 *continued*

Industry Code	Description or Industry Name
4540	Nonstore retailers
6230	Nursing and residential care facilities
5611	Office administrative services
2111	Oil and gas extraction
3259	Other chemical products and preparations
4299	Other durable goods
3359	Other electrical equipment and components
3329	Other fabricated metal products
5299	Other finance, except depository institutions
5238	Other financial investment activities and exchanges
3119	Other food products
3339	Other general purpose machinery
9999	Other industries
5191	Other information services
3999	Other manufacturing
2127	Other metal ores
3399	Other miscellaneous manufacturing
4298	Other nondurable goods
3279	Other nonmetallic mineral products
4868	Other pipeline transportation
5419	Other professional, scientific, and technical services
5329	Other rental and leasing services
8100	Other services (except public administration and private households)
5619	Other support services
5179	Other telecommunications
3369	Other transportation equipment
4939	Other warehousing and storage
4839	Other water transportation
3255	Paints, coatings, and adhesives
3220	Paper
4241	Paper and paper products
7110	Performing arts, spectator sports, and related industries
8120	Personal and laundry services
3253	Pesticides, fertilizers, and other agricultural chemicals
8999	Petroleum
3240	Petroleum and coal products
4247	Petroleum and petroleum products
3243	Petroleum refining, excluding oil and gas extraction
4932	Petroleum storage for hire
4833	Petroleum tanker operations
3254	Pharmaceuticals and medicines
4860	Pipeline transportation
4863	Pipeline transportation of crude oil, refined petroleum products, and natural gas
3260	Plastics and rubber products

Table 22.60 *continued*

Industry Code	Description or Industry Name
3261	Plastics products
3310	Primary and fabricated metals
3310	Primary metals
3231	Printing and related support activities
4214	Professional and commercial equipment and supplies
5400	Professional, scientific, and technical services
5110	Publishing industries
3221	Pulp, paper, and paperboard mills
5151	Radio and television broadcasting
4821	Rail transportation
3365	Railroad rolling stock
5310	Real estate
5300	Real estate and rental and leasing
8130	Religious, grantmaking, civic, professional, and similar organizations
5320	Rental and leasing (except real estate)
8110	Repair and maintenance
3252	Resins and synthetic rubber, fibers, and filaments
4400	Retail trade
3262	Rubber products
5174	Satellite telecommunications
4870	Scenic and sightseeing transportation
5417	Scientific research and development services
3117	Seafood product preparation and packaging
5231	Securities and commodity contracts intermediation and brokerage
5230	Securities, commodity contracts, and other intermediation and related activities
3344	Semiconductors and other electronic components
5000	Services
5617	Services to buildings and dwellings
3366	Ship and boat building
3256	Soap, cleaning compounds, and toilet preparations
6240	Social assistance
5112	Software publishers
5122	Sound recording industries
2350	Special trade contractors
5414	Specialized design services
4510	Sporting goods, hobby, book, and music stores
3326	Spring and wire products
3312	Steel products from purchased steel
3113	Sugar and confectionery products
1150	Support activities for agriculture and forestry
2130	Support activities for mining
2133	Support activities for mining, except for oil and gas extraction
2132	Support activities for oil and gas extraction
4880	Support activities for transportation

Table 22.60 *continued*

Industry Code	Description or Industry Name
5133	Telecommunications
5173	Telecommunications resellers
3130	Textile mills
3140	Textile product mills
3170	Textiles, apparel, and leather products
3122	Tobacco products
3000	Total manufacturing
4850	Transit and ground passenger transportation
4800	Transportation and warehousing
3360	Transportation equipment
5615	Travel arrangement and reservation services
4840	Truck transportation
2200	Utilities
3334	Ventilation, heating, air conditioning, and commercial refrigeration equipment
4930	Warehousing and storage facilities
5620	Waste management and remediation services
4830	Water transportation
2213	Water, sewage, and other systems
4251	Wholesale electronic markets and agents and brokers
4200	Wholesale trade
5171	Wired telecommunications carriers
5172	Wireless telecommunications carriers (except satellite)
3210	Wood products

The state codes available for the Activities of Multinational Enterprise (AMNE) data set are listed in [Table 22.61](#). Each state (or area) has a unique code.

Table 22.61 MNE State Codes and Descriptions

State Code	Description or State Name
all	All
00	Total states and areas
01	Alabama
02	Alaska
04	Arizona
05	Arkansas
06	California
08	Colorado
09	Connecticut
10	Delaware
11	District of Columbia
12	Florida
75	Foreign
13	Georgi

Table 22.61 *continued*

State Code	Description or State Name
15	Hawaii
16	Idaho
17	Illinois
18	Indiana
19	Iowa
20	Kansas
21	Kentucky
22	Louisiana
23	Maine
24	Maryland
25	Massachusetts
26	Michigan
27	Minnesota
28	Mississippi
29	Missouri
30	Montana
31	Nebraska
32	Nevada
33	New Hampshire
34	New Jersey
35	New Mexico
36	New York
37	North Carolina
38	North Dakota
39	Ohio
40	Oklahoma
41	Oregon
70	Other US areas
42	Pennsylvania
43	Puerto Rico
44	Rhode Island
45	South Carolina
46	South Dakota
47	Tennessee
48	Texas
9000	Unspecified
99	Unspecified state or area
49	Utah
50	Vermont
51	Virginia
53	Washington
54	West Virginia
55	Wisconsin
56	Wyoming

Regional Data

The table names available for the Regional data set are listed in [Table 22.62](#). Each table is identified by a unique table name.

Table 22.62 Regional Table Names and Descriptions

TableName	Description
CAEMP25N	Total Full-Time and Part-Time Employment by NAICS Industry (NAICS)
CAEMP25S	Total Full-Time and Part-Time Employment by SIC Industry (SIC)
CAGDP1	Gross Domestic Product (GDP) summary by county and metropolitan area (Non-Industry)
CAGDP11	Contributions to percent change in real GDP (NAICS)
CAGDP2	Gross domestic product (GDP) by county and metropolitan area (NAICS)
CAGDP8	Chain-type quantity indexes for real GDP by county and metropolitan area (2012=100.0) (NAICS)
CAGDP9	Real GDP by county and metropolitan area (NAICS)
CAINC1	Personal Income Summary: Personal Income, Population, Per Capita Personal Income (Non-Industry)
CAINC30	Economic Profile (Non-Industry)
CAINC35	Personal Current Transfer Receipts (Non-Industry)
CAINC4	Personal Income and Employment by Major Component (Non-Industry)
CAINC45	Farm Income and Expenses (Non-Industry)
CAINC5N	Personal Income by Major Component and Earnings by NAICS Industry (NAICS)
CAINC5S	Personal Income by Major Component and Earnings by SIC Industry (SIC)
CAINC6N	Compensation of Employees by NAICS Industry (NAICS)
CAINC6S	Compensation of Employees by SIC Industry (SIC)
CAINC91	Gross Flow of Earnings (Non-Industry)
MAIRPD	Implicit Regional Price Deflators by MSA (Non-Industry)
MARPI	Real Personal Income by MSA (Non-Industry)
MARPP	Regional Price Parities by MSA (Non-Industry)
PAIRPD	Implicit Regional Price Deflators by Portion (Non-Industry)
PARPI	Real Personal Income by Portion (Non-Industry)
PARPP	Regional Price Parities by Portion (Non-Industry)
SAACArtsComp	ACPSA compensation by ACPSA industry (NAICS)
SAACArtsEmp	ACPSA full-time and part-time employment by ACPSA industry (NAICS)
SAACArtsVA	ACPSA value added by ACPSA industry (NAICS)
SAACCompLQ	Arts Compensation Location Quotient (NAICS)
SAACCompRatio	Arts Compensation Ratio (NAICS)
SAACEmpLQ	Arts Employment Location Quotient (NAICS)
SAACEmpRatio	Arts Employment Ratio (NAICS)
SAACVALQ	Arts Value Added Location Quotient (NAICS)
SAACVARatio	Arts Value Added Ratio (NAICS)
SAEMP25N	Total Full-Time and Part-Time Employment by NAICS Industry (NAICS)
SAEMP25S	Total Full-Time and Part-Time Employment by SIC Industry (SIC)
SAEMP27N	Full-Time and Part-Time Wage and Salary Employment by NAICS Industry (NAICS)
SAEMP27S	Full-Time and Part-Time Wage and Salary Employment by SIC Industry (SIC)

Table 22.62 *continued*

TableName	Description
SAEXP1	Total personal consumption expenditures (PCE) by state (NAICS)
SAEXP2	Per capita personal consumption expenditures (PCE) by state (NAICS)
SAGDP1	Gross Domestic Product (GDP) summary, annual by state (Non-Industry)
SAGDP11N	Contributions to percent change in real GDP (NAICS)
SAGDP2N	Gross domestic product (GDP) by state (NAICS)
SAGDP2S	Gross domestic product (GDP) by state (SIC)
SAGDP3N	Taxes on production and imports less subsidies (NAICS)
SAGDP3S	Taxes on production and imports less subsidies (SIC)
SAGDP4N	Compensation of employees (NAICS)
SAGDP4S	Compensation of employees (SIC)
SAGDP5N	Subsidies (NAICS)
SAGDP5S	Subsidies (SIC)
SAGDP6N	Taxes on production and imports (NAICS)
SAGDP6S	Taxes on production and imports (SIC)
SAGDP7N	Gross operating surplus (NAICS)
SAGDP7S	Gross operating surplus (SIC)
SAGDP8N	Chain-type quantity indexes for real GDP by state (2012=100.0) (NAICS)
SAGDP8S	Chain-type quantity indexes for real GDP by state (1997=100.0) (SIC)
SAGDP9N	Real GDP by state (NAICS)
SAGDP9S	Real GDP by state (SIC)
SAINC1	Personal Income Summary: Personal Income, Population, Per Capita Personal Income (Non-Industry)
SAINC30	Economic Profile (Non-Industry)
SAINC35	Personal Current Transfer Receipts (Non-Industry)
SAINC4	Personal Income and Employment by Major Component (Non-Industry)
SAINC40	Property Income (Non-Industry)
SAINC45	Farm Income and Expenses (Non-Industry)
SAINC50	Personal Current Taxes (Non-Industry)
SAINC51	Disposable Personal Income Summary: Disposable Personal Income, Population, and Per Capita Personal Income (Non-Industry)
SAINC5H	Personal Income by Major Component and Earnings by Industry (Historical) (SIC)
SAINC5N	Personal Income by Major Component and Earnings by NAICS Industry (NAICS)
SAINC5S	Personal Income by Major Component and Earnings by SIC Industry (SIC)
SAINC6N	Compensation of Employees by NAICS Industry (NAICS)
SAINC6S	Compensation of Employees by SIC Industry (SIC)
SAINC7H	Wages and Salaries by Industry (Historical) (SIC)
SAINC7N	Wages and Salaries by NAICS Industry (NAICS)
SAINC7S	Wages and Salaries by SIC Industry (SIC)
SAINC91	Gross Flow of Earnings (Non-Industry)
SAIRPD	Implicit Regional Price Deflators by state (Non-Industry)
SARPI	Real Personal Income by state (Non-Industry)
SARPP	Regional Price Parities by state (Non-Industry)
SQGDP1	Gross Domestic Product (GDP) summary, quarterly by state (Non-Industry)
SQGDP11	Contributions to percent change in real GDP (NAICS)

Table 22.62 *continued*

TableName	Description
SQGDP2	Gross domestic product (GDP) by state (NAICS)
SQGDP8	Chain-type quantity indexes for real GDP by state (2012=100.0) (NAICS)
SQGDP9	Real GDP by state (NAICS)
SQINC1	Personal Income Summary: Personal Income, Population, Per Capita Personal Income (Non-Industry)
SQINC35	Personal Current Transfer Receipts (Non-Industry)
SQINC4	Personal Income by Major Component (Non-Industry)
SQINC5H	Personal Income by Major Component and Earnings by Industry (Historical) (SIC)
SQINC5N	Personal Income by Major Component and Earnings by NAICS Industry (NAICS)
SQINC5S	Personal Income by Major Component and Earnings by SIC Industry (SIC)
SQINC6N	Compensation of Employees by NAICS Industry (NAICS)
SQINC6S	Compensation of Employees by SIC Industry (SIC)
SQINC7H	Wages and Salaries by Industry (Historical) (SIC)
SQINC7N	Wages and Salaries by NAICS Industry (NAICS)
SQINC7S	Wages and Salaries by SIC Industry (SIC)

Two older data sets, `RegionalIncome` and `RegionalProduct`, are now deprecated by the BEA. The SASEBEA engine does not support the deprecated data sets. However, for compatibility, [Table 22.63](#) and [Table 22.64](#) show the crosswalk that gives the correct table name for the new Regional data set that replaces either the old component parameter in the `RegionalProduct` data set or the old table name parameter in the `RegionalIncome` data set.

Table 22.63 New Table Name to Old Component Parameter

New Table Name	Old Component Parameter
SQGDP2	GDP_SQN
SQGDP9	RGDP_SQN
SQGDP8	QI_SQN
SQGDP11	CTG_SQN
SAGDP2N	GDP_SAN
SAGDP2S	GDP_SAS
SAGDP4N	COMP_SAN
SAGDP4S	COMP_SAS
SAGDP3N	TOPILS_SAN
SAGDP3S	TOPILS_SAS
SAGDP7N	GOS_SAN
SAGDP7S	GOS_SAS
SAGDP9N	RGDP_SAN
SAGDP9S	RGDP_SAS
SAGDP8N	QI_SAN
SAGDP8S	QI_SAS
SAGDP5N	SUBS_SAN
SAGDP5S	SUBS_SAS

Table 22.63 *continued*

New Table Name	Old Component Parameter
SAGDP6N	TOPI_SAN
SAGDP6S	TOPI_SAS
SAGDP10N	PCRGDP_SAN
SAGDP10S	PCRGDP_SAS
SAGDP11N	CTG_SAN
CAGDP2	GDP_MAN
CAGDP9	RGDP_MAN
CAGDP11	CTG_MAN
SAEXP1	PCE_SAN
SAEXP2	PCPCE_SAN

Table 22.64 New Table Name to Old Table Parameter

New Table Name	Old Table Parameter
SQINC1	SQ1
SQINC4	SQ4
SQINC5N	SQ5N
SQINC5S	SQ5
SQINC5H	SQ5H
SQINC6N	SQ6N
SQINC6S	SQ6
SQINC7N	SQ7N
SQINC7S	SQ7
SQINC7H	SQ7H
SQINC35	SQ35
SAINC1	SA1
SAINC51	SA51
SAINC4	SA4
SAINC5N	SA5N
SAINC5S	SA5
SAINC5H	SA5H
SAINC6N	SA6N
SAINC6S	SA6
SAINC7N	SA7N
SAINC7S	SA7
SAINC7H	SA7H
SAEMP25N	SA25N
SAEMP25S	SA25
SAEMP27N	SA27N
SAEMP27S	SA27
SAINC30	SA30
SAINC35	SA35
SAINC40	SA40

Table 22.64 *continued*

New Table Name	Old Table Parameter
SAINC45	SA45
SAINC50	SA50
CAINC1	CA1 or MAINC1
CAINC4	CA4 or MAINC4
CAINC5N	CA5N
CAINC5S	CA5
CAINC6N	CA6N
CAINC6S	CA6
CAEMP25N	CA25N
CAEMP25S	CA25
CAINC30	CA30 or MAINC30
CAINC35	CA35
CAINC45	CA45
CAINC91	CA91
SARPI	RPI1
MARPI	RPI2
PARPI	
SARPP	RPP1
MARPP	RPP2
PARPP	
SAIRPD	IRPD1
MAIRPD	IRPD2
PAIRPD	

There are also changes in the units of measurement for the new Regional tables. For more information about the new Regional data set changes to the units, see the section “Change in Units” at the following web page:

<https://apps.bea.gov/regional/docs/RegionalApi.cfm>

SAS Output Data Set

You can use a SAS DATA step to write the selected BEA time series data to a SAS data set. This enables you to use SAS software to easily analyze the data. If you specify the name of the output data set in the DATA statement, the SAS engine supervisor creates a SAS data set that has the specified name in the location specified by the SASEBEA engine libref’s *physical-name*.

The contents of the SAS data set include the date of each observation and the name of each time series that is read from the BEA website. Refer to the details of each BEA data set to find that data set’s naming convention for time series.

You can use the PRINT and CONTENTS procedures to print your output data set and its contents. Alternatively, you can view your SAS output observations by opening the desired output data set in a SAS Explorer window. You can also use the SQL procedure with your SASEBEA engine libref to create a custom view of your data.

Because each SASEBEA engine libref results in retrieving the requested data from the BEA's economic statistics data site, it is best to use a DATA step to store the data. You should avoid the inefficient use of the SASEBEA engine libref that follows:

```
proc print data=BEANIPA.NIPGDP;
```

Here the member name, NIPGDP, which is used in the PROC PRINT statement, invokes the BEANIPA libref to run the SASEBEA engine. It is more efficient to refer to the SAS data set myGDP repeatedly than to invoke the interface engine repeatedly. This use of the member name, NIPGDP, corresponds to specifying the OUT=NIPGDP option. Although using this statement might seem easier, it is much less efficient, because every time you use the SASEBEA engine libref, the SASEBEA engine reads the entire JSON file into SAS again.

Series Names for BEA SETNAME=NIPA and SETNAME=NIPAUNDERLYINGDETAIL

For the NIPA tables, the series names can be listed by using the CONTENTS procedure on the SAS output data set. The available table names are listed in [Table 22.14](#) and [Table 22.15](#). For the NIPA Underlying Detail tables, the series names can be listed by using the CONTENTS procedure on the SAS output data set. The table names are listed in [Table 22.16](#).

Series Names for BEA SETNAME=GDPBYINDUSTRY and SETNAME=UNDERLYINGGDPBYINDUSTRY

For the Gross Domestic Product data, the series names can be listed by using the CONTENTS procedure on the SAS output data set. Time series names begin with an underscore, followed by the industry code as a prefix, which is then joined to the description using an underscore ('_') and truncated to 24 bytes. Then the table ID is appended to the 24-byte name. The three parts of the time series names are joined together using underscores ('_'). The available table names for the GDPbyIndustry data are listed in [Table 22.17](#). The available table names for the UnderlyingGDPbyIndustry data are listed in [Table 22.21](#).

Series Names for BEA SETNAME=FIXEDASSETS

For the FixedAssets data, the series names can be listed by using the CONTENTS procedure on the SAS output data set. The SASEBEA engine forms each time series name by starting with the series code, followed by an underscore ('_'), and then the line number. The available table names for the FixedAssets data are listed in [Table 22.22](#).

Series Names for BEA SETNAME=MNE

For the Direct Investment Data and Activities of Multinational Enterprise data, the series names can be listed by using the CONTENTS procedure on the SAS output data set. Each time series name is made up of the following concatenations in this order: 'SeriesID_', seriesID, 'ROW_', row position, 'COL_', and column code. The available series IDs for the MNE data are listed in [Table 22.57](#).

Series Names for BEA SETNAME=ITA

For the International Transactions Accounts data, the series names can be listed by using the CONTENTS procedure on the SAS output data set. Each time series name is made up of the following concatenation, in this order: time series ID, area or country code (first 7 bytes or characters), ‘_’, and frequency code. For very long time series IDs, the engine uses only bytes 8–18 and the last 12 bytes as a prefix to the country code and frequency code.

You can generate a list of all series names for each indicator that is listed in [Table 22.24](#) through [Table 22.47](#). The ITA database can contain more or fewer indicators than are listed because it is updated periodically.

To generate a list of all the time series names for a particular indicator, specify one indicator by using the INDICATOR= option, and specify AREAORCOUNTRY=ALL. Then use the CONTENTS procedure on the SAS output data set to list all the series names for the specified indicator. See [Example 22.7](#) for an example.

Series Names for BEA SETNAME=IIP

For the International Investment Position data, the series names can be listed by using the CONTENTS procedure on the SAS output data set. Each time series name is made up of the following concatenation in this order: ‘TSI_Iip’, type of investment position, component, ‘_’, and frequency code. For very long time series IDs, the engine does not use the prefix ‘TSI_Iip’. Instead, the time series name is made up of the first 10 bytes of the type of investment, followed by the last 10 bytes of the type of investment, followed by ‘_’ and then the component. See [Example 22.9](#) for an example.

Series Names for BEA SETNAME=InputOutput

For the Input-Output statistics data, the series names can be listed by using the CONTENTS procedure on the SAS output data set. Each time series name is made up of the following concatenation, in this order: ‘Tab’, table ID, ‘_ROW_’, commodity code, ‘_COL_’, and industry code. See [Example 22.10](#) for an example.

Series Names for BEA SETNAME=IntIServTrade

For the international trade in services data, the series names can be listed by using the CONTENTS procedure on the SAS output data set. The engine renames the time series to ISTVAR1-ISTVARn, where n is the number of variables (number of time series). The label for each time series contains the BEA assigned time series ID. To use the longer time series description for the label, use the DESCRIPTION=LONG option. If column headings in the SAS output are preferred to match the label instead of the ISTVARn series name, use the LABEL option on the PROC PRINT statement. See [Example 22.11](#) for an example.

Series Names for BEA SETNAME=Regional

For the regional data, the series names can be listed by using the CONTENTS procedure on the SAS output data set. Each time series name is made up of the following concatenation, in this order: table name, ‘-’, line number, ‘_GeoFips’, and FIPS code. See [Example 22.12](#) for an example.

Data Elements Reference: SASEBEA Interface Engine

Table 22.3 lists the data set names and respective options for each data set. This table is not exhaustive because the BEA website is updated constantly. Visit the website for current BEA data set names, parameters, and valid parameter values. This section demonstrates how to retrieve an updated list of valid parameter values for each parameter of a specified BEA data set.

You can use the following sample code to obtain an updated list of valid data set names, as shown in Table 22.3:

```
/* First, read the data set names list using the Bureau of Economic
   Analysis API with the URL engine in the FILENAME statement */

filename trans url
  'https://apps.bea.gov/api/data/?&userid=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  &method=getdatasetlist&resultformat=xml';

filename fast 'U:\mytestfolder\fast2t.xml';
filename MyMap "U:\mytestfolder\fast2t.map";

data _NULL_;
  FILE fast;
  INFILE trans;
  INPUT;
  PUT _infile_;
RUN;

libname fast xmlv2 xmlmap=MyMap automap=REPLACE;

proc datasets
  lib=fast;
run;

/* Print the data set names and their descriptions */
proc print
  data=fast.Dataset;
run;
```

Available Options for Each BEA Data Set

Table 22.4 shows a list of options specific to the NIPA and NIPAUnderlyingDetail data sets. Because these options can change, you can run the following sample code to get an updated list of the allowed options for the NIPA data set:

```
/* First, retrieve an updated list of options for the NIPA data set */

filename fast 'U:\mytestfolder\fasttab.json';
filename MyMap "U:\mytestfolder\test\fasttab.map";

proc http
  url='https://apps.bea.gov/api/data/?&userid=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```

        &method=getparameterlist&datasetname=NIPA'
    method="GET"
    out=fast;
run;

libname posts JSON fileref=fast map=mymap automap=replace;;

proc datasets
    lib=posts;
run;

/* Print the available option names (parameters) and their descriptions */
proc print
    data=posts.results_parameter;
run;

```

You can run the preceding code for any of the BEA data sets by specifying the appropriate data set name in the HTTP procedure statement. This enables you to update the parameter (option) list available for the specified data set.

Available Values for Each Parameter of a Specified BEA Data Set

To retrieve an updated list of the available table names for the NIPA data set, you can run the following sample code:

```

/* First, get a list of the table names (to use in the TABLENAME= option)
   for the NIPA data set from Bureau of Economic Analysis API by using
   the HTTP procedure in the FILENAME statement */

filename fast 'U:\mytestfolder\fasttab.json';
filename MyMap "U:\mytestfolder\test\fasttab.map";

proc http
    url='https://apps.bea.gov/api/data/?&userid=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        &method=getparametervalues&datasetname=NIPA&ParameterName=TableName'
    method="GET"
    out=fast;
run;

libname posts JSON fileref=fast map=mymap automap=replace;;

proc datasets
    lib=posts;
run;

proc print
    data=posts.RESULTS_PARAMVALUE;
run;

```

You can repeat this code for the next option that you want to explore by naming the parameter you want (PARAMETERNAME=) in the HTTP procedure's URL= option. You can also specify a different data set name (in the DATASETNAME=) in the HTTP procedure's URL= option.

In this way, you can easily access the most updated version of each data set that is available on the BEA website. Each data set has its own specific list of parameters, and each parameter has its own specific list of available parameter values. The tables that describe the options and valid parameter values in this document are included for your convenience, but to ensure that you have a complete and updated list of parameter values with their corresponding parameters, it is recommended that you run the preceding sample code periodically for each data set.

The BEA publishes a release schedule at the following URL:

<https://www.bea.gov/news/schedule/full>

Examples: SASEBEA Interface Engine

The following examples use the VALIDVARNAME=ANY option. This option allows special characters to be used in the time series names (SAS variable names) that are generated for the BEA time series data.

Example 22.1: Retrieving Auto and Truck Unit Sales, Production, Inventories, Expenditures, and Price

This example requests auto and truck detail data from the NIPA Underlying Detail data set. The request is for monthly data for the years 2015 and 2016. The SETNAME= option specifies the data set NIUNDERLYINGDETAIL. The TABLENAME= option specifies the table U70205S (Truck and Auto Unit Sales, Production, Inventories, Expenditures, and Price) time series. The FREQ= option selects monthly time series, and the YEAR= option specifies the years 2015 and 2016 as the range of data. The SAS code is shown after the next paragraph. [Output 22.1.1](#) shows the output for the selected date range.

The SET statement reads observations from the input data set Beanip.NIPU, and the DATA step stores them in a SAS data set named myNIPAU. When all the data are retrieved, they are placed in a SAS data set that is named by the OUT= option and is located in the folder specified by the *physical-name* in the LIBNAME statement.

```
options validvarname=any;

title 'Request Truck Unit Sales, Production, Inventories, Expenditures, and Price,
      Monthly, for 2015 and 2016';
LIBNAME beanip sasebea "physical path to your folder for storing the BEA data"
      setname=NIUnderlyingDetail
      TableName="U70205S"
      user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
      freq=M
      outjson=NIPU
      year='2015,2016'
      format=json;

data myNIPAU;
      set beanip.NIPU;
run;
```

```
proc print data=myNIPAU; run;
```

Output 22.1.1 Auto and Truck Unit Sales, Production, Inventories, Expenditures, and Price, Monthly, for 2015 and 2016**Request Truck Unit Sales, Production, Inventories, Expenditures, and Price, Monthly, for 2015 and 2016**

Obs	TableName	Date	ALLO03	AUTO35	AUTO40	BEACPT	BEASMR	CHG03	CHG04	CHG05	CHG07
1	U70205S	2015M01	45.83	3,726.4	3,542.2	4.271	355.9	-3.9	-5.9	-8.0	1.7
2	U70205S	2015M02	46.03	3,619.7	3,481.4	3.847	320.6	-9.9	-7.9	-8.5	-1.3
3	U70205S	2015M03	45.77	3,924.7	3,740.5	4.247	353.9	10.4	10.4	11.2	24.4
4	U70205S	2015M04	44.96	3,866.2	3,537.8	4.247	353.9	0.0	-0.3	0.2	-5.6
5	U70205S	2015M05	44.21	4,059.2	3,579.7	4.488	374.0	5.7	5.9	4.9	-1.9
6	U70205S	2015M06	45.05	3,940.7	3,612.7	4.086	340.5	-9.0	-9.1	-8.8	-0.2
7	U70205S	2015M07	43.19	4,110.8	3,460.2	4.741	395.1	16.0	16.3	11.6	-16.5
8	U70205S	2015M08	45.09	3,958.2	3,641.9	4.177	348.1	-11.9	-12.1	-6.0	15.6
9	U70205S	2015M09	43.85	3,954.2	3,442.5	4.126	343.8	-1.2	-1.3	-2.8	0.8
10	U70205S	2015M10	43.09	4,036.8	3,389.7	4.073	339.4	-1.3	-1.9	2.2	5.4

Obs	CIMPS	CPA43	EXPTS	FRB	FRBSAL	IIP01	INVN	INVS	INVSAL	INVSC	LTAVG	LTCONV
1	107.6	143.964	116.4	4.182	5.434	127.460	1,263.4	1,222.9	2.687	4.9	36,228	35,730
2	106.4	144.450	89.0	3.851	5.250	116.567	1,295.6	1,244.2	2.829	21.3	36,154	35,746
3	112.0	144.574	112.4	4.252	5.622	129.650	1,280.4	1,232.7	2.618	-11.5	36,126	35,800
4	107.9	144.551	110.8	4.241	5.517	129.882	1,269.8	1,241.0	2.685	8.3	36,190	36,073
5	114.5	144.666	113.3	4.492	5.754	136.223	1,217.2	1,250.3	2.595	9.3	36,168	35,631
6	106.8	145.157	108.5	4.085	5.669	124.276	1,208.6	1,232.0	2.595	-18.3	36,269	36,015
7	111.6	144.531	111.7	4.750	5.784	138.703	1,123.2	1,248.6	2.578	16.6	36,385	36,146
8	108.7	144.512	111.1	4.173	5.743	130.385	1,098.1	1,245.2	2.589	-3.4	36,359	35,958
9	110.4	144.423	112.5	4.119	5.581	126.704	1,133.0	1,230.8	2.633	-14.4	36,198	36,028
10	100.6	144.155	115.5	4.040	5.654	129.494	1,176.4	1,185.8	2.504	-45.0	36,174	35,870

Obs	LTCSAL	MIMPS	MVMA	NSAD	NSAF	NSAT	SAARD	SAARF	SAART	SAMR	T1	T2	T3	T4
1	5,068.8	113.0	326.3	386.0	129.4	515.4	5.462	1.928	7.390	455.2	663.0	532.3	100.0	30.7
2	5,173.0	123.2	321.9	419.5	138.7	558.2	5.278	1.935	7.214	439.9	727.6	585.2	111.1	31.3
3	5,416.1	105.9	400.3	525.0	192.5	717.5	5.650	2.133	7.784	470.9	862.9	680.9	144.0	38.0
4	5,450.2	119.4	377.9	477.3	169.1	646.4	5.546	1.971	7.516	462.1	834.0	664.9	132.9	36.2
5	5,467.0	115.9	370.8	563.2	184.9	748.0	5.782	1.967	7.748	481.8	921.6	733.9	150.6	37.1
6	5,391.1	117.6	369.9	491.4	169.8	661.2	5.697	1.978	7.675	474.7	858.5	671.5	143.5	43.5
7	5,471.2	105.9	309.0	481.7	166.2	647.9	5.812	1.883	7.695	484.3	899.8	702.5	156.7	40.6
8	5,508.6	131.8	357.3	496.8	174.0	670.8	5.771	1.955	7.726	480.9	928.5	715.7	173.8	39.0
9	5,427.7	111.4	360.0	446.6	152.4	599.0	5.610	1.925	7.535	467.5	877.0	694.5	145.0	37.5
10	5,270.1	104.0	379.4	447.7	151.9	599.7	5.682	1.888	7.570	473.5	887.9	680.9	168.0	38.9

Obs	TEMD	TEMF	TEMG	TEMH	UPR00	UPR01	UPR02	UPR03	UPR05
1	9.522	7.702	1.382	0.438	25,276	24,153	28,471	26,743	24,139
2	9.682	7.759	1.488	0.434	25,418	24,185	28,797	26,946	24,214
3	10.110	8.023	1.652	0.434	25,437	24,083	29,038	26,982	24,199
4	10.171	8.028	1.705	0.438	25,238	24,086	28,502	26,641	24,068
5	10.197	8.046	1.694	0.457	25,195	24,219	28,085	26,491	24,070
6	10.206	7.935	1.791	0.481	25,058	24,150	27,690	26,433	23,941
7	10.608	8.294	1.838	0.477	25,066	24,175	27,831	26,279	24,044
8	10.682	8.317	1.897	0.468	25,168	23,974	28,716	26,583	24,035
9	10.738	8.465	1.833	0.439	25,148	24,112	28,188	26,470	24,094
10	10.701	8.225	2.022	0.453	25,098	24,173	27,901	26,339	24,110

Output 22.1.1 continued

Request Truck Unit Sales, Production, Inventories, Expenditures, and Price,
Monthly, for 2015 and 2016

Obs	TableName	Date	ALLO03	AUTO35	AUTO40	BEACPT	BEASMR	CHG03	CHG04	CHG05	CHG07
11	U70205S	2015M11	44.08	3,856.9	3,390.0	3.817	318.1	-6.3	-5.2	-5.1	-17.8
12	U70205S	2015M12	44.33	3,683.0	3,272.8	3.823	318.6	0.2	-0.1	3.2	-10.9
13	U70205S	2016M01	41.90	3,941.2	3,123.2	4.007	333.9	4.8	4.8	4.8	10.7
14	U70205S	2016M02	42.40	3,897.5	3,163.5	3.965	330.4	-1.0	-1.1	-2.3	13.2
15	U70205S	2016M03	42.90	3,660.2	3,049.4	3.936	328.0	-0.7	-0.5	0.5	1.3
16	U70205S	2016M04	42.75	3,679.2	3,037.3	3.859	321.6	-2.0	-2.1	0.2	-3.8
17	U70205S	2016M05	43.28	3,660.3	3,098.5	3.738	311.5	-3.1	-3.0	-1.7	-5.2
18	U70205S	2016M06	44.00	3,583.3	3,134.3	4.153	346.1	11.1	10.8	10.9	16.8
19	U70205S	2016M07	43.90	3,642.3	3,165.3	3.954	329.5	-4.8	-4.6	-2.6	-34.8
20	U70205S	2016M08	42.79	3,681.5	3,036.3	3.989	332.4	0.9	0.6	-1.6	51.7

Obs	CIMPS	CPA43	EXPTS	FRB	FRBSAL	IIP01	INVN	INVS	INVSAL	INVSC	LTAVGV	LTCONV
11	112.3	143.917	104.4	3.829	5.488	122.856	1,221.3	1,167.0	2.539	-18.8	36,299	35,907
12	114.2	143.942	108.7	3.824	5.308	126.769	1,178.6	1,156.0	2.600	-11.0	36,497	36,300
13	107.8	143.543	114.1	4.006	5.442	132.883	1,193.3	1,131.5	2.494	-24.5	36,428	36,028
14	108.5	144.078	108.3	3.963	5.411	129.822	1,196.0	1,118.1	2.479	-13.4	36,573	36,235
15	110.9	143.992	110.1	3.943	5.102	130.462	1,172.2	1,120.9	2.635	2.8	36,688	35,908
16	115.1	143.813	116.2	3.859	5.046	130.708	1,149.7	1,120.5	2.664	-0.4	36,566	35,963
17	101.4	143.685	105.3	3.742	5.114	128.502	1,093.7	1,104.7	2.591	-15.8	36,513	35,968
18	100.8	143.238	110.7	4.147	5.079	142.526	1,119.3	1,119.0	2.643	14.3	36,556	35,919
19	111.7	143.445	117.6	3.957	5.185	138.780	1,049.5	1,142.9	2.644	23.9	36,497	36,056
20	105.8	143.406	113.9	3.981	5.112	136.610	1,066.7	1,152.2	2.703	9.3	36,634	36,126

Obs	LTCSAL	MIMPS	MVMA	NSAD	NSAF	NSAT	SAARD	SAARF	SAART	SAMR	T1	T2	T3	T4
11	5,472.0	114.8	312.0	387.8	142.1	529.9	5.516	1.872	7.388	459.6	830.0	637.3	157.4	35.3
12	5,145.6	109.6	278.0	472.1	162.4	634.5	5.336	1.765	7.101	444.7	1,037.9	797.8	199.0	41.1
13	5,261.8	101.5	307.8	361.4	111.7	473.1	5.444	1.742	7.185	453.6	716.2	551.9	132.9	31.4
14	5,313.9	107.0	348.5	427.3	128.6	555.9	5.412	1.776	7.188	451.0	819.1	637.9	148.0	33.1
15	5,176.8	99.3	353.1	502.1	164.8	666.9	5.104	1.758	6.862	425.3	948.1	729.9	177.9	40.4
16	5,419.7	99.8	339.7	447.8	161.7	609.5	5.048	1.791	6.839	420.7	915.7	711.2	170.0	34.4
17	5,377.9	102.9	322.1	468.3	158.0	626.2	5.116	1.770	6.886	426.3	927.4	702.8	190.6	34.0
18	5,451.5	101.5	376.3	452.3	153.2	605.4	5.081	1.756	6.837	423.4	944.4	724.0	184.6	35.7
19	5,626.8	132.6	245.3	437.2	154.4	591.5	5.187	1.722	6.909	432.3	956.1	728.0	197.9	30.1
20	5,516.6	111.2	372.1	433.4	147.4	580.8	5.114	1.695	6.809	426.2	960.0	726.6	199.8	33.6

Obs	TEMF	TEMG	TEMH	UPR00	UPR01	UPR02	UPR03	UPR05
11	10.934	8.420	2.055	0.459	25,018	23,972	28,123	26,345
12	10.391	7.954	2.026	0.411	25,213	24,128	28,519	26,571
13	10.885	8.453	1.972	0.460	25,365	24,658	27,575	26,483
14	10.892	8.486	1.966	0.440	25,073	24,339	27,310	26,226
15	10.427	8.036	1.928	0.462	25,088	24,425	27,013	26,330
16	10.826	8.350	2.058	0.418	24,993	24,542	26,264	26,175
17	10.811	8.140	2.262	0.409	25,361	24,435	28,038	26,619
18	10.892	8.276	2.221	0.395	25,321	24,163	28,672	26,636
19	11.213	8.569	2.266	0.379	25,915	24,503	30,168	27,222
20	11.107	8.502	2.230	0.375	26,094	24,752	30,144	27,293

Output 22.1.1 *continued***Request Truck Unit Sales, Production, Inventories, Expenditures, and Price, Monthly, for 2015 and 2016**

Obs	TableName	Date	ALLO03	AUTO35	AUTO40	BEACPT	BEASMR	CHG03	CHG04	CHG05	CHG07
21	U70205S	2016M09	43.85	3,668.1	3,169.6	3.928	327.3	-1.5	-1.3	-0.8	-7.8
22	U70205S	2016M10	45.63	3,371.9	3,163.5	3.773	314.4	-3.9	-4.1	-4.2	-2.7
23	U70205S	2016M11	44.84	3,458.9	3,133.4	3.752	312.7	-0.6	0.2	-1.2	-5.5
24	U70205S	2016M12	45.46	3,465.8	3,230.2	3.574	297.8	-4.7	-4.8	-4.7	-17.6

Obs	CIMPS	CPA43	EXPTS	FRB	FRBSAL	IIP01	INVN	INVS	INVSAL	INVSC	LTAVG	LTCON
21	111.4	143.575	113.8	3.931	5.211	135.537	1,116.5	1,162.0	2.675	9.8	36,703	36,096
22	104.7	143.863	116.3	3.769	4.945	129.849	1,172.8	1,157.2	2.808	-4.8	36,628	36,217
23	108.0	143.889	112.6	3.775	4.984	128.291	1,251.9	1,177.3	2.834	20.1	36,555	36,020
24	91.8	143.732	105.8	3.593	5.094	122.308	1,209.8	1,156.8	2.724	-20.5	36,632	36,170

Obs	LTCSAL	MIMPS	MVMA	NSAD	NSAF	NSAT	SAARD	SAARF	SAART	SAMR	T1	T2	T3	T4
21	5,691.6	119.3	343.0	419.6	136.7	556.3	5.213	1.719	6.932	434.4	907.0	700.6	174.1	32.4
22	5,707.1	104.6	333.6	377.3	130.0	507.2	4.946	1.687	6.633	412.2	891.1	692.0	169.3	29.9
23	5,489.9	127.5	315.4	374.9	134.1	509.1	4.985	1.712	6.697	415.5	891.7	680.8	181.1	29.8
24	5,637.6	120.4	259.8	444.2	157.1	601.3	5.096	1.726	6.822	424.7	1,118.3	850.5	231.4	36.4

Obs	TEMD	TEMF	TEMG	TEMH	UPR00	UPR01	UPR02	UPR03	UPR05
21	11.017	8.448	2.195	0.374	25,756	24,679	29,024	27,001	24,699
22	11.334	8.837	2.135	0.361	25,661	24,529	28,981	27,074	24,550
23	11.088	8.468	2.246	0.373	25,408	24,363	28,453	26,750	24,351
24	11.456	8.670	2.414	0.372	25,377	24,364	28,367	26,783	24,363

Example 22.2: Retrieving Table IDs 6 and 25 for the Housing Industry

In this example, the BEA data set name is GDPbyIndustry as specified in the SETNAME= option. The TABLEID='6,25' specifies that the statistics for table ID 6 (Components of Value Added by Industry) and table ID 25 (Composition of Gross Output by Industry) are requested for the housing industry (INDUS-TRY='HS'). The YEARS=ALL option specifies the range of data is for all available years. The output is shown in [Output 22.2.1](#), which contains fourteen variables whose names are prefixed with the industry code (HS) followed by '_'. With the exception of the DATE variable, the variable names also have a suffix of '_' followed by the table ID. Between the prefix and suffix is the main part of the name, which is the truncated description of the time series. Together, the three parts, prefix, description and suffix, name each of the time series variables in the SAS output data set. The FREQ= option selects annual data and the OUTJSON= option specifies the name of the SAS output data set (which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS. [Output 22.2.1](#) shows the output for all available years.

```
options validvarname=any;
```

```
title 'Request Housing Statistics from Gross Domestic Product by Industry';
LIBNAME beagi sasebea "physical path to your folder for storing the BEA data"
setname=GDPbyIndustry
TableID='6,25'
user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
```

```
freq=A
industry='HS'
outjson=gdp2i
year=ALL
format=json;

data myGDP2I;
    set beagi.gdp2i;
run;

proc print data=myGDP2I; run;
```


Output 22.2.1 Housing: Components of Value Added and Composition of Gross Output

Request Housing Statistics from Gross Domestic Product by Industry

Obs	Date	_HS_Compensation_of_empl_25	_HS_Energy_inputs_25	_HS_Gross_operating_surp_25
1	1997	8.8	0.0	660.3
2	1998	9.4	0.0	700.0
3	1999	9.9	0.0	748.1
4	2000	10.8	0.0	797.6
5	2001	11.1	0.0	860.3
6	2002	11.9	0.0	890.9
7	2003	11.9	0.0	918.5
8	2004	12.4	0.0	976.8
9	2005	12.8	0.0	1045.9
10	2006	13.2	0.0	1099.5

Obs	_HS_Housing_25	_HS_Intermediate_inputs_25	_HS_Materials_inputs_25	_HS_Purchased-services_i_25
1	857.9	88.0	0.5	87.5
2	911.4	99.1	0.4	98.7
3	968.0	102.5	0.3	102.1
4	1028.5	107.9	0.4	107.5
5	1103.7	113.6	0.2	113.4
6	1148.9	118.5	0.5	117.9
7	1193.8	128.7	0.8	127.9
8	1264.8	132.8	2.8	130.0
9	1355.0	144.2	3.3	140.9
10	1429.1	151.6	3.8	147.8

Obs	_HS_Taxes_on_production_25	_HS_Value_added_25	_HS_Compensation_of_empl_6
1	100.7	769.8	8.8
2	102.9	812.3	9.4
3	107.5	865.5	9.9
4	112.3	920.6	10.8
5	118.6	990.1	11.1
6	127.5	1030.4	11.9
7	134.8	1065.2	11.9
8	142.7	1132.0	12.4
9	152.0	1210.7	12.8
10	164.8	1277.5	13.2

Obs	_HS_Gross_operating_surp_6	_HS_Housing_6	_HS_Taxes_on_production_6
1	660.3	769.8	100.7
2	700.0	812.3	102.9
3	748.1	865.5	107.5
4	797.6	920.6	112.3
5	860.3	990.1	118.6
6	890.9	1030.4	127.5
7	918.5	1065.2	134.8
8	976.8	1132.0	142.7
9	1045.9	1210.7	152.0
10	1099.5	1277.5	164.8

Output 22.2.1 *continued***Request Housing Statistics from Gross Domestic Product by Industry**

Obs	Date	_HS_Compensation_of_empl_25	_HS_Energy_inputs_25	_HS_Gross_operating_surp_25
11	2007	14.0	0.0	1160.6
12	2008	14.3	0.0	1255.0
13	2009	14.2	0.0	1263.2
14	2010	14.1	0.0	1289.8
15	2011	14.8	0.0	1315.4
16	2012	15.5	0.0	1335.5
17	2013	15.6	0.0	1373.8
18	2014	16.5	0.0	1410.5
19	2015	17.5	0.0	1452.8
20	2016	18.0	0.1	1503.4

Obs	_HS_Housing_25	_HS_Intermediate_inputs_25	_HS_Materials_inputs_25	_HS_Purchased-services_i_25
11	1505.9	159.0	9.7	149.3
12	1593.9	143.3	6.5	136.9
13	1616.8	148.1	4.3	143.8
14	1650.9	155.0	3.8	151.3
15	1686.0	162.2	6.9	155.3
16	1726.7	180.5	7.4	173.0
17	1781.3	191.2	9.5	181.6
18	1834.7	198.1	7.9	190.2
19	1895.7	209.0	8.8	200.2
20	1970.6	224.7	10.4	214.2

Obs	_HS_Taxes_on_production_25	_HS_Value_added_25	_HS_Compensation_of_empl_6
11	172.3	1346.9	14.0
12	181.3	1450.5	14.3
13	191.4	1468.7	14.2
14	192.0	1495.9	14.1
15	193.7	1523.9	14.8
16	195.2	1546.2	15.5
17	200.6	1590.1	15.6
18	209.6	1636.6	16.5
19	216.4	1686.7	17.5
20	224.5	1745.9	18.0

Obs	_HS_Gross_operating_surp_6	_HS_Housing_6	_HS_Taxes_on_production_6
11	1160.6	1346.9	172.3
12	1255.0	1450.5	181.3
13	1263.2	1468.7	191.4
14	1289.8	1495.9	192.0
15	1315.4	1523.9	193.7
16	1335.5	1546.2	195.2
17	1373.8	1590.1	200.6
18	1410.5	1636.6	209.6
19	1452.8	1686.7	216.4
20	1503.4	1745.9	224.5

Output 22.2.1 *continued***Request Housing Statistics from Gross Domestic Product by Industry**

Obs	Date	_HS_Compensation_of_empl_25	_HS_Energy_inputs_25	_HS_Gross_operating_surp_25
21	2017	18.9	0.1	1560.9
22	2018	19.9	0.1	1631.6
23	2019	21.2	0.1	1693.9
24	2020	22.1	-0.1	1757.8

Obs	_HS_Housing_25	_HS_Intermediate_inputs_25	_HS_Materials_inputs_25	_HS_Purchased-services_i_25
21	2048.5	233.5	11.2	222.2
22	2132.3	236.2	12.0	224.1
23	2229.7	249.6	11.8	237.6
24	2324.2	273.5	14.1	259.5

Obs	_HS_Taxes_on_production__25	_HS_Value_added_25	_HS_Compensation_of_empl_6
21	235.2	1815.1	18.9
22	244.6	1896.1	19.9
23	265.0	1980.1	21.2
24	270.8	2050.7	22.1

Obs	_HS_Gross_operating_surp_6	_HS_Housing_6	_HS_Taxes_on_production__6
21	1560.9	1815.1	235.2
22	1631.6	1896.1	244.6
23	1693.9	1980.1	265.0
24	1757.8	2050.7	270.8

Example 22.3: Retrieving All Tables for GDP by Industry from the UnderlyingGDPbyIndustry Data Set

In this example, the BEA data set name is UnderlyingGDPbyIndustry, as specified in the SETNAME= option. The TABLEID=ALL specifies that the statistics for all table IDs are requested for the gross domestic product industry (INDUSTRY='GDP'). The YEAR=ALL option specifies that the range of data is for all available years. The output is shown in [Output 22.3.1](#), which contains nine variables, whose names are prefixed with the industry code (GDP) followed by '_'. With the exception of the DATE variable, the variable names also have a suffix of '_', followed by the table ID. Between the prefix and suffix is the main part of the name, which is the truncated description of the time series. Together, the three parts, prefix, description and suffix, name each of the time series variables in the SAS output data set. The FREQ= option selects annual data, and the OUTJSON= option specifies the name of the SAS output data set, which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS. [Output 22.3.1](#) shows the output for all available years.

```
options validvarname=any;
```

```
title 'Request GDP Statistics from Underlying Gross Domestic Product by Industry';
LIBNAME beagu sasebea "physical path to your folder for storing the BEA data"
    setname=underlyingGDPbyIndustry
    TableID=ALL
```

```
user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'  
freq=A  
industry='GDP'  
outjson=gdp3u  
year=ALL  
format=json;  
  
data myGDP3U;  
    set beagu.gdp3u;  
run;  
  
proc print data=myGDP3U; run;
```

Output 22.3.1 Gross Domestic Product: From UnderLyingGDPbyIndustry Data Set, All Table IDS

Request GDP Statistics from Underlying Gross Domestic Product by Industry

Obs	Date	_GDP_Gross_domestic_prod_210	_GDP_Gross_domestic_prod_211	_GDP_Gross_domestic_prod_212
1	1997	8577.6	100.0	70.931
2	1998	9062.8	100.0	74.110
3	1999	9631.2	100.0	77.663
4	2000	10251.0	100.0	80.830
5	2001	10581.9	100.0	81.601
6	2002	10929.1	100.0	82.985
7	2003	11456.5	100.0	85.305
8	2004	12217.2	100.0	88.592
9	2005	13039.2	100.0	91.678
10	2006	13815.6	100.0	94.229
11	2007	14474.2	100.0	96.123
12	2008	14769.9	100.0	96.241
13	2009	14478.1	100.0	93.739
14	2010	15049.0	100.0	96.278

Obs	_GDP_Gross_domestic_prod_213	_GDP_Gross_domestic_prod_214	_GDP_Gross_domestic_prod_215
1		11529.2	74.399
2	4.5	12045.8	75.219
3	4.8	12623.4	76.272
4	4.1	13138.0	78.016
5	1.0	13263.4	79.814
6	1.7	13488.4	81.013
7	2.8	13865.5	82.635
8	3.9	14399.7	84.842
9	3.5	14901.3	87.490
10	2.8	15315.9	90.212
11	2.0	15623.9	92.653
12	0.1	15643.0	94.397
13	-2.6	15236.3	95.019
14	2.7	15649.0	96.164

Obs	_GDP_Gross_domestic_prod_216	_GDP_Gross_domestic_prod_217	_GDP_Gross_domestic_prod_218
1			
2	1.1	4.5	1.1
3	1.4	4.8	1.4
4	2.3	4.1	2.3
5	2.3	1.0	2.3
6	1.5	1.7	1.5
7	2.0	2.8	2.0
8	2.7	3.9	2.7
9	3.1	3.5	3.1
10	3.1	2.8	3.1
11	2.7	2.0	2.7
12	1.9	0.1	1.9
13	0.7	-2.6	0.7
14	1.2	2.7	1.2

Output 22.3.1 *continued***Request GDP Statistics from Underlying Gross Domestic Product by Industry**

Obs	Date	_GDP_Gross_domestic_prod_210	_GDP_Gross_domestic_prod_211	_GDP_Gross_domestic_prod_212
15	2011	15599.7	100.0	97.770
16	2012	16254.0	100.0	100.000
17	2013	16843.2	100.0	101.842
18	2014	17550.7	100.0	104.172
19	2015	18206.0	100.0	106.991
20	2016	18695.1	100.0	108.775
21	2017	19479.6	100.0	111.229
22	2018	20527.2	100.0	114.475
23	2019	21372.6	100.0	117.096
24	2020	20893.7	100.0	113.109

Obs	_GDP_Gross_domestic_prod_213	_GDP_Gross_domestic_prod_214	_GDP_Gross_domestic_prod_215
15	1.5	15891.5	98.157
16	2.3	16254.0	100.000
17	1.8	16553.3	101.769
18	2.3	16932.1	103.662
19	2.7	17390.3	104.662
20	1.7	17680.3	105.703
21	2.3	18079.1	107.742
22	2.9	18606.8	110.326
23	2.3	19032.7	112.279
24	-3.4	18384.7	113.740

Obs	_GDP_Gross_domestic_prod_216	_GDP_Gross_domestic_prod_217	_GDP_Gross_domestic_prod_218
15	2.1	1.5	2.1
16	1.9	2.3	1.9
17	1.8	1.8	1.8
18	1.9	2.3	1.9
19	1.0	2.7	1.0
20	1.0	1.7	1.0
21	1.9	2.3	1.9
22	2.4	2.9	2.4
23	1.8	2.3	1.8
24	1.3	-3.4	1.3

Example 22.4: Retrieving One Table from the FixedAssets Data Set

In this example, the BEA data set name is FixedAssets, as specified in the SETNAME= option. The TABLENAME="FAAt201" specifies that the statistics for one table, named FAAt201, is requested. The table contains current-cost net stock of private fixed assets, equipment, structures, and intellectual property products by type. The YEAR='2015,2016,2017,2018,2019' option specifies that the range of data is for five years, 2015 through 2019. The output is shown in [Output 22.4.1](#), which contains 106 variables whose names are given by the series code (with the exception of the DATE variable). The FREQ= option selects annual data, and the OUTJSON= option specifies the name of the SAS output data set, which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS. [Output 22.4.1](#) shows the output for the last five years, ranging from 2015 to 2019.

```
options validvarname=any;

title 'Request Table FAAt201 Statistics from Fixed Assets, Annual Data';
LIBNAME beaFAAt sasebea "physical path to your folder for storing the BEA data"
    setname=FixedAssets
    TableName="FAAt201"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    outjson=MCOST
    year='2015,2016,2017,2018,2019'
    format=json;

data myMCOST;
    set beaFAAt.MCOST;
run;

proc print data=myMCOST; run;
```

Output 22.4.1 Table FAAt201: From FixedAssets Data Set 2015 to 2019

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	TableName	Date	k1ntotl1ae00_98	NoteRef	k1ntotl1ae10_99	k1ntotl1ae20_100	k1ntotl1ae30_101	k1ntotl1ae40_102
1	FAAt201	2015	479.2	FAAt201	162.4	187.9	79.9	20.5
2	FAAt201	2016	495.3	FAAt201	166.0	200.2	79.7	20.7
3	FAAt201	2017	515.3	FAAt201	170.2	213.7	80.4	21.7
4	FAAt201	2018	531.7	FAAt201	172.9	225.4	81.3	22.2
5	FAAt201	2019	552.8	FAAt201	177.9	238.7	82.1	23.5

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1ae50_103	k1ntotl1ei00_11	k1ntotl1ei10_12	k1ntotl1ei20_13	k1ntotl1ei30_14	k1ntotl1ei40_15	k1ntotl1ei50_16
1	28.5	2,001.0	164.2	135.9	269.4	324.4	620.6
2	28.7	2,035.6	163.3	137.6	273.7	327.6	634.9
3	29.3	2,099.5	167.6	137.4	282.2	334.7	661.6
4	29.8	2,190.1	173.9	143.6	297.3	340.2	691.4
5	30.5	2,267.1	175.9	149.5	301.9	353.9	715.2

Output 22.4.1 *continued***Request Table FAAt201 Statistics from Fixed Assets, Annual Data**

Obs	k1ntotl1ei60_17	k1ntotl1ens0_78	k1ntotl1ens1_79	k1ntotl1ens2_80	k1ntotl1ens3_81	k1ntotl1eo00_26
1	486.5	579.3	150.5	296.0	132.9	1,491.0
2	498.5	619.7	164.0	314.4	141.4	1,501.1
3	516.0	662.9	180.2	334.5	148.2	1,521.3
4	543.6	719.7	202.6	361.3	155.7	1,582.6
5	570.7	771.1	224.5	382.0	164.7	1,619.2

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1eo10_27	k1ntotl1eo33_28	k1ntotl1eo43_29	k1ntotl1eo50_30	k1ntotl1eo60_31	k1ntotl1eo70_32
1	324.2	233.5	232.1	143.8	176.3	32.1
2	328.2	232.9	231.9	138.2	182.9	32.9
3	335.8	233.0	232.1	138.2	189.4	34.0
4	351.9	241.4	241.6	145.0	194.6	36.2
5	356.8	248.2	248.7	145.9	198.6	38.5

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1eo80_33	k1ntotl1ep00_4	k1ntotl1ep11_5	k1ntotl1ep12_10	k1ntotl1ep20_6	k1ntotl1ep31_9	k1ntotl1ep33_7
1	349.0	1,406.3	238.5	11.7	503.3	36.1	413.9
2	354.1	1,446.1	237.9	11.2	518.9	36.7	429.2
3	358.8	1,492.1	245.6	10.2	532.7	37.3	445.0
4	372.0	1,564.4	258.6	9.7	564.5	38.3	463.9
5	382.4	1,618.2	258.4	9.2	588.2	38.5	486.7

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1ep36_8	k1ntotl1eq00_3	k1ntotl1et00_18	k1ntotl1et10_19	k1ntotl1et11_20	k1ntotl1et12_21	k1ntotl1et20_22
1	202.8	6,307.0	1,408.7	536.4	328.1	208.3	210.2
2	212.2	6,460.2	1,477.4	592.2	380.0	212.2	208.5
3	221.2	6,650.9	1,538.0	635.9	416.7	219.2	191.5
4	229.5	6,955.9	1,618.8	704.2	465.7	238.5	173.2
5	237.2	7,197.4	1,693.0	771.7	517.3	254.4	159.0

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1et30_23	k1ntotl1et40_24	k1ntotl1et50_25	k1ntotl1ip00_77	k1ntotl1rd00_82	k1ntotl1rd11_85	k1ntotl1rd12_86
1	404.7	93.4	164.0	2,737.3	1,678.8	478.5	62.8
2	415.9	94.8	166.0	2,905.0	1,790.0	506.5	63.5
3	446.1	97.4	167.2	3,100.0	1,921.8	539.9	65.0
4	473.6	99.1	168.7	3,313.9	2,062.6	577.6	66.1
5	491.0	101.7	169.6	3,530.4	2,206.5	620.9	66.6

Output 22.4.1 *continued***Request Table FAAt201 Statistics from Fixed Assets, Annual Data**

Obs	k1ntotl1rd20_88	k1ntotl1rd23_87	k1ntotl1rd31_89	k1ntotl1rd32_90	k1ntotl1rd70_93	k1ntotl1rd91_96
1	104.1	106.3	49.4	50.1	48.1	18.6
2	111.3	111.6	54.4	54.1	48.7	21.0
3	119.2	116.4	60.2	57.1	51.4	23.4
4	124.4	120.1	65.3	58.1	54.1	25.7
5	125.4	125.1	67.7	55.4	57.1	27.3

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1rd92_97	k1ntotl1rdb5_83	k1ntotl1rdmf_84	k1ntotl1rdnm_92	k1ntotl1rdnp_95	k1ntotl1rdom_91
1	94.8	1,565.5	1,107.6	457.9	113.4	256.5
2	98.8	1,670.3	1,169.7	500.6	119.7	268.4
3	103.8	1,794.6	1,242.5	552.1	127.2	284.8
4	109.0	1,927.9	1,312.2	615.7	134.6	300.5
5	111.4	2,067.8	1,376.1	691.6	138.7	315.0

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1rdon_94	k1ntotl1sb0x_57	k1ntotl1sb10_58	k1ntotl1sb20_59	k1ntotl1sb30_40	k1ntotl1sb31_41
1	409.8	2,743.6	317.4	525.6	813.7	665.8
2	451.9	2,871.8	332.5	535.6	827.9	677.9
3	500.7	3,003.0	343.0	565.0	863.0	707.0
4	561.6	3,178.2	357.7	605.6	917.0	752.1
5	634.5	3,329.9	368.4	641.9	956.1	784.2

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1sb32_42	k1ntotl1sb41_60	k1ntotl1sb42_61	k1ntotl1sc01_46	k1ntotl1sc03_44	k1ntotl1sc04_45
1	147.8	561.7	353.7	440.3	699.6	331.4
2	150.0	609.5	377.4	460.4	715.0	336.3
3	156.1	650.3	398.0	495.4	744.9	347.2
4	164.9	703.1	425.1	535.5	772.5	359.2
5	171.9	750.3	447.6	583.3	802.5	375.6

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1schc_37	k1ntotl1scxx_47	k1ntotl1shc0_39	k1ntotl1si00_48	k1ntotl1sm00_54	k1ntotl1sm01_55
1	4,848.0	534.8	1,025.7	1,515.0	1,784.0	1,662.0
2	4,968.0	542.8	1,044.4	1,548.8	1,721.2	1,592.3
3	5,178.7	560.1	1,090.8	1,629.0	1,662.5	1,527.8
4	5,477.9	577.7	1,161.2	1,741.0	1,616.8	1,474.0
5	5,756.1	602.5	1,214.0	1,835.8	1,622.6	1,471.3

Output 22.4.1 *continued***Request Table FAAt201 Statistics from Fixed Assets, Annual Data**

Obs	k1ntotl1sm02_56	k1ntotl1sn00_65	k1ntotl1so00_66	k1ntotl1soo1_38	k1ntotl1soo2_43	k1ntotl1st00_36
1	122.0	354.5	174.0	1,816.2	212.0	13,442.6
2	128.8	373.7	174.8	1,869.2	216.5	13,765.2
3	134.7	389.3	176.2	1,940.2	227.8	14,249.5
4	142.8	407.0	181.3	2,071.8	244.2	14,982.4
5	151.3	420.8	183.8	2,178.1	258.0	15,683.4

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1stax_63	k1ntotl1stlx_64	k1ntotl1strx_62	k1ntotl1su20_53	k1ntotl1su3x_51	k1ntotl1su4x_52	k1ntotl1su6x_50
1	41.3	415.4	456.7	627.7	1,390.7	533.7	1,924.4
2	44.8	423.7	468.5	631.0	1,471.0	553.4	2,024.4
3	49.2	432.0	481.2	644.5	1,543.7	588.1	2,131.7
4	55.3	443.1	498.5	680.2	1,649.5	638.8	2,288.2
5	61.3	455.8	517.1	700.8	1,745.8	692.5	2,438.3

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1ntotl1suxx_49	k1ptotl1eq00_2	k1ptotl1es00_1	k1ptotl1ip00_76	k1ptotl1st00_35	k1r53101eq00_34
1	2,552.1	6,360.3	41,443.7	2,737.3	32,346.1	53.2
2	2,655.3	6,514.1	43,342.2	2,905.0	33,923.2	53.9
3	2,776.2	6,706.7	45,154.9	3,100.0	35,348.2	55.8
4	2,968.4	7,018.5	47,599.6	3,313.9	37,267.1	62.7
5	3,139.0	7,260.3	49,540.9	3,530.4	38,750.3	62.9

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1r53101hh00_68	k1r53101im00_74	k1r53101mf00_71	k1r53101mh00_72	k1r53101os00_75	k1r53101ot00_73
1	14,363.0	4,214.4	1,989.8	241.1	75.9	250.2
2	15,308.9	4,505.2	2,083.3	241.2	83.4	260.5
3	15,947.6	4,781.6	2,245.3	243.1	88.6	280.9
4	16,817.7	5,092.5	2,373.6	245.4	95.2	279.3
5	17,368.3	5,326.8	2,401.8	266.2	100.1	271.6

Request Table FAAt201 Statistics from Fixed Assets, Annual Data

Obs	k1r53101ps00_69	k1r53101sf00_70	k1r53101st00_67
1	14,121.9	12,132.1	18,903.5
2	15,067.7	12,984.4	20,158.0
3	15,704.4	13,459.2	21,098.7
4	16,572.3	14,198.7	22,284.7
5	17,102.1	14,700.3	23,066.9

Example 22.5: Retrieving US Direct Investment Position in Canada from the MNE Data Set

In this example, the BEA data set name is MNE, as specified in the SETNAME= option. The SERIESID="30" option specifies that the statistics for one series, US Direct Investment Position Abroad on a Historical-Cost Basis, is requested. This series contains the direct investors' net financial claims on their affiliates, which is defined as the value of US direct investors' equity in their foreign affiliates, plus the value of net outstanding loans to their affiliates. The YEAR= option specifies that the selected data are for five years, 2015 to 2019. The output is shown in [Output 22.5.1](#), which contains 19 variables, whose names are made up of the following concatenations in this order: 'SeriesID_', series ID, 'ROW_', row position, 'COL_', and industry code (column code). The OUTJSON= option specifies the name of the SAS output data set, which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS. [Output 22.5.1](#) shows the output for 2015 to 2019. Note that the SASEBEA engine sorts the years in chronological order, even though they can be specified in any order. Also, the output value, "(D)", indicates that the data in the cell have been suppressed to avoid disclosure of data of individual companies.

```
options validvarname=any;

title 'Request US Direct Investment Position in Canada, Annual Data';
LIBNAME beadia sasebea "physical path to your folder for storing the BEA data"
    setname=MNE
    seriesid="30"
    country="100"           /* Canada */
    industry="ALL"
    directionofinvestment=outward
    classification=CountryByIndustry
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    outjson=US7DIA
    year='2015,2016,2017,2018,2019'
    format=json;

data myUS7DIA;
    set beadia.US7DIA;
run;

proc print data=myUS7DIA; run;
```

Output 22.5.1 US Direct Investment Position in Canada: From the MNE Data Set 2015 to 2019**Request US Direct Investment Position in Canada, Annual Data**

Obs	Date	SeriesID_30_ROW_1_COL_0000	SeriesID_30_ROW_2_COL_3250	SeriesID_30_ROW_3_COL_3340
1	2015	361,954	17,339	7,116
2	2016	355,394	17,230	6,371
3	2017	371,274	17,766	7,032
4	2018	373,604	18,262	7,315
5	2019	404,589	19,073	8,255

Obs	SeriesID_30_ROW_4_COL_5225	SeriesID_30_ROW_5_COL_3350	SeriesID_30_ROW_6_COL_5220
1	3,548	3,075	52,029
2	4,369	2,740	51,802
3	4,446	2,828	55,077
4	(D)	2,784	55,985
5	5,082	2,981	67,118

Obs	SeriesID_30_ROW_7_COL_3110	SeriesID_30_ROW_8_COL_5511	SeriesID_30_ROW_9_COL_5100
1	12,571	97,851	7,894
2	9,055	91,773	7,945
3	9,695	95,598	11,214
4	7,301	101,507	11,823
5	7,244	115,463	13,299

Obs	SeriesID_30_ROW_10_COL_3330	SeriesID_30_ROW_11_COL_2100	SeriesID_30_ROW_12_COL_9999
1	5,128	10,096	46,974
2	3,437	10,955	53,268
3	4,545	13,210	55,425
4	3,957	15,162	(D)
5	3,756	11,997	53,612

Obs	SeriesID_30_ROW_13_COL_3999	SeriesID_30_ROW_14_COL_3310	SeriesID_30_ROW_15_COL_5400
1	49,483	7,616	7,534
2	43,800	6,927	8,741
3	39,562	5,906	8,471
4	40,864	4,655	8,231
5	41,124	5,701	8,094

Obs	SeriesID_30_ROW_16_COL_3000	SeriesID_30_ROW_17_COL_3360	SeriesID_30_ROW_18_COL_4200
1	110,232	7,904	25,795
2	99,651	10,092	26,891
3	99,178	11,844	28,655
4	97,480	12,342	27,584
5	102,296	14,162	27,629

Example 22.6: Net Income and Sales for Brazilian, Canadian, and UAE Affiliates of US Parent Enterprises

In this example, the BEA data set name is MNE, as specified in the SETNAME= option. The SERIESID="5,4" option specifies that the statistics for two series, Net Income and Total Sales, are requested. The COUNTRY="202,100,513" option specifies that Brazil, Canada, and the United Arab Emirates (UAE) are the three countries whose data are selected to be retrieved. The INDUSTRY=ALL option indicates that all industries are to be reported. The DIRECTIONOFINVESTMENT=OUTWARD option indicates that the United States is where the parent company is from. The CLASSIFICATION=CountryByIndustry option indicates that each country's data are to be retrieved by industry—in this case, all industries. The YEAR='2012,2011' option specifies that the selected data are for two years, 2012 and 2011. The output is shown in [Output 22.6.1](#), which contains 97 variables, whose names are made up of the following concatenations in this order: 'SeriesID_', series ID, 'ROW_', row position, 'COL_', and industry code (column code). The OUTJSON= option specifies the name of the SAS output data set, which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS. [Output 22.6.1](#) shows the output for 2011 and 2012. Note that the SASEBEA engine sorts the years in chronological order, even though they can be specified in any order. Also, the output value, "(D)", indicates that the data in the cell have been suppressed to avoid disclosure of data of individual companies.

```
options validvarname=any;

title 'Net Income and Sales for Brazil, Canada, and UAE Affiliates
of US Parent Enterprises';
LIBNAME beamne sasebea "physical path to your folder for storing the BEA data"
    setname=MNE
    seriesid="5,4"
    country="202,100,513"
    industry=ALL
    directionofinvestment=outward
    classification=CountryByIndustry
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    nonbankaffiliatesonly=0
    ownershiplevel=0
    outjson=USAMNE
    year='2012,2011'
    format=json;

data myUSAMNE;
    set beamne.USAMNE;
run;

proc print data=myUSAMNE; run;
```

Output 22.6.1 Net Income and Sales for Brazilian, Canadian, and UAE Affiliates of US Parent Enterprises:
From MNE Data Set 2011 to 2012

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	Date	SeriesID_4_ROW_1_COL_0000	SeriesID_4_ROW_2_COL_3250	SeriesID_4_ROW_3_COL_3340
1	2011	202,988	28,132	4,186
2	2012	200,364	29,256	5,221

Output 22.6.1 *continued***Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises**

Obs	SeriesID_4_ROW_4_COL_3350	SeriesID_4_ROW_5_COL_5200	SeriesID_4_ROW_6_COL_3110
1	3,978	14,819	26,318
2 (D)		14,562	31,158

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_7_COL_5100	SeriesID_4_ROW_8_COL_3330	SeriesID_4_ROW_9_COL_2100
1	9,437	11,988	7,404
2	9,674	11,203	5,650

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_10_COL_9999	SeriesID_4_ROW_11_COL_3310	SeriesID_4_ROW_12_COL_5400
1 (D)		3,341	5,888
2 (D)		3,659	6,090

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_13_COL_4400	SeriesID_4_ROW_14_COL_3000	SeriesID_4_ROW_15_COL_3360
1 (D)		128,340	29,802
2 (D)		128,347	27,135

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_16_COL_4200	SeriesID_4_ROW_17_COL_0000	SeriesID_4_ROW_18_COL_3250
1	17,691	664,129	35,394
2	17,547	664,152	35,379

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_19_COL_3340	SeriesID_4_ROW_20_COL_3350	SeriesID_4_ROW_21_COL_5200
1	11,258	3,492	34,254
2	11,479	4,377	28,784

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_22_COL_3110	SeriesID_4_ROW_23_COL_5100	SeriesID_4_ROW_24_COL_3330
1	29,637	8,235	9,125
2	32,441	8,747	8,989

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_25_COL_2100	SeriesID_4_ROW_26_COL_9999	SeriesID_4_ROW_27_COL_3310
1	60,309	46,997	9,365
2	40,382	47,634	9,408

Output 22.6.1 *continued***Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises**

Obs	SeriesID_4_ROW_28_COL_5400	SeriesID_4_ROW_29_COL_4400	SeriesID_4_ROW_30_COL_3000
1	19,634	78,350	282,831
2	21,066	83,512	279,649

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_31_COL_3360	SeriesID_4_ROW_32_COL_4200	SeriesID_4_ROW_33_COL_0000
1	82,428	133,518	27,608
2	83,747	154,377	29,664

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_34_COL_3250	SeriesID_4_ROW_35_COL_3340	SeriesID_4_ROW_36_COL_3350
1	(D)	(D)	(D)
2	(D)	(D)	(D)

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_37_COL_5200	SeriesID_4_ROW_38_COL_3110	SeriesID_4_ROW_39_COL_5100
1	864	(D)	(D)
2	1,043	(D)	(D)

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_40_COL_3330	SeriesID_4_ROW_41_COL_2100	SeriesID_4_ROW_42_COL_9999
1	(D)	8,169	1,091
2	(D)	9,198	842

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_43_COL_3310	SeriesID_4_ROW_44_COL_5400	SeriesID_4_ROW_45_COL_4400
1	31	(D)	1
2	(D)	(D)	(D)

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_4_ROW_46_COL_3000	SeriesID_4_ROW_47_COL_3360	SeriesID_4_ROW_48_COL_4200
1	4,045	0	11,094
2	4,110	0	11,981

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_49_COL_0000	SeriesID_5_ROW_50_COL_3250	SeriesID_5_ROW_51_COL_3340
1	13,084	2,128	-93
2	10,407	895	62

Output 22.6.1 *continued***Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises**

Obs	SeriesID_5_ROW_52_COL_3350	SeriesID_5_ROW_53_COL_5200	SeriesID_5_ROW_54_COL_3110
1 (D)		1,353	508
2 (D)		2,165	597

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_55_COL_5100	SeriesID_5_ROW_56_COL_3330	SeriesID_5_ROW_57_COL_2100
1 977		373	1,140
2 611		4	425

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_58_COL_9999	SeriesID_5_ROW_59_COL_3310	SeriesID_5_ROW_60_COL_5400
1 2,064		-68	505
2 2,343		-57	539

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_61_COL_4400	SeriesID_5_ROW_62_COL_3000	SeriesID_5_ROW_63_COL_3360
1 190		6,519	768
2 -145		4,520	672

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_64_COL_4200	SeriesID_5_ROW_65_COL_0000	SeriesID_5_ROW_66_COL_3250
1 335		87,740	3,984
2 -52		64,754	4,410

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_67_COL_3340	SeriesID_5_ROW_68_COL_3350	SeriesID_5_ROW_69_COL_5200
1 670		222	8,089
2 665		518	5,550

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_70_COL_3110	SeriesID_5_ROW_71_COL_5100	SeriesID_5_ROW_72_COL_3330
1 1,520		626	819
2 1,297		1,007	812

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_73_COL_2100	SeriesID_5_ROW_74_COL_9999	SeriesID_5_ROW_75_COL_3310
1 6,264		39,577	273
2 633		29,882	-444

Output 22.6.1 *continued***Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises**

Obs	SeriesID_5_ROW_76_COL_5400	SeriesID_5_ROW_77_COL_4400	SeriesID_5_ROW_78_COL_3000
1	1,611	3,797	23,769
2	1,854	2,644	15,691

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_79_COL_3360	SeriesID_5_ROW_80_COL_4200	SeriesID_5_ROW_81_COL_0000
1	3,301	4,006	1,753
2	-185	7,493	2,426

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_82_COL_3250	SeriesID_5_ROW_83_COL_3340	SeriesID_5_ROW_84_COL_3350
1	220	(D)	1
2	232	(D)	1

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_85_COL_5200	SeriesID_5_ROW_86_COL_3110	SeriesID_5_ROW_87_COL_5100
1	250	9	(D)
2	209	(D)	(D)

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_88_COL_3330	SeriesID_5_ROW_89_COL_2100	SeriesID_5_ROW_90_COL_9999
1	(D)	528	85
2	(D)	717	56

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_91_COL_3310	SeriesID_5_ROW_92_COL_5400	SeriesID_5_ROW_93_COL_4400
1	(D)	85	-5
2	8	81	-3

Net Income and Sales for Brazil, Canada, and UAE Affiliates of US Parent Enterprises

Obs	SeriesID_5_ROW_94_COL_3000	SeriesID_5_ROW_95_COL_3360	SeriesID_5_ROW_96_COL_4200
1	(D)	0	400
2	(D)	0	(D)

Example 22.7: Retrieving Available Time Series Names for a Specified Indicator from the ITA Data Set

In this example, the BEA data set name is ITA, as specified in the SETNAME= option. The INDICATOR="BalGds" option specifies that the request is for balance on goods time series, and AREAORCOUNTRY=ALL specifies this time series for all countries where it is available. The YEAR='2011,2012' option specifies that the selected data are for two years, 2011 and 2012. The output is shown in [Output 22.7.1](#), which contains 112 variables, whose names are listed by the CONTENTS procedure. The OUTJSON= option specifies the name of the SAS output data set, which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS.

```
options validvarname=any;

title 'Request Balance on Goods Annual ITA Data for Years 2011 and 2012';
LIBNAME beaita sasebea "physical path to your folder for storing the BEA data"
    setname=ITA
    indicator="BalGds"
    areaOrCountry=All
    freq=A
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    outjson=ITAB
    year='2011,2012'
    format=json;

data myITAB;
    set beaita.ITAB;
run;

proc contents data=myITAB; run;
```

Output 22.7.1 Balance on Goods in All Areas or Countries: From the ITA Data Set for 2011 and 2012

Request Balance on Goods Annual ITA Data for Years 2011 and 2012

The CONTENTS Procedure

Data Set Name	WORK.MYITAB	Observations	2
Member Type	DATA	Variables	103
Engine	V9	Indexes	0
Created	11/17/2021 16:46:05	Observation Length	718
Last Modified	11/17/2021 16:46:05	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	latin1 Western (ISO)		

Output 22.7.1 *continued*

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	91
Obs in First Data Page	2
Number of Data Set Repairs	0
Filename	/sastmp/SAS_work78EB000042B6_lax95d04.unx.sas.com/myitab.sas7bdat
Release Created	V.0400M0
Host Created	Linux
Inode Number	2951562770
Access Permission	rw-r--r--
Owner Name	saskff
File Size	128KB
File Size (bytes)	131072

Output 22.7.1 continued

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Label
2	BalGdsAfricasNsaDetail_A_AfricaO	Char	7	Balance on goods; Africa; other countries (those not listed in table 2.3); annual
7	BalGdsAsiaAnhNsaDetail_A_AsiaAnd	Char	7	Balance on goods; Asia and Pacific; other countries (those not listed in table 2.3); annual
32	BalGdsEuropehNsaDetail_A_EuropeO	Char	7	Balance on goods; Europe; other countries (those not listed in table 2.3); annual
50	BalGdsLatAmAOthWestHem_A_LatAmAn	Char	7	Balance on goods; Latin America and Other Western Hemisphere; annual
57	BalGdsMiddlesNsaDetail_A_MiddleE	Char	7	Balance on goods; Middle East; other countries (those not listed in table 2.3); annual
68	BalGdsOthAsidPacExcl15_A_OthAsia	Char	7	Balance on goods; Other Asia and Pacific (excluding 15 countries); annual
69	BalGdsOthAsindPacExcl8_A_OthAsia	Char	7	Balance on goods; Other Asia and Pacific (excluding 8 countries); annual
72	BalGdsOthMideEastExcl5_A_OthMidd	Char	7	Balance on goods; Other Middle East (excluding 5 countries); annual
74	BalGdsOthSouCenAmExcl4_A_OthSout	Char	7	Balance on goods; Other South and Central America (excluding 4 countries); annual
73	BalGdsOthSouenAmExcl13_A_OthSout	Char	7	Balance on goods; Other South And Central America (excluding 13 countries); annual
75	BalGdsOthWesemOthExcl3_A_OthWest	Char	7	Balance on goods; Other Western Hemisphere, Other (excluding 3 countries); annual
90	BalGdsSouthAhNsaDetail_A_SouthAn	Char	7	Balance on goods; South and Central America; other countries (those not listed in table 2.3); annual
1	Date	Char	4	Date of observation
3	TSI_ItaBalGdsAfrica_A	Char	7	Balance on goods; Africa; annual
4	TSI_ItaBalGdsAlgeria_A	Char	7	Balance on goods; Algeria; annual
5	TSI_ItaBalGdsAllOthSeas_A	Char	7	Balance on goods; all other countries (those not listed separately in tables 2.2); annual
6	TSI_ItaBalGdsArgentina_A	Char	7	Balance on goods; Argentina; annual
8	TSI_ItaBalGdsAsiaAndPac_A	Char	7	Balance on goods; Asia and Pacific; annual
9	TSI_ItaBalGdsAustralia_A	Char	7	Balance on goods; Australia; annual
10	TSI_ItaBalGdsAustria_A	Char	7	Balance on goods; Austria; annual
11	TSI_ItaBalGdsBahrain_A	Char	7	Balance on goods; Bahrain; annual
12	TSI_ItaBalGdsBelgium_A	Char	7	Balance on goods; Belgium; annual
13	TSI_ItaBalGdsBermuda_A	Char	7	Balance on goods; Bermuda; annual
14	TSI_ItaBalGdsBrazil_A	Char	7	Balance on goods; Brazil; annual
15	TSI_ItaBalGdsBrunei_A	Char	7	Balance on goods; Brunei; annual
16	TSI_ItaBalGdsBulgaria_A	Char	7	Balance on goods; Bulgaria; annual
17	TSI_ItaBalGdsCaftaDrCountries_A	Char	7	Balance on goods; CAFTA-DR countries; annual
18	TSI_ItaBalGdsCanada_A	Char	7	Balance on goods; Canada; annual
19	TSI_ItaBalGdsChile_A	Char	7	Balance on goods; Chile; annual
20	TSI_ItaBalGdsChina_A	Char	7	Balance on goods; China; annual
21	TSI_ItaBalGdsColombia_A	Char	7	Balance on goods; Colombia; annual
22	TSI_ItaBalGdsCostaRica_A	Char	7	Balance on goods; Costa Rica; annual
23	TSI_ItaBalGdsCroatia_A	Char	7	Balance on goods; Croatia; annual
24	TSI_ItaBalGdsCyprus_A	Char	7	Balance on goods; Cyprus; annual
25	TSI_ItaBalGdsCzechRep_A	Char	7	Balance on goods; Czech Republic; annual
26	TSI_ItaBalGdsDenmark_A	Char	7	Balance on goods; Denmark; annual
27	TSI_ItaBalGdsDominicanRep_A	Char	7	Balance on goods; Dominican Republic; annual
28	TSI_ItaBalGdsEU_A	Char	7	Balance on goods; European Union; annual

Output 22.7.1 *continued*

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len Label
29	TSI_ItaBalGdsElSalvador_A	Char	7 Balance on goods; El Salvador; annual
30	TSI_ItaBalGdsEstonia_A	Char	7 Balance on goods; Estonia; annual
31	TSI_ItaBalGdsEuroArea_A	Char	7 Balance on goods; Euro area; annual
33	TSI_ItaBalGdsEurope_A	Char	7 Balance on goods; Europe; annual
34	TSI_ItaBalGdsFinland_A	Char	7 Balance on goods; Finland; annual
35	TSI_ItaBalGdsFrance_A	Char	7 Balance on goods; France; annual
36	TSI_ItaBalGdsGermany_A	Char	7 Balance on goods; Germany; annual
37	TSI_ItaBalGdsGreece_A	Char	7 Balance on goods; Greece; annual
38	TSI_ItaBalGdsGuatemala_A	Char	7 Balance on goods; Guatemala; annual
39	TSI_ItaBalGdsHonduras_A	Char	7 Balance on goods; Honduras; annual
40	TSI_ItaBalGdsHongKong_A	Char	7 Balance on goods; Hong Kong; annual
41	TSI_ItaBalGdsHungary_A	Char	7 Balance on goods; Hungary; annual
42	TSI_ItaBalGdsIndia_A	Char	7 Balance on goods; India; annual
43	TSI_ItaBalGdsIndonesia_A	Char	7 Balance on goods; Indonesia; annual
44	TSI_ItaBalGdsIntOrgAndUnalloc_A	Char	7 Balance on goods; International organizations and unallocated; annual
45	TSI_ItaBalGdsIreland_A	Char	7 Balance on goods; Ireland; annual
46	TSI_ItaBalGdsIsrael_A	Char	7 Balance on goods; Israel; annual
47	TSI_ItaBalGdsItaly_A	Char	7 Balance on goods; Italy; annual
48	TSI_ItaBalGdsJapan_A	Char	7 Balance on goods; Japan; annual
49	TSI_ItaBalGdsJordan_A	Char	7 Balance on goods; Jordan; annual
51	TSI_ItaBalGdsLatvia_A	Char	7 Balance on goods; Latvia; annual
52	TSI_ItaBalGdsLithuania_A	Char	7 Balance on goods; Lithuania; annual
53	TSI_ItaBalGdsLuxembourg_A	Char	7 Balance on goods; Luxembourg; annual
54	TSI_ItaBalGdsMalaysia_A	Char	7 Balance on goods; Malaysia; annual
55	TSI_ItaBalGdsMalta_A	Char	7 Balance on goods; Malta; annual
56	TSI_ItaBalGdsMexico_A	Char	7 Balance on goods; Mexico; annual
58	TSI_ItaBalGdsMiddleEast_A	Char	7 Balance on goods; Middle East; annual
59	TSI_ItaBalGdsMorocco_A	Char	7 Balance on goods; Morocco; annual
60	TSI_ItaBalGdsNetherlands_A	Char	7 Balance on goods; Netherlands; annual
61	TSI_ItaBalGdsNewZealand_A	Char	7 Balance on goods; New Zealand; annual
62	TSI_ItaBalGdsNicaragua_A	Char	7 Balance on goods; Nicaragua; annual
63	TSI_ItaBalGdsNigeria_A	Char	7 Balance on goods; Nigeria; annual
64	TSI_ItaBalGdsNorway_A	Char	7 Balance on goods; Norway; annual
65	TSI_ItaBalGdsOman_A	Char	7 Balance on goods; Oman; annual
66	TSI_ItaBalGdsOthAfricaExcl1_A	Char	7 Balance on goods; Other Africa (excluding 1 country); annual
67	TSI_ItaBalGdsOthAfricaExcl3_A	Char	7 Balance on goods; Other Africa (excluding 3 countries); annual
70	TSI_ItaBalGdsOthEuropeExcl32_A	Char	7 Balance on goods; Other Europe (excluding 32 countries); annual
71	TSI_ItaBalGdsOthEuropeExcl7_A	Char	7 Balance on goods; Other Europe (excluding 7 countries); annual
76	TSI_ItaBalGdsOthWestHem_A	Char	7 Balance on goods; Other Western Hemisphere; annual
77	TSI_ItaBalGdsPanama_A	Char	7 Balance on goods; Panama; annual
78	TSI_ItaBalGdsPeru_A	Char	7 Balance on goods; Peru; annual
79	TSI_ItaBalGdsPhilippines_A	Char	7 Balance on goods; Philippines; annual
80	TSI_ItaBalGdsPoland_A	Char	7 Balance on goods; Poland; annual
81	TSI_ItaBalGdsPortugal_A	Char	7 Balance on goods; Portugal; annual
82	TSI_ItaBalGdsResidualSeas_A	Char	7 Balance on goods; residual between the commodity-based seasonally adjusted total and the sum of the seasonally adjusted individual countries and the "all other countries" aggregate; annual
83	TSI_ItaBalGdsRomania_A	Char	7 Balance on goods; Romania; annual

Output 22.7.1 *continued*

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len Label
84	TSI_ItaBalGdsRussia_A	Char	7 Balance on goods; Russia; annual
85	TSI_ItaBalGdsSaudiArabia_A	Char	7 Balance on goods; Saudi Arabia; annual
86	TSI_ItaBalGdsSingapore_A	Char	7 Balance on goods; Singapore; annual
87	TSI_ItaBalGdsSlovakia_A	Char	7 Balance on goods; Slovakia; annual
88	TSI_ItaBalGdsSlovenia_A	Char	7 Balance on goods; Slovenia; annual
89	TSI_ItaBalGdsSouthAfrica_A	Char	7 Balance on goods; South Africa; annual
91	TSI_ItaBalGdsSouthAndCenAm_A	Char	7 Balance on goods; South and Central America; annual
92	TSI_ItaBalGdsSouthKorea_A	Char	7 Balance on goods; South Korea; annual
93	TSI_ItaBalGdsSpain_A	Char	7 Balance on goods; Spain; annual
94	TSI_ItaBalGdsSweden_A	Char	7 Balance on goods; Sweden; annual
95	TSI_ItaBalGdsSwitzerland_A	Char	7 Balance on goods; Switzerland; annual
96	TSI_ItaBalGdsTaiwan_A	Char	7 Balance on goods; Taiwan; annual
97	TSI_ItaBalGdsThailand_A	Char	7 Balance on goods; Thailand; annual
98	TSI_ItaBalGdsTurkey_A	Char	7 Balance on goods; Turkey; annual
99	TSI_ItaBalGdsUkIslandsCarib_A	Char	7 Balance on goods; United Kingdom Islands, Caribbean; annual
100	TSI_ItaBalGdsUnitedKingdom_A	Char	7 Balance on goods; United Kingdom; annual
101	TSI_ItaBalGdsVenezuela_A	Char	7 Balance on goods; Venezuela; annual
102	TSI_ItaBalGdsVietnam_A	Char	7 Balance on goods; Vietnam; annual
103	TSI_ItaBalGds_A	Char	7 Balance on goods; annual

Example 22.8: Retrieving Time Series for Two Specified Indicators in the ITA Data Set

In this example, the BEA data set name is ITA, as specified in the SETNAME= option. The INDICATOR="BALCURRACCT,BALGDS" option specifies that the request is for two indicators: balance on current account on goods, and balance of goods. Only one area or country may be specified when more than one indicator is specified. This example specifies AREAORCOUNTRY=CHINA so that only the time series for China are returned. The YEAR= ALL option specifies that the selected data are for all available years, and the FREQ= A option specifies annual data is selected. The output is shown in [Output 22.8.1](#), which contains 2 time series, TSI_ItaBalCurrAcctChina_A and TSI_ItaBalGdsChina_A. These data are available for the years 1999-2019. The OUTJSON= option specifies the name of the SAS output data set, which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS.

[illegible]

```

format=json;

data myIFAB;
    set beaita.IFAB;
run;

proc contents data=myIFAB; run;
proc print data=myIFAB; run;

```

Output 22.8.1 Two Series for China: From the ITA Data Set, 1999 to 2019**Balance on Current Account, and Balance on Goods Data for China****The CONTENTS Procedure**

Data Set Name	WORK.MYIFAB	Observations	22
Member Type	DATA	Variables	3
Engine	V9	Indexes	0
Created	11/17/2021 16:46:10	Observation Length	18
Last Modified	11/17/2021 16:46:10	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	latin1 Western (ISO)		

Engine/Host Dependent Information

Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	3613
Obs in First Data Page	22
Number of Data Set Repairs	0
Filename	/sastmp/SAS_work78EB000042B6_lax95d04.unx.sas.com/myifab.sas7bdat
Release Created	V.0400M0
Host Created	Linux
Inode Number	2951562773
Access Permission	rw-r--r--
Owner Name	saskff
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
1	Date	Char	4	Date of observation
2	TSI_ItaBalCurrAcctChina_A	Char	7	Balance on current account; China; annual
3	TSI_ItaBalGdsChina_A	Char	7	Balance on goods; China; annual

Output 22.8.1 *continued***Balance on Current Account, and Balance on Goods Data for China**

Obs	Date	TSI_ItaBalCurrAcctChina_A	TSI_ItaBalGdsChina_A
1	1999		-68741
2	2000		-83866
3	2001		-83174
4	2002		-103182
5	2003	-131792	-124328
6	2004	-172522	-162623
7	2005	-220545	-202825
8	2006	-261229	-234433
9	2007	-295781	-258662
10	2008	-308264	-268234
11	2009	-263956	-227236
12	2010	-303674	-273067
13	2011	-319099	-295187
14	2012	-334037	-314937
15	2013	-328705	-318764
16	2014	-347168	-344932
17	2015	-365103	-367567
18	2016	-335078	-347098
19	2017	-360682	-374265
20	2018	-407447	-417267
21	2019	-335072	-343371
22	2020	-310588	-309749

Example 22.9: Retrieving Quarterly Time Series for Two Types of Investments from the IIP Data Set

In this example, the BEA data set name is IIP, as specified in the SETNAME= option. The TYPEOFINVESTMENT= option specifies that the request is for two types of investments: US liabilities excluding financial derivatives and US liabilities to foreign official agencies. The COMPONENT= option requests the position statistics (POS) for the two specified investment types. The YEAR= option specifies that the selected data are for 2013, and the FREQ= option specifies the time period to be quarterly, not seasonally adjusted. The output is shown in [Output 22.9.1](#), which contains three variables: DATE, FinLiabsExclFinDerivPos_QNSA, and TSI_lipFinLiabsFoaPos_QNSA. The CONTENTS procedure shows the attributes for each variable, including the description (shown as the label). The OUTJSON= option specifies the name of the SAS output data set, which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS. [Output 22.9.1](#) shows the output for the requested year, 2013.

```
options validvarname=any;

title 'Request Two Quarterly Investment Positions for 2013';
LIBNAME beaiip sasebea "physical path to your folder for storing the BEA data"
    setname=IIP
    typeOfInvestment="FinLiabsFoa,FinLiabsExclFinDeriv"
```



```

component=Pos
freq=QNSA
user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
outjson=IIPQ
year='2013'
format=json;

data myIIPQ;
  set beaiip.IIPQ;
run;

proc contents data=myIIPQ; run;
proc print data=myIIPQ; run;

```

Output 22.9.1 Position on FinLiabsFoa and FinLiabsExclFinDeriv: From the IIP Data Set for 2013

Request Two Quarterly Investment Positions for 2013

The CONTENTS Procedure

Data Set Name	WORK.MYIIPQ	Observations	4
Member Type	DATA	Variables	3
Engine	V9	Indexes	0
Created	11/17/2021 16:46:15	Observation Length	22
Last Modified	11/17/2021 16:46:15	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	latin1 Western (ISO)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	2960
Obs in First Data Page	4
Number of Data Set Repairs	0
Filename	/sastmp/SAS_work78EB000042B6_lax95d04.unx.sas.com/myiipq.sas7bdat
Release Created	V.0400M0
Host Created	Linux
Inode Number	2951562778
Access Permission	rw-r--r--
Owner Name	saskff
File Size	128KB
File Size (bytes)	131072

Output 22.9.1 *continued*

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len Label
1	Date	Char	6 Date of observation
2	FinLiabsExclFinDeriv_Pos	Char	8 U.S. liabilities excluding financial derivatives; quarterly not seasonally adjusted
3	FinLiabsFoa_Pos	Char	8 U.S. liabilities to foreign official agencies; quarterly not seasonally adjusted

Request Two Quarterly Investment Positions for 2013

Obs	Date	FinLiabsExclFinDeriv_Pos	FinLiabsFoa_Pos
1	2013Q1	24559565	5811800
2	2013Q2	24705269	5739553
3	2013Q3	25264912	5838401
4	2013Q4	26573878	5943352

Example 22.10: Retrieving Use of Commodities by Industries (Sector Level) from the InputOutput Data Set

In this example, the BEA data set name is InputOutput, as specified in the SETNAME= option. The TABLEID= option specifies that the request is for TableID 258, Use of Commodities by Industry. The YEAR= option specifies that the selected data are for two years, 2011 and 2012. The output is shown in [Output 22.10.1](#), which contains only the kept variables specified in the KEEP= option: DATE and the variables whose names start with 'Tab258_ROW_FIRE'. Because the row code in the kept variable names is 'FIRE', only the variables in the commodity FIRE (finance, insurance, real estate, rental, and leasing) are kept. The OUTJSON= option specifies the name of the SAS output data set, which is also the name of the JSON data file that is retrieved from the BEA website and read into SAS. [Output 22.10.1](#) shows the output for the requested years, 2011 and 2012.

```
options validvarname=any;

title 'Request Use of Commodities by Industry, Table ID 258';
LIBNAME beaIO sasebea "physical path to your folder for storing the BEA data"
    setname=InputOutput
    tableID='258'
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    outjson=IOINDC
    year='2011,2012'
    format=json;

data myIOIC;
    set beaio.IOINDC;
run;

proc print data=myIOIC(keep=date Tab258_ROW_FIRE:); run;
```

Output 22.10.1 Use of Commodities by Industry (FIRE): From the InputOutput Data Set for 2011, 2012

Request Use of Commodities by Industry, Table ID 258

Obs	Date	Tab258_ROW_FIRE_COL_11	Tab258_ROW_FIRE_COL_21	Tab258_ROW_FIRE_COL_22
1	2011	34862	24406	11153
2	2012	37703	31079	11769

Obs	Tab258_ROW_FIRE_COL_23	Tab258_ROW_FIRE_COL_31G	Tab258_ROW_FIRE_COL_42
1	26628	77329	107213
2	30991	91746	124736

Obs	Tab258_ROW_FIRE_COL_44RT	Tab258_ROW_FIRE_COL_48TW	Tab258_ROW_FIRE_COL_51
1	115014	75138	53260
2	128045	82454	64151

Obs	Tab258_ROW_FIRE_COL_6	Tab258_ROW_FIRE_COL_7	Tab258_ROW_FIRE_COL_81
1	208843	83477	62428
2	219315	91134	68973

Obs	Tab258_ROW_FIRE_COL_F010	Tab258_ROW_FIRE_COL_F020	Tab258_ROW_FIRE_COL_F040
1	2563017	86795	190645
2	2627224	102842	196924

Obs	Tab258_ROW_FIRE_COL_FIRE	Tab258_ROW_FIRE_COL_G	Tab258_ROW_FIRE_COL_PROF
1	866969	134844	184358
2	937733	149063	201335

Obs	Tab258_ROW_FIRE_COL_T001	Tab258_ROW_FIRE_COL_T019
1	2065922	4906380
2	2270227	5197217

Example 22.11: Retrieving Exports and Imports of Advertising Services from the IntlServTrade Data Set

In this example, the BEA data set name is IntlServTrade, as specified in the SETNAME= option. The TYPEOFSERVICE= option specifies that the request is for advertising services. The TRADEDIRECTION= option specifies that both imports and exports data are selected. The AFFILIATION= option specifies that all affiliations are selected. The YEAR= option specifies that the selected data are for all years. Because these data are available starting in 1999 and ending in 2019, only the data for those years are retrieved. The output is shown in [Output 22.11.1](#). Because DESCRIPTION=SHORT is the default, when the PROC PRINT statement uses the LABEL option, each column heading in the output lists the BEA time series indicator using the prefix 'TSI_Ist' for each indicator. If the PROC PRINT statement did not use the LABEL option, then the column headings in the output would have been ISTVAR1, ISTVAR2, ISTVAR3, and ISTVAR4, which are the actual variable names used by the engine. To see a list of all of the variables and their corresponding labels, use the PROC CONTENTS statement.

```

options validvarname=any;

title 'Advertising Services, All Affiliations and All Years';
LIBNAME beaISL sasebea "physical path to your folder for storing the BEA data"
    setname=IntlServTrade
    typeOfService='Advertising'
    TradeDirection=All
    affiliation=All
    areaOrCountry='Canada, Germany'
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    outjson=ISLADS
    year=All
    format=json;

data myISLADS;
    set beaISL.ISLADS;
run;

proc print label data=myISLADS; run;

```

Output 22.11.1 Advertising Services: From the IntlServTrade Data Set, for All Years

Advertising Services, All Affiliations and All Years

Obs	Date of observation	TSI_IstAdvertisingExpCanadaAllAff_A	TSI_IstAdvertisingExpGermanyAllAff_A
1	1999		
2	2000		
3	2001		
4	2002		
5	2003		
6	2004		
7	2005		
8	2006	184	61
9	2007	196	51
10	2008	282	
11	2009	738	75
12	2010	1002	85
13	2011	1189	141
14	2012	1426	210
15	2013	1833	191
16	2014	2279	414
17	2015	2721	845
18	2016	3520	610
19	2017	4574	387
20	2018	5697	517

Obs	TSI_IstAdvertisingImpCanadaAllAff_A	TSI_IstAdvertisingImpGermanyAllAff_A
1		
2		
3		
4		
5		
6		
7		
8	207	156
9	284	182
10	358	163
11	427	131
12	431	149
13	404	243
14	407	251
15	326	189
16	418	151
17	359	194
18	350	170
19	357	186
20	314	178

Output 22.11.1 *continued***Advertising Services, All Affiliations and All Years**

Date of			
Obs	observation	TSI_IstAdvertisingExpCanadaAllAff_A	TSI_IstAdvertisingExpGermanyAllAff_A
21	2019	6615	856
22	2020	6942	

Obs	TSI_IstAdvertisingImpCanadaAllAff_A	TSI_IstAdvertisingImpGermanyAllAff_A
21	406	149
22	340	126

Example 22.12: Retrieving Per Capita Personal Income from the Regional Data Set

In this example, the BEA data set name is *Regional*, as specified in the `SETNAME=` option. The `TABLENAME='SAINC1'` option specifies that the request is for state annual income. The `LINECODE=3` option specifies that per capita personal income is selected. The `GEOFIPS=STATE` option specifies that all states are selected. The `YEAR=` option specifies that the selected data are for the years 2013 through 2017. The output is shown in [Output 22.11.1](#).

```
options validvarname=any;

title 'Request Per Capita Personal Income, for Every State';
LIBNAME bear sasebea "physical path to your folder for storing the BEA data"
  setname=Regional
  TableName="SAINC1"          /* state annual income */
  linecode=3
  geofips=STATE
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  outjson=MSAIS
  year='2013,2014,2015,2016,2017'
  format=json;

data myMSAIS;
  set bear.MSAIS;
run;

proc print data=myMSAIS; run;
```

Output 22.12.1 Per Capita Personal Income for Every State: From the Regional Data Set

Request Per Capita Personal Income, for Every State

Obs	Date	SAINC1-3_GeoFips00000	SAINC1-3_GeoFips01000	SAINC1-3_GeoFips02000	SAINC1-3_GeoFips04000
1	2013	44,894	36,258	52,843	36,739
2	2014	47,017	37,384	55,910	38,211
3	2015	48,891	38,954	57,629	39,635
4	2016	49,812	39,525	56,341	40,618
5	2017	51,811	40,864	57,352	42,267

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips05000	SAINC1-3_GeoFips06000	SAINC1-3_GeoFips08000	SAINC1-3_GeoFips09000
1	36,605	48,549	47,339	62,900
2	38,650	51,332	50,706	65,844
3	39,842	54,632	52,222	67,761
4	40,720	56,667	52,251	69,215
5	41,873	58,942	55,125	70,797

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips10000	SAINC1-3_GeoFips11000	SAINC1-3_GeoFips12000	SAINC1-3_GeoFips13000
1	44,478	67,066	41,069	37,813
2	46,115	70,494	43,388	39,994
3	47,997	74,352	45,493	41,974
4	48,890	76,623	46,253	43,074
5	50,703	78,116	48,774	45,045

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips15000	SAINC1-3_GeoFips16000	SAINC1-3_GeoFips17000	SAINC1-3_GeoFips18000
1	44,378	36,229	47,285	39,829
2	46,658	37,863	49,564	41,274
3	48,566	39,622	51,864	42,778
4	50,089	40,385	52,543	43,780
5	52,222	41,905	54,252	45,278

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips19000	SAINC1-3_GeoFips20000	SAINC1-3_GeoFips21000	SAINC1-3_GeoFips22000
1	43,555	46,291	36,131	41,396
2	44,856	47,169	37,744	43,112
3	46,304	47,527	39,360	43,146
4	46,531	47,557	39,905	42,763
5	47,570	49,033	40,954	44,120

Output 22.12.1 *continued***Request Per Capita Personal Income, for Every State**

Obs	SAINC1-3_GeoFips23000	SAINC1-3_GeoFips24000	SAINC1-3_GeoFips25000	SAINC1-3_GeoFips26000
1	40,452	52,576	56,977	39,696
2	41,964	54,100	59,400	41,383
3	43,888	56,392	62,998	43,655
4	45,003	58,329	65,060	44,809
5	46,737	60,014	67,545	45,983

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips27000	SAINC1-3_GeoFips28000	SAINC1-3_GeoFips29000	SAINC1-3_GeoFips30000
1	47,977	34,222	40,576	40,746
2	50,157	34,846	41,973	42,591
3	52,229	35,470	43,334	44,066
4	52,890	35,946	44,424	44,211
5	54,670	36,817	45,393	46,282

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips31000	SAINC1-3_GeoFips32000	SAINC1-3_GeoFips33000	SAINC1-3_GeoFips34000
1	46,842	39,565	51,010	55,599
2	49,216	41,637	52,839	58,137
3	50,887	44,314	54,935	60,551
4	49,745	45,232	56,629	62,213
5	50,658	47,279	58,791	64,410

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips35000	SAINC1-3_GeoFips36000	SAINC1-3_GeoFips37000	SAINC1-3_GeoFips38000
1	35,149	54,117	38,225	55,222
2	37,304	56,270	40,061	57,963
3	38,211	58,743	41,778	54,503
4	38,907	60,833	42,574	52,592
5	39,592	64,964	44,094	53,142

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips39000	SAINC1-3_GeoFips40000	SAINC1-3_GeoFips41000	SAINC1-3_GeoFips42000
1	41,373	43,951	39,787	46,456
2	42,984	46,522	42,133	48,394
3	44,641	44,785	44,693	50,352
4	45,452	42,399	45,998	51,700
5	46,999	44,295	47,982	53,117

Output 22.12.1 *continued***Request Per Capita Personal Income, for Every State**

Obs	SAINC1-3_GeoFips44000	SAINC1-3_GeoFips45000	SAINC1-3_GeoFips46000	SAINC1-3_GeoFips47000
1	45,990	36,005	45,525	39,519
2	47,937	37,878	47,374	40,841
3	49,625	39,698	49,040	42,648
4	50,320	40,744	49,018	43,637
5	52,341	42,296	50,145	45,134

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips48000	SAINC1-3_GeoFips49000	SAINC1-3_GeoFips50000	SAINC1-3_GeoFips51000
1	44,745	36,628	46,224	48,654
2	47,273	38,328	48,260	50,428
3	47,345	40,668	49,662	52,379
4	46,445	42,008	50,572	53,439
5	49,055	43,711	51,839	55,181

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips53000	SAINC1-3_GeoFips54000	SAINC1-3_GeoFips55000	SAINC1-3_GeoFips56000
1	47,912	35,354	43,194	53,620
2	50,966	36,618	44,905	57,530
3	53,180	37,529	46,800	57,576
4	55,038	37,453	47,509	54,522
5	57,389	39,258	49,105	56,757

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips91000	SAINC1-3_GeoFips92000	SAINC1-3_GeoFips93000	SAINC1-3_GeoFips94000
1	55,140	52,182	42,656	44,996
2	57,505	54,282	44,452	46,674
3	60,177	56,598	46,419	47,973
4	61,822	58,379	47,274	48,331
5	63,907	61,066	48,776	49,585

Request Per Capita Personal Income, for Every State

Obs	SAINC1-3_GeoFips95000	SAINC1-3_GeoFips96000	SAINC1-3_GeoFips97000	SAINC1-3_GeoFips98000
1	39,581	42,793	42,770	47,320
2	41,426	45,136	45,358	50,056
3	43,129	45,316	47,049	53,065
4	43,899	44,681	47,359	54,895
5	45,684	46,967	49,615	57,115

Example 22.13: Using the GOCAS= Option

To use the GOCAS= option, add the options shown in [Table 22.65](#) to your SASEBEA LIBNAME statement.

Table 22.65 Additional Options for SASEBEA Engine When CAS Is Used

Option Specification	Description of Option
CASLIB=	Names the CAS library reference for storing the in-memory tables created by the SASEBEA engine
CASOUT=	Names the in-memory table where the BEA data are stored
GOCAS=ON	Specifies that the CAS server is being used

The following code shows how to use CAS with [Example 22.12](#):

```
options validvarname=any;
title 'Request Per Capita Personal Income, for Every State';

LIBNAME bear sasebea "physical path to your folder for storing the BEA data"
    gocas=on
    caslib=casuserhdfs
    casout=M2MSAIS
    setname=Regional
    TableName="SAINC1"          /* state annual income */
    linecode=3
    geofips=STATE
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    outjson=MSAIS
    year='2013,2014,2015,2016,2017'
    format=json;

data myMSAIS;
    set bear.MSAIS;
run;

proc print data=myMSAIS; run;
```

In the preceding code, you can reference the in-memory CAS table, M2MSAIS, later in the same CAS session to save the data in a SASHDAT file, but do not specify the SASHDAT extension in the file name:

```
proc casutil;
    save casdata="M2MSAIS" casout="<physical path and filename>" replace;
quit;
```

To list the tables in the active caslib, specify the following statements:

```
proc casutil; list; quit;
```

References

- Bureau of Economic Analysis (2011). “Measuring the Nation’s Economy: An Industry Perspective (A Primer on BEA’s Industry Accounts).” Accessed November 16, 2020. https://www.bea.gov/sites/default/files/methodologies/industry_primer.pdf.
- Bureau of Economic Analysis (2019). “BEA API for Data Retrieval.” Accessed November 16, 2020. https://apps.bea.gov/api/_pdf/bea_web_service_api_user_guide.pdf.

Chapter 23

SASEBLS Interface Engine

Contents

Overview: SASEBLS Interface Engine	1596
Using CAS Sessions and CAS Engine Librefs	1596
Getting Started: SASEBLS Interface Engine	1597
Syntax: SASEBLS Interface Engine	1600
The LIBNAME <i>libref</i> SASEBLS Statement	1601
Details: SASEBLS Interface Engine	1605
Data Elements Reference: SASEBLS Interface Engine	1607
Useful Links for BLS Time Series Data	1609
SAS Output Data Set	1609
SAS Output Auxiliary Data Set for Aspects Data	1610
SAS Output Auxiliary Data Set for Catalog Data	1610
SAS OUTJSON File	1610
Examples: SASEBLS Interface Engine	1610
Example 23.1: Retrieving Consumer Price Index (CPI) Data for Multiple Time Series	1610
Example 23.2: Retrieving Producer Price Index (PPI) Data for Multiple Time Series	1613
Example 23.3: Retrieving Import/Export Price Index (EI) Data for Multiple Time Series	1615
Example 23.4: Retrieving Consumer Expenditure Survey (CX) Data	1618
Example 23.5: Retrieving Current Employment Statistics National Survey (CE) Data	1620
Example 23.6: Retrieving State and Metro Area Current Employment (SM) Data	1625
Example 23.7: Retrieving Labor Force Current Population Statistics (CPS) Data	1628
Example 23.8: Retrieving Job Openings and Labor Turnover Statistics (JOLT) Data	1633
Example 23.9: Retrieving Quarterly Census of Employment and Wages (QCEW) Data	1637
Example 23.10: Retrieving Business Employment Dynamics Statistics (BDM) Data	1641
Example 23.11: Retrieving Occupational Employment and Wage Statistics (OEWS) Data	1646
Example 23.12: Retrieving American Time Use Survey (ATUS) Data	1648
Example 23.13: Retrieving Employment Projections (EPP) Data	1654
Example 23.14: Retrieving International Labor Comparisons (ILC) Data	1656
Example 23.15: Retrieving Occupational Requirements Survey (ORS) Data	1660
Example 23.16: Retrieving Work Stoppages (WS) Data	1662
Example 23.17: Retrieving Local Area Unemployment Statistics (LAUS) Data	1666
References	1669

Overview: SASEBLS Interface Engine

The SASEBLS interface engine enables SAS users to retrieve economic data from the United States Bureau of Labor Statistics (BLS) website (www.bls.gov), which is hosted by the principal federal agency in the US Department of Labor that measures labor market activity, working conditions, and price changes in the economy. The BLS data focus on the US economy in the areas of employment and unemployment, consumer and producer prices, consumer expenditures, import and export prices, wages and other worker compensation, productivity and technological change, employment projections, and safety and health statistics. The economic time series on the BLS website contain observation or measurement periods that are associated with data values.

The SASEBLS interface engine uses the LIBNAME statement to enable you to specify how to subset your BLS data, select according to frequency of the data, and retrieve annual average time series at the specified frequency. You can then use a SAS DATA step to perform further subsetting, retrieve the data, and store the resulting time series in a SAS data set and in a CAS data table (if desired).

The SASEBLS interface engine supports 64-bit Windows and Linux X64 (LAX) platforms.

Note that the SASEBLS engine uses the BLS API, but it is not endorsed or certified by the Bureau of Labor Statistics, and that by using the SASEBLS interface, you are agreeing to comply with the BLS terms of use, which are described on the web page at the following URL: <https://www.bls.gov/developers/termsOfService.htm>.

To get started using the SASEBLS engine, see the section “[Getting Started: SASEBLS Interface Engine](#)” on page 1597, which shows how to view the time series data. The sample SAS code for accessing these data appears at the end of that section.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Getting Started: SASEBLS Interface Engine

You can query the Bureau of Labor Statistics (BLS) databases to retrieve the observations or data values for a list of economic time series by specifying the series ID of each time series that you want to read into SAS and by specifying your unique BLS API key. To obtain your own unique API key, visit the BLS website at the following URL:

<https://data.bls.gov/registrationEngine/>

The BLS API key is a 32-character alphanumeric lowercase string, such as 'abcdefghijklmnopqrstu-vwxyz123456', and is represented by 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' in the USER= option in the following example. In addition, the example URLs in this section and in the section “Details: SASEBLS Interface Engine” on page 1605 use the same BLS API key as the argument *your_bls_apikey*.

Now that you are informed about the terms of service of the BLS data, you can use your BLS API registration key to access the BLS data, as shown in the following example.

The following statements enable you to access the three different time series from the Consumer Price Index (CPI) database: 'CUUR0000SA0' from the CPI-U (Urban consumers, US city average, all items), 'CWUR0000SA0' from the CPI-W (Urban wage earners, US city average, all items), and 'SUUR0000SA0' from the C-CPI-U (Chained CPI, all urban consumers). These three time series are listed in the following SAS DATA step and shown in Figure 23.1:

```
options validvarname=any;

DATA inseries;
  length seriesid $ 17;                /* Allow up to 17 bytes for series names */
  seriesid = 'CUUR0000SA0'; output; /* CPI-U, Urban consumers, US city avg */
  seriesid = 'CWUR0000SA0'; output; /* CPI-W, Wage earners, US city avg */
  seriesid = 'SUUR0000SA0'; output; /* C-CPI-U, Chained CPI, all urban consumers*/
RUN;

title 'Retrieve Data for Three Consumer Price Index Time Series';
libname _all_ clear;
```

```

libname bls sasebls "%sysget(BLS) "
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=CPI
  INSET=inseries
  PAYPATH="%sysget(BLS)payload"
  STARTYEAR='2020'
  ENDYEAR='2021'
  FREQ=month
  OUTJSON=cpigs
  FORMAT=json;

data cpi_gsa;
  set bls.cpigs ;
run;

proc contents data=cpi_gsa; run;
proc print data=cpi_gsa; run;

```

Figure 23.1 Getting Started with Consumer Price Index Time Series: cpi_gsa

Retrieve Data for Three Consumer Price Index Time Series

Obs	year	period	periodName	CUUR0000SA0	CWUR0000SA0	SUUR0000SA0
1	2020	M01	January	257.971	251.361	144.813
2	2020	M02	February	258.678	251.935	145.193
3	2020	M03	March	258.115	251.375	144.913
4	2020	M04	April	256.389	249.515	144.142
5	2020	M05	May	256.394	249.521	144.018
6	2020	M06	June	257.797	251.054	144.847
7	2020	M07	July	259.101	252.636	145.747
8	2020	M08	August	259.918	253.597	146.270
9	2020	M09	September	260.280	254.004	146.417
10	2020	M10	October	260.388	254.076	146.382
11	2020	M11	November	260.229	253.826	146.242
12	2020	M12	December	260.474	254.081	146.408
13	2021	M01	January	261.582	255.296	147.125
14	2021	M02	February	263.014	256.843	147.845
15	2021	M03	March	264.877	258.935	148.746
16	2021	M04	April	267.054	261.237	149.903
17	2021	M05	May	269.195	263.612	151.022
18	2021	M06	June	271.696	266.412	152.295
19	2021	M07	July	273.003	267.789	153.076
20	2021	M08	August	273.567	268.387	153.301
21	2021	M09	September	274.310	269.086	153.734
22	2021	M10	October	276.589	271.552	154.884
23	2021	M11	November	277.948	273.042	155.538
24	2021	M12	December	278.802	273.925	155.926

The JSON data that the BLS website returns are placed in a file named by the OUTJSON= option—in this case, *CPIGS.json*. Note that the JSON file extension is excluded from the file name that is specified in the OUTJSON= option. When the SET statement is executed, the JSON data are read into a SAS data set named *cpigs.sas7bdat*, which resides in the location that is specified inside the string enclosed in double quotation

marks in the SASEBLS LIBNAME statement. So, in the preceding example, if the BLS environment variable is set to `/sasusr/bls/test/`, then the SAS data set that is created from reading the downloaded JSON file is placed into the following file:

```
/sasusr/bls/test/cpigs.sas7bdat
```

An equivalent LIBNAME statement that does not use any environment variables could be as follows:

```
libname bls sasebls "/sasusr/bls/test/"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=CPI
  INSET=inseries
  PAYPATH="/sasusr/bls/test/payload"
  STARTYEAR='2020'
  ENDYEAR='2021'
  FREQ=month
  OUTJSON=cpigs
  FORMAT=json;
```

You could also use either a SAS macro variable or a system environment variable to store the value of your BLS API registration key so that the key does not appear explicitly in your SAS code. The SETNAME= option names the BLS data set that you are retrieving data from. The INSET= option specifies SAS input data set that contains the list of time series that you want to retrieve. This option accepts a string that names the SAS data set containing a list of one or more time series that you select (keep) in the resulting SAS data set. The SASEBLS interface engine creates a file named by the PAYPATH= option—in this case, `/sasusr/bls/test/payload.txt`. When you specify the pay path, do not add the .txt file extension; the SASEBLS engine automatically adds it. The result, CPI_GSA, is named in the DATA step and is shown in [Figure 23.1](#). It is more efficient to use the DATA step to store your BLS data in a SAS data set and then refer to the SAS data set directly in your PROC PRINT or PROC SGPLOT statement, but you can also refer to the SASEBLS libref directly, as in the following statements:

```
proc print data=bls.cpigs; run;
```

This statement uses the member name, CPIGS, in the PROC PRINT statement; this usage corresponds to specifying the OUTJSON=CPIGS option. Although using this code might seem easier, it is not as efficient, because every time you use the SASEBLS libref, the BLS interface reads the entire JSON file again into SAS. It is better to refer to the SAS data set repeatedly than to invoke the interface engine repeatedly. For another example that uses more SASEBLS LIBNAME statement options, see the section “[Details: SASEBLS Interface Engine](#)” on page 1605.

Syntax: SASEBLS Interface Engine

The SASEBLS interface engine uses standard engine syntax to read the observations or data values for one or more economic time series. [Table 23.1](#) summarizes the options that the SASEBLS engine uses. There are two required options: `USER='bls_apikey'` and `INSET='bls_input_dataset_name'`.

Table 23.1 Summary of LIBNAME *libref* SASEBLS Options

Option	Description
ASPECTS=	Specifies whether or not to retrieve the aspects data containing name-value pairs of variables associated with a given data point for a specified BLS series ID
CALCULATIONS=	Specifies whether or not to retrieve the net change and percentage change calculations available for 1, 2, 3, 6, or 12 months
CASLIB=	Specifies the name of the caslib where the in-memory CAS table containing the BLS data is stored
CASOUT=	Specifies the name of the in-memory CAS table, which contains the data that the SASEBLS interface engine returns
CATALOG=	Specifies whether or not to retrieve the series descriptive information for a BLS series ID
CONNECT=	Specifies whether or not to use the connect method for a secure connection via a proxy server. You must specify the <code>PROXY=</code> option when you use the <code>CONNECT=ON</code> option.
DEBUG=	Specifies whether or not to use diagnostic message logging in the SAS log window
ENDYEAR=	Specifies the ending year for the observation period ('YYYY' formatted string, optional)
FORMAT=	Specifies a file extension that indicates the type of file to retrieve. Only the JSON format is supported.
FREQ=	Specifies the reporting frequency of the selected data. For the complete list of frequencies, see Table 23.2 .
GOCAS=	Specifies whether to generate an in-memory CAS table for the BLS data
INSET=	Specifies a SAS input data set that contains a list of BLS time series IDs for retrieving BLS data
OUTJSON=	Specifies the name of the output SAS data set and the JSON file requested by the <code>INSET=</code> option
PAYPATH=	Specifies the fully qualified name of the location where the SASEBLS engine stores the payload text file that contains the IDs of the series to retrieve
PROXY=	Specifies the proxy server that you want to use (if you have trouble connecting without specifying a proxy). If you also need the connect method for a secure connection, use the <code>CONNECT=ON</code> option in addition to the <code>PROXY=</code> option. See the <code>CONNECT=</code> option.
SETNAME=	Specifies the data set name of the data you want to access
STARTYEAR=	Specifies the starting year for the observation period ('YYYY' formatted string, optional)
USER=	Specifies the required BLS registration key that enables you to access the data that the BLS website provides

The LIBNAME libref SASEBLS Statement

LIBNAME libref **SASEBLS** 'physical-name' options ;

The LIBNAME statement assigns a SAS library reference (libref) to the physical path of the directory where the SASEBLS interface engine stores the BLS economic data. The required *physical-name* argument specifies the final location of the folder where your BLS data reside in a SAS data set that is named in the OUT= option. The *physical-name* should end with a backslash (for example, C:\physical_name\) when you are in a Windows environment or a forward slash (for example, /user/physical_name/) when you are in a UNIX environment.

You can specify the following *options*:

ASPECTS=TRUE | FALSE

specifies whether or not to retrieve the aspects data that are associated with a given data point for a specified BLS series ID. For each observation, the associated aspects data consist of one or more name-value pairs of variables along with footnotes. When ASPECTS=TRUE and an aspect data file exists for the requested time series, the SASEBLS engine creates an auxiliary data set containing the aspects data. This auxiliary data set is named using the name specified in the OUTJSON= option, which is appended with “_data_aspects”. For example, if OUTJSON=MYCPI, then the aspects data set is named mycpi_data_aspects.sas7bdat.

TRUE specifies that the aspects data be retrieved.

FALSE specifies that no aspects data be retrieved.

By default, ASPECTS=FALSE.

CALCULATIONS=TRUE | FALSE

specifies whether to retrieve (TRUE) or not retrieve (FALSE) the net change and percentage change calculations available for 1, 2, 3, 6, or 12 months. Depending on the frequency of the data and the availability of the data, there might not be calculations

available for all intervals (1, 2, 3, 6, or 12 months).

TRUE retrieves the calculations.

FALSE does not retrieve the calculations.

By default, CALCULATIONS=FALSE. When CALCULATIONS=TRUE and a catalog is available for the requested time series, each calculation series is labeled appropriately. To see the labels, use the CONTENTS procedure with the output data set.

CASLIB=bls_caslib_name

specifies the name of the CAS library (caslib) where the in-memory CAS table is stored when the JSON data from the BLS website are read into SAS. You can use this option to name your caslib where the in-memory CAS table is stored. The default value is the name of the active caslib. The GOCAS=ON option is required when you specify the CASLIB= option.

CASOUT=*libref.data-table***OUTCAS=***libref.data-table*

specifies the name of the in-memory CAS table that is created when the JSON data from BLS are read into SAS. The default value is the name that you specify in the OUT= option. The in-memory table is created only when you also specify the GOCAS=ON option. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section “Using CAS Sessions and CAS Engine Librefs” on page 1596.

CATALOG=TRUE | FALSE

specifies whether or not to retrieve additional descriptive data about the requested time series. These catalog data include the series description (series title); the seasonality description; the survey name description; the survey abbreviation; and (where appropriate) descriptions such as measured data type, commerce industry, occupation, and regional area. When CATALOG=TRUE and a catalog is available for the requested time series, the SASEBLS engine creates an auxiliary data set that contains the catalog data. This auxiliary data set is named using the name specified in the OUTJSON= option, which is then appended with “_catalog”. For example, if OUTJSON=MYCPI, then the catalog data set is named mycpi_catalog.sas7bdat.

TRUE retrieves additional descriptive data about the requested time series.

FALSE does not retrieve additional descriptive data about the requested time series.

By default, CATALOG=FALSE.

CONNECT=ON | OFF

specifies whether to use (ON) or not use (OFF) the connect method along with the PROXY= option.

NOTE: To use the connect method, you must use the PROXY= option and specify your proxy server in addition to the CONNECT=ON option. For more information about a secure connection, see the [PROXY=](#) option.

DEBUG=ON | OFF

specifies whether or not to include diagnostic message logging in the SAS log window. This information can be useful for troubleshooting. Specify DEBUG=ON to see all the log messages (including debugging information) in the SAS log. By default, DEBUG=OFF.

ENDYEAR=*'bls_endyear'*

specifies the ending year of the time series in the format 'YYYY'.

FORMAT=*bls_jsonformat*

specifies the format of the file to be retrieved from the BLS website. Although BLS can report data in many formats, the SASEBLS engine supports only the JSON format (the default).

FREQ=*'bls_frequency_code'*

specifies the frequency of the data that you want to retrieve. You can specify the option values shown in the Frequency Code column in [Table 23.2](#). For an example that retrieves monthly data, see [Example 23.1](#).

Table 23.2 BLS Frequency Codes

Period	Period Description	Frequency Code	Description
M01–M12	Jan.–Dec.	M	Displays data on a monthly basis
M13	Annual average	M13	Displays annual averages of monthly data
M01–M12	Monthly	Mon	Displays data on a monthly basis
M01–M12	Monthly	Month	Displays data on a monthly basis
M01–M12	Monthly	Monthly	Displays data on a monthly basis
Q01–Q04	1st–4th quarter	Q	Displays data on a quarterly basis
Q05	Annual average	Q05	Displays annual averages of quarterly data
Q01–Q04	1st–4th quarter	QTR	Displays data on a quarterly basis
Q01–Q04	1st–4th quarter	Quarter	Displays data on a quarterly basis
Q01–Q04	1st–4th quarter	Quarterly	Displays data on a quarterly basis
S01–S02	1st–2nd half	S	Displays data on a semiannual basis
S03	Annual average	S03	Displays annual averages of semiannual data
S01–S02	1st–2nd half	Semiannual	Displays data on a semiannual basis
A01	Annual	A or A01	Displays data on an annual basis
A02	Annual average	A02	Displays annual averages of annual data
A02,S03,Q05,M13	Annual average	Annual_avg	Displays data for annual averages
A01	Annual	Year	Displays data on an annual basis
A01	Annual	Yearly	Displays data on an annual basis

GOCAS=ON | OFF

specifies whether to create (ON) or not create (OFF) an in-memory CAS table of the BLS data (in addition to the SAS data set that is created). When GOCAS=ON, the SASEBLS engine assumes that there is an active CAS session running when it executes the LIBNAME statement. By default, GOCAS=OFF.

INSET=bls_input_SAS_dataset_name

specifies the name of the input SAS data set that contains a list of time series to include in the output SAS data set. The list is in the following format:

```
DATA \Argument{bls\_input\_SAS\_dataset\_name};
    seriesid = \Argument{\Quotes{bls\_seriesid\_name1}}; output;
    seriesid = \Argument{\Quotes{bls\_seriesid\_name2}}; output;
    .
    .
    .
    seriesid = \Argument{\Quotes{bls\_seriesid\_nameN}}; output;
run;
```

The BLS API enables you to retrieve up to 50 series IDs in one request. To see a list of all series IDs for a particular survey, such as CU, visit the web page at the following URL: <https://download.bls.gov/pub/time.series/cu/cu.series>.

The CPI and PPI survey names are as follows: CU, CW, SU, AP, LI, MU, MW, ND, PC, PD, WP, and WD. You can replace the 'CU' with a different survey name to see a list of that survey's series IDs. For example, for the WD survey, use the following URL

: <https://download.bls.gov/pub/time.series/wd/wd.series>.

For examples that retrieve CPI and PPI data, see [Example 23.1](#) and [Example 23.2](#), respectively.

OUTJSON=*bls_json_file_name*

specifies the name of the JSON file (downloaded to the SAS Work library) and the name of the SAS data set that is created when the JSON data are read into SAS. The SAS data set is stored in the folder that is specified by *physical-name*.

PAYPATH='bls_payload_path_and_filename'

specifies the physical path and file name of the *payload.txt* file. Do not specify the .txt file extension, because the SASEBLS engine automatically adds .txt to the end of the *payload.txt* file name. The *payload.txt* file is

created by the engine and contains information from the INSET= option, as well as information from the following options: STARTYR=, ENDYR=, CATALOG=, and ASPECTS=.

PROXY='bls_proxyserver'

specifies which proxy server to use. This option is not required. The specified proxy server is used only when a connection-refused error or a connection-timed-out error occurs. For *bls_proxyserver*, specify the server's HTTP address followed by a colon and the port number, and enclose that string in double quotation marks; for example, PROXY="http://inetgw.unx.sas.com:8118". See also the [CONNECT=](#) option.

SETNAME=*bls_setname*

DATASETNAME=*bls_setname*

DATASET=*bls_setname*

SETID=*bls_setname*

specifies the name of the BLS data set to retrieve the BLS data from. [Table 23.3](#) shows the list of available BLS data set names. This release of SASEBLS supports the data set names shown in [Table 23.4](#). See [Example 23.1](#) for a CPI example, [Example 23.2](#) for a PPI example, and [Example 23.3](#) for an IPP example.

STARTYEAR='bls_startyear'

specifies the starting year of the time series in the format 'YYYY'.

USER='bls_user'

APIKEY='bls_user'

REGISTRATIONKEY='bls_user'

specifies the unique user registration key, enclosed in quotation marks, that the BLS assigns to the user (subscriber) to allow access to the data provided by the BLS website. The BLS access key is a 32-character alphanumeric lowercase string that you can request from the website at the following URL: <https://data.bls.gov/registrationEngine/>.

Details: SASEBLS Interface Engine

Table 23.3 shows all the data sets that BLS offers. Currently, the SASEBLS interface engine enables you to access only the time series data from the BLS data sets shown in Table 23.4. To specify a supported BLS data set, use one of the data set names in this table.

To access the data set, specify SETNAME=<data set name>, as shown in the SASEBLS LIBNAME statement in the section “Getting Started: SASEBLS Interface Engine” on page 1597.

Table 23.3 BLS Data Sets

Data Set	Long Name	Surveys	Subject Area
CPI	Consumer Price Index	CU CW SU AP LI MU MW	Inflation and Prices
PPI	Producer Price Index	ND PC PD WP WD	Inflation and Prices
IPP	Import/Export Price Index	EI	Inflation and Prices
CEX	Consumer Expenditure Surveys	CX	Inflation and Prices
ECT	Employment Cost Trends	CI CM	Inflation and Prices
CES	Current Employment Statistics	CE EE	Employment
SAE	State and Metro Area Employment, Hours, Earnings	SM	Employment
CPS	Current Population Survey	LN LE LU FM	Employment
JOLT	Job Openings and Labor Turnover	JT JL	Employment
QCEW	Qtrly Census of Employment and Wages	EN	Employment
BDM	Qtrly Business Employment Dynamics	BD GG	Employment
OES	Occupational Employment Statistics	OE	Employment
EPP	Employment Projections	EP	Employment
ILC	International Labor Comparisons	IN	Employment
ORS	Occupational Requirements Survey	OR	Employment
WSP	Work Stoppages	WS	Work Stoppages
LAUS	Local Area Unemployment Statistics	LA	Unemployment
MLS	Mass Layoff Statistics, National	ML	Unemployment
NCS	National Compensation	EB NB NC	Pay and Benefits
ECT	Employment Cost Index	CI CM EC CC	Pay and Benefits
MWE	Modeled Wage Estimates	WM	Pay and Benefits
CEX	Consumer Expenditure Index	CX	Spending and Time Use
ATUS	American Time Use Survey	TU	Spending and Time Use
LPC	Major Sector Productivity and Cost Index	PR IP	Productivity
MFP	Multi-Factor Productivity	MP IP	Productivity
IIF_1	Injuries, Illnesses, Fatalities	IS CS FW HC	Workplace Injuries
IIF_2	Injuries, Illnesses, Fatalities	II CD CF CH	Workplace Injuries
IIF_3	Fatal Injuries and Illnesses	FI HS SH SI	Workplace Injuries
MXP	Import/Export	EI	International
REGIONAL	State and Local, Geographic	BG GP SA SM SU NL	Regional

Data Elements Reference: SASEBLS Interface Engine

Table 23.4 shows where to find a summary of each data set’s details. This table lists the supported data sets and gives useful links to information about each data set, such as included surveys and links to details about each survey in that data set. The table also has links to examples that show how to retrieve the data, including example code for each data set.

Table 23.4 Supported BLS Data Sets

Data Set	Survey Name	Series Details	Example
ATUS	TU	https://download.bls.gov/pub/time.series/tu/tu.txt	Example 23.12
BDM	BD	https://download.bls.gov/pub/time.series/bd/bd.txt	Example 23.10
	GG	https://download.bls.gov/pub/time.series/gg/gg.txt	
CES	CE	https://download.bls.gov/pub/time.series/ce/ce.txt	Example 23.5
	EE	https://download.bls.gov/pub/time.series/ee/ee.txt	
CEX	CX	https://download.bls.gov/pub/time.series/cx	Example 23.4
CPI	AP	https://download.bls.gov/pub/time.series/ap/ap.txt	Example 23.1
	CU	https://download.bls.gov/pub/time.series/cu/cu.txt	
	CW	https://download.bls.gov/pub/time.series/cw/cw.txt	
	LI	https://download.bls.gov/pub/time.series/li/li.txt	
	MU	https://download.bls.gov/pub/time.series/mu/mu.txt	
	MW	https://download.bls.gov/pub/time.series/mw/mw.txt	
	SU	https://download.bls.gov/pub/time.series/su/su.txt	
CPS	FM	https://download.bls.gov/pub/time.series/fm/fm.txt	Example 23.7
	LE	https://download.bls.gov/pub/time.series/le/le.txt	
	LN	https://download.bls.gov/pub/time.series/ln/ln.series	
	LU	https://download.bls.gov/pub/time.series/lu/lu.txt	
EPP	EP	https://download.bls.gov/pub/time.series/ep/ep.txt	Example 23.13
ILC	IN	https://download.bls.gov/pub/time.series/in/in.txt	Example 23.14
IPP	EI	https://download.bls.gov/pub/time.series/ei/ei.txt	Example 23.3
JOLT	JL	https://download.bls.gov/pub/time.series/jl/jl.txt	Example 23.8
	JT	https://download.bls.gov/pub/time.series/jt/jt.txt	
LAUS	LA	https://download.bls.gov/pub/time.series/la/la.txt	Example 23.17
OES	OE	https://download.bls.gov/pub/time.series/oe/oe.txt	Example 23.11
ORS	OR	https://download.bls.gov/pub/time.series/or/or.txt	Example 23.15
PPI	ND	https://download.bls.gov/pub/time.series/nd/nd.txt	Example 23.2
	PC	https://download.bls.gov/pub/time.series/pc/pc.txt	
	PD	https://download.bls.gov/pub/time.series/pd/pd.txt	
	WD	https://download.bls.gov/pub/time.series/wd/wd.txt	
	WP	https://download.bls.gov/pub/time.series/wp/wp.txt	
QCEW	EN	https://data.bls.gov/PDQWeb/en https://data.bls.gov/cgi-bin/dsrv?en	Example 23.9
	EW	https://data.bls.gov/PDQWeb/ew https://data.bls.gov/cgi-bin/dsrv?ew	
SAE	SM	https://download.bls.gov/pub/time.series/sm/sm.txt	Example 23.6
WSP	WS	https://download.bls.gov/pub/time.series/ws/ws.txt	Example 23.16

You can learn how to retrieve data from these supported BLS data sets by using the sample code shown in the section “Examples: SASEBLS Interface Engine” on page 1610.

Useful Links for BLS Time Series Data

To obtain a list of all available surveys of BLS economic data, enter the following URL in your web browser:

`https://api.bls.gov/publicAPI/v2/surveys`

To obtain a list of the most popular BLS and survey-specific series, use the following URL:

`https://api.bls.gov/publicAPI/v2/timeseries/popular`

To obtain a list of the most popular survey-specific series, include the survey name:

`https://api.bls.gov/publicAPI/v2/timeseries/popular?survey=LA`

To obtain a list of the most recent data points for a given BLS series ID, such as series ID LAUCN040010000000005, use the following form of the URL:

`https://api.bls.gov/publicAPI/v2/timeseries/data/LAUCN040010000000005?latest=true`

For a complete list of the flat-file equivalent data for all BLS time series (from the LABSTAT data server), use the following URL. You can use this series list to see details about each series by clicking on the series ID that you are interested in.

`https://download.bls.gov/pub/time.series/`

To see an overview of the contents of the entire catalog, click on the file, *overview.txt*, located at the following URL:

`https://download.bls.gov/pub/time.series/overview.txt`

To see a list of the limitations of the BLS API (application interface), use the following URL:

`https://www.bls.gov/developers/api_faqs.htm#register1`

SAS Output Data Set

You can use the SAS DATA step to write the selected BLS data to a SAS data set. This enables you to use SAS software to easily analyze the data. If you specify the name of the output data set in the DATA statement, the engine supervisor creates a SAS data set that has the specified name in either the SAS Work library or, if specified, the User library. The SASEBLS engine also keeps a copy of the output data set in the location that is specified by the *physical-name* in the LIBNAME *libref* SASEBLS statement, which is described in the section “[The LIBNAME libref SASEBLS Statement](#)” on page 1601.

The contents of the SAS data set include the date of each observation and the series name of each series that is read from the BLS data source. When CALCULATIONS=TRUE and the calculations time series data exist, then all available calculations time series are included in the SAS output data set.

You can use the PRINT and CONTENTS procedures to print your output data set and its contents. Alternatively, you can view your SAS output observations by opening the desired output data set in a SAS Explorer window. You can also use the SQL procedure with your SASEBLS libref to create a custom view of your data.

SAS Output Auxiliary Data Set for Aspects Data

When ASPECTS=TRUE and the aspects data exist, then an auxiliary data set is created containing the aspects data. It is named using the name of the output data set and appended with “_data_aspects”. The aspects data set is placed in the location that is specified by the *physical-name* in the LIBNAME *libref* SASEBLS statement, which is described in the section “The LIBNAME *libref* SASEBLS Statement” on page 1601.

SAS Output Auxiliary Data Set for Catalog Data

When CATALOG=TRUE and the catalog data exist, then an auxiliary data set is created containing the catalog data. It is named using the name of the output data set and appended with “_catalog”. The catalog data set is placed in the location that is specified by the *physical-name* in the LIBNAME *libref* SASEBLS statement, which is described in the section “The LIBNAME *libref* SASEBLS Statement” on page 1601. See Output 23.5.2 for an example of the information contained in the CATALOG data set.

SAS OUTJSON File

The SAS JSON data (in JSON format) that the BLS website returns are placed in a file named by the OUTJSON= option. The SAS JSON data file is placed in the SAS Work directory, but the SAS data set that is created by reading the JSON data into SAS is placed in the location that is specified by the *physical-name* in the LIBNAME *libref* SASEBLS statement, which is described in the section “The LIBNAME *libref* SASEBLS Statement” on page 1601.

Examples: SASEBLS Interface Engine

Example 23.1: Retrieving Consumer Price Index (CPI) Data for Multiple Time Series

This example shows how to use multiple time series IDs to retrieve Average Price (AP) data, all Urban Consumers (SU) chained Consumer Price Index (CPI) data, Urban Wage Earners and Clerical Workers (CW) current series data, and all Urban Consumers (CU) current series data from the CPI databases. You can view these data at the following URL:

<https://www.bls.gov/cpi/data.htm>

INSERIES is an input SAS data set that lists the IDs of the time series that you want to retrieve from the CPI database. In this example, this input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, specify the name of the input data set as ‘INSERIES’. Next, the length of the series ID is specified as 17 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical

path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= name and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES for the list of series to be retrieved using their IDs. The date range of the data starts with 2019, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year to include in the range of data to be retrieved. FREQ=MONTH specifies the selection of only the monthly data, and the FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.1.1](#) shows the CPI data that are retrieved.

```
options validvarname=any;

DATA inseries;
  length seriesid $ 17;
  seriesid = 'CUUR0000SA0'; output; /* CPI-U, US city avg, all items */
  seriesid = 'APU0000704111'; output; /* AP, CPI average price data */
  seriesid = 'APU0000711211'; output; /* bananas */
  seriesid = 'CWUR0000SA0'; output; /* CPI-W, Wage earners */
  seriesid = 'APU0000702111'; output; /* bread */
  seriesid = 'APU0000706111'; output; /* chicken */
  seriesid = 'SUUR0000SA0'; output; /* C-CPI-U, Chained CPI, all urban consumers*/
  seriesid = 'APU0000708111'; output; /* eggs */
  seriesid = 'APU0000709112'; output; /* milk */
RUN;

title 'Retrieve Consumer Price Index Data for Multiple Time Series';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=CPI
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  STARTYEAR='2019'
  ENDYEAR='2021'
  FREQ=month
  OUTJSON=blsex01
  FORMAT=json;

data myCPI;
  set bls.blsex01 ;
run;

proc contents data=myCPI; run;
proc print data=myCPI; run;
```

Output 23.1.1 Retrieve Consumer Price Index Data for Multiple Time Series**Retrieve Consumer Price Index Data for Multiple Time Series**

Obs	year	period	periodName	APU0000702111	APU0000704111	APU0000706111	APU0000708111
1	2019	M01	January	1.274	5.518	1.473	1.554
2	2019	M02	February	1.282	5.502	1.486	1.557
3	2019	M03	March	1.261	5.608	1.468	1.544
4	2019	M04	April	1.285	5.552	1.463	1.463
5	2019	M05	May	1.289	5.812	1.479	1.362
6	2019	M06	June	1.280	5.878	1.590	1.203
7	2019	M07	July	1.281	5.701	1.562	1.243
8	2019	M08	August	1.275	5.576	1.514	1.219
9	2019	M09	September	1.296	5.571	1.487	1.383
10	2019	M10	October	1.325	5.653	1.538	1.282
11	2019	M11	November	1.361	5.505	1.430	1.405
12	2019	M12	December	1.363	5.474	1.450	1.535
13	2020	M01	January	1.351	5.505	1.409	1.461
14	2020	M02	February	1.375	5.497	1.362	1.449
15	2020	M03	March	1.374	5.257	1.400	1.525
16	2020	M04	April	1.406	5.346	1.571	2.019
17	2020	M05	May	1.412	5.348		1.640
18	2020	M06	June	1.474	5.772	1.747	1.554
19	2020	M07	July	1.485	5.776	1.712	1.401
20	2020	M08	August	1.495	5.559	1.609	1.328
21	2020	M09	September	1.492	5.619	1.542	1.353
22	2020	M10	October	1.503	5.722	1.580	1.408

Obs	APU0000709112	APU0000711211	CUUR0000SA0	CWUR0000SA0	SUUR0000SA0
1	2.913	0.576	251.712	245.133	142.001
2	2.900	0.571	252.776	246.218	142.571
3	2.944	0.581	254.202	247.768	143.297
4	2.980	0.567	255.548	249.332	143.926
5	2.963	0.579	256.092	249.871	144.183
6	3.054	0.574	256.143	249.747	144.243
7	3.031	0.569	256.571	250.236	144.409
8	3.045	0.578	256.558	250.112	144.388
9	3.102	0.569	256.759	250.251	144.428
10	3.119	0.573	257.346	250.894	144.629
11	3.189	0.573	257.208	250.644	144.518
12	3.188	0.573	256.974	250.452	144.251
13	3.253	0.570	257.971	251.361	144.813
14	3.196	0.574	258.678	251.935	145.193
15	3.248	0.576	258.115	251.375	144.913
16	3.267	0.581	256.389	249.515	144.142
17	3.210	0.589	256.394	249.521	144.018
18	3.198	0.588	257.797	251.054	144.847
19	3.255	0.580	259.101	252.636	145.747
20	3.406	0.576	259.918	253.597	146.270
21	3.448	0.574	260.280	254.004	146.417
22	3.380	0.577	260.388	254.076	146.382

Output 23.1.1 *continued***Retrieve Consumer Price Index Data for Multiple Time Series**

Obs	year	period	periodName	APU0000702111	APU0000704111	APU0000706111	APU0000708111
23	2020	M11	November	1.515	5.754	1.639	1.450
24	2020	M12	December	1.538	5.831	1.621	1.481
25	2021	M01	January	1.546	5.831	1.595	1.466
26	2021	M02	February	1.537	5.778	1.583	1.597
27	2021	M03	March	1.526	5.852	1.543	1.625
28	2021	M04	April	1.510	6.215	1.515	1.620
29	2021	M05	May	1.511	6.346	1.486	1.625
30	2021	M06	June	1.510	6.670	1.474	1.642
31	2021	M07	July	1.491	6.857	1.435	1.642
32	2021	M08	August	1.467	7.096	1.472	1.709
33	2021	M09	September	1.580	7.220	1.504	1.835
34	2021	M10	October	1.526	7.317	1.523	1.821
35	2021	M11	November	1.547	7.265	1.583	1.718
36	2021	M12	December	1.532	7.211	1.606	1.788

Obs	APU0000709112	APU0000711211	CUUR0000SA0	CWUR0000SA0	SUUR0000SA0
23	3.425	0.578	260.229	253.826	146.242
24	3.535	0.572	260.474	254.081	146.408
25	3.468	0.597	261.582	255.296	147.125
26	3.368	0.590	263.014	256.843	147.845
27	3.348	0.594	264.877	258.935	148.746
28	3.447	0.604	267.054	261.237	149.903
29	3.497	0.600	269.195	263.612	151.022
30	3.557	0.595	271.696	266.412	152.295
31	3.627	0.594	273.003	267.789	153.076
32	3.560	0.589	273.567	268.387	153.301
33	3.585	0.597	274.310	269.086	153.734
34	3.663	0.600	276.589	271.552	154.884
35	3.671	0.619	277.948	273.042	155.538
36	3.743	0.622	278.802	273.925	155.926

Example 23.2: Retrieving Producer Price Index (PPI) Data for Multiple Time Series

This example shows how to use multiple time series IDs to retrieve Producer Price Index Revision–Current (PC) series data and Producer Price Index–Current Commodity (WP) series data. You can view these data at the following URL:

<https://www.bls.gov/ppi/data.htm>

INSERTIES is an input SAS data set that lists the IDs of the time series to be retrieved from the Producer Price Index (PPI) database. In this example, the input SAS data set is named in the INSET=INSERTIES option. To create the INSERTIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERTIES'. Next, the length of the series ID is specified as 17 bytes prior

to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSET= option specifies the list of series to retrieve using their IDs. The date range of the data starts with 1985, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year to include in the range of data to retrieve. FREQ=M13 specifies the selection of only the annual averaged monthly data, and the FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.2.1](#) shows the PPI data that are retrieved.

```
options validvarname=any;

DATA inseries;
  length seriesid $ 17;
  seriesid = 'WPU013'; output; /* Commodity data for farm livestock, has M13 */
  seriesid = 'WPU014'; output; /* Commodity data for farm poultry, has M13 */
  seriesid = 'PCU212221212221'; output; /* PPI, Gold Ore Mining, has M13 */
  seriesid = 'PCU212312212312S'; output; /* PPI, Limestone Mining, has M13 */
RUN;

title 'Retrieve Producer Price Index Data for Multiple Time Series';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=PPI
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex02
  STARTYEAR='1985'
  ENDYEAR='1994'
  FREQ=M13
  FORMAT=json;

data myPPI;
  set bls.blsex02 ;
run;

proc contents data=myPPI; run;
proc print data=myPPI; run;
```


Output 23.2.1 Retrieve Producer Price Index Annual Average Monthly Data for Multiple Time Series**Retrieve Producer Price Index Data for Multiple Time Series**

Obs	year	period	periodName	PCU212312212312S	WPU013	WPU014
1	1985	M13	Annual	100.4	89.2	117.9
2	1986	M13	Annual	100.7	91.8	129.7
3	1987	M13	Annual	101.1	102.0	101.2
4	1988	M13	Annual	101.8	103.3	121.5
5	1989	M13	Annual	102.0	106.1	128.8
6	1990	M13	Annual	102.8	115.6	118.8
7	1991	M13	Annual	106.1	107.9	111.2
8	1992	M13	Annual	103.5	104.7	112.6
9	1993	M13	Annual	105.4	107.0	122.0
10	1994	M13	Annual	108.1	96.4	124.4

Example 23.3: Retrieving Import/Export Price Index (EI) Data for Multiple Time Series

This example shows how to use multiple time series IDs to retrieve data from the Import/Export Price Index. You can view these data by using the following URL:

<https://www.bls.gov/mxp/data.htm>

INSERTS is an input SAS data set that lists the IDs of the time series to be retrieved from the Import/Export Price Index (EI) database. In this example, the input SAS data set is named in the INSET=INSERTS option. To create the INSERTS data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERTS'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERTS data set for the list of series to retrieve using their IDs. The date range of the data starts with 2019, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year to include in the range of data to retrieve. The FREQ=MONTH option specifies the selection of only the monthly data (M1–M12) in the output data set. The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.3.1](#) shows the EI data that are retrieved.

```
options validvarname=any;

DATA insertseries;
    length seriesid $ 32;
    seriesid = 'EIUIZ3371';    output;
    seriesid = 'EIUIY1113';    output;
    seriesid = 'EIUIV14211';    output;
    seriesid = 'EIUIS231';    output;
```

```

seriesid = 'EIUIRATP08';    output;
seriesid = 'EIUIQ131';     output;
seriesid = 'EIUIP88';      output;
seriesid = 'EIUIH14211';   output;
seriesid = 'EIUID8471';    output;
seriesid = 'EIUIC131';     output;
seriesid = 'EIUCTINDUSTOT'; output;
seriesid = 'EIUCOPRIMTOT'; output;
seriesid = 'EIUCDMEXTOT';  output;
RUN;

title 'Retrieve Import/Export Price Index Data for Multiple Time Series';
libname bls sasebls "%sysget(BLS)";
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=IPP
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex03
  STARTYEAR='2019'
  ENDYEAR='2021'
  FREQ=Month
  FORMAT=json;

data myIPP;
  set bls.blsex03 ;
run;

proc contents data=myIPP; run;
proc print data=myIPP; run;

```

Output 23.3.1 Retrieve Import/Export Index Monthly Data for Multiple Time Series

Retrieve Import/Export Price Index Data for Multiple Time Series

Obs	year	period	periodName	EIUCDMEXTOT	EIUCOPRIMTOT	EIUCTINDUSTOT	EIUC131	EIUID8471
1	2019	M01	January	100.3	98.7	101.7	174.1	36.3
2	2019	M02	February	100.8	98.7	99.8	166.4	36.3
3	2019	M03	March	102.1	98.7	99.4	166.9	36.0
4	2019	M04	April	101.7	98.5	98.7	165.7	35.9
5	2019	M05	May	101.6	98.4	98.2	167.3	35.4
6	2019	M06	June	100.3	98.2	99.6	168.1	35.5
7	2019	M07	July	100.9	98.1	99.6	164.4	35.4
8	2019	M08	August	100.2	98.0	99.6	161.0	34.7
9	2019	M09	September	99.7	97.8	99.3	160.1	34.7
10	2019	M10	October	99.8	97.6	99.7	161.8	34.5
11	2019	M11	November	100.2	97.5	99.4	163.9	34.5
12	2019	M12	December	99.8	97.7	99.3	165.8	34.0
13	2020	M01	January	101.0	97.6	100.1	164.3	33.8
14	2020	M02	February	98.5	97.6	100.5	166.3	34.0
15	2020	M03	March	98.0	97.5	101.6	167.9	34.8
16	2020	M04	April	92.9	97.6	103.5	207.5	34.1
17	2020	M05	May	94.2	97.6	102.9	246.2	34.0
18	2020	M06	June	94.8	97.7	100.7	240.1	33.5
19	2020	M07	July	95.8	97.8	100.9	216.4	33.4
20	2020	M08	August	95.9	97.9	99.4	220.1	33.3
21	2020	M09	September	96.4	98.0	99.9	208.8	32.8
22	2020	M10	October	97.2	97.9	100.4	215.1	32.3

Obs	EIUIH14211	EIUIP88	EIUIQ131	EIUIRATP08	EIUIS231	EIUIV14211	EIUIY1113	EIUIZ3371
1	132.1		102.0		144.2	218.1	119.7	117.9
2	138.1		102.7		144.5	221.1	119.8	117.8
3	135.3		101.8		145.2	200.9	117.9	117.7
4	123.0		101.8		144.3	171.5	115.1	118.0
5	124.1		100.6		144.8	172.4	113.5	118.1
6	125.7		98.9		144.8	192.4	120.0	117.4
7	127.6		97.7		148.6	166.6	117.0	117.3
8	106.7		96.0		146.4	136.3	116.4	117.0
9	120.0		94.4		141.8	187.6	120.1	116.9
10	119.4		92.6		143.2	178.3	118.6	116.5
11	126.8		91.8		146.3	206.9	121.5	116.1
12	126.7		90.8	100.0	147.6	200.6	122.9	116.0
13	113.4		90.9	100.7	149.9	196.1	117.4	115.9
14	113.2		91.0	100.9	190.1	196.7	116.4	115.9
15	107.7		93.9	100.9	153.7	167.3	115.7	115.7
16	88.9		94.4	101.1	157.4	103.8	108.6	115.6
17	81.0		94.3	101.1	158.4	100.8	109.1	115.5
18	91.3		95.0	101.0	149.8	122.4	104.3	115.4
19	99.8		95.8	101.2	150.6	103.1	102.2	115.6
20	93.5		97.5	101.4	150.4	108.1	101.1	115.7
21	96.3		100.4	101.5	144.0	118.0	104.7	115.7
22	103.5		103.3	101.6	137.4	121.0	104.1	115.7

Output 23.3.1 *continued***Retrieve Import/Export Price Index Data for Multiple Time Series**

Obs	year	period	periodName	EIUCDMEXTOT	EIUCOPRIMTOT	EIUCTINDUSTOT	EIUC131	EIUID8471
23	2020	M11	November	98.3	98.0	101.2	228.7	32.1
24	2020	M12	December	99.3	98.3	100.1	243.5	31.8
25	2021	M01	January	102.5	98.7	99.2	240.1	32.6
26	2021	M02	February	105.0	99.0	98.4	239.2	32.2
27	2021	M03	March	109.2	99.4	97.4	232.0	32.2
28	2021	M04	April	112.1	99.7	97.4	242.2	32.4
29	2021	M05	May	120.7	100.2	97.1	256.9	32.7
30	2021	M06	June	121.6	100.5	97.1	264.3	33.0
31	2021	M07	July	121.8	101.0	98.0	248.8	32.9
32	2021	M08	August	122.3	101.4	99.2	247.9	32.6
33	2021	M09	September	122.0	101.7	99.9	256.9	33.1
34	2021	M10	October	124.7	102.2	99.4	267.6	32.9
35	2021	M11	November	125.1	102.5	99.2	288.7	32.9
36	2021	M12	December	121.7	103.0	99.1	276.9	32.8

Obs	EIUIH14211	EIUIP88	EIUIQ131	EIUIRATP08	EIUIS231	EIUIV14211	EIUIY1113	EIUIZ3371
23	125.3		106.6	101.5	144.1	116.7	105.2	115.7
24	129.6	100.0	110.4	102.2	151.4	120.8	107.9	116.0
25	123.3	99.9	117.8	102.8	151.2	107.6	113.1	116.7
26	123.7	100.3	121.3	103.2	163.4	125.8	115.8	116.9
27	132.9	100.7	128.0	103.6	164.0	118.7	120.9	117.2
28	134.1	100.8	135.8	103.9	162.2	121.1	117.5	118.0
29	128.3	100.8	139.0	103.9	149.2	130.2	116.2	118.6
30	135.5	100.7	141.6	103.9	141.9	141.3	116.5	119.2
31	149.0	100.7	144.7	104.0	147.5	142.2	116.5	119.2
32	144.4	100.6	145.2	104.0	241.3	138.5	122.5	119.5
33	140.6	100.8	146.3	104.2	150.2	169.9	123.5	120.5
34	135.5	100.2	146.4	104.0	152.3	167.6	122.5	122.0
35	138.7	100.9	146.4	104.3	200.4	179.4	124.3	122.4
36	153.3	100.9	146.8	104.7	211.6	158.4	124.3	125.4

Example 23.4: Retrieving Consumer Expenditure Survey (CX) Data

This example shows how to retrieve data from the Consumer Expenditure Survey. You can view these data by using the following URL:

<https://www.bls.gov/cex/data.htm>

INSERTIES is an input SAS data set that lists the IDs of the time series to be retrieved from the Consumer Expenditure (CX) database. In this example, the input SAS data set is named in the INSET=INSERTIES option. To create the INSERTIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERTIES'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement.

The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSET= data set for the list of series to retrieve using their IDs. In this example, only one time series, CXUCHGASLILB0101M, is requested. The date range of the data starts with 2010, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2020, to include in the range of data to retrieve. The FREQ=ANNUAL option specifies the selection of yearly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include the net change and percentage change calculation statistics. In this example, only the 12-month net change, CXUCHGASLILB0101M_12, and the 12-month percentage change, CXUCHGASLILB0101M%12, statistics are available. The 1-month net change, 3-month net change, and 6-month net change are missing (not available). In addition, the 1-month percentage change, 3-month percentage change, and 6-month percentage change are also missing (not available). The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.4.1](#) shows the data that are retrieved.

```
options validvarname=any;

DATA inseries;
    length seriesid $ 32;
    seriesid = 'CXUCHGASLILB0101M'; output;
RUN;

title 'Retrieve Consumer Expenditure Survey Annual Data';
libname bls sasebls "%sysget(BLS)"
    USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    SETNAME=CEX
    INSET=inseries
    PAYPATH= "%sysget(BLS)payload"
    OUTJSON=blsex04
    STARTYEAR='2010'
    ENDYEAR='2020'
    FREQ=Annual
    CALCULATIONS=true
    CATALOG=true
    FORMAT=json;

data myCEX;
    set bls.blsex04 ;
run;

proc contents data=myCEX; run;
proc print data=myCEX; run;
```

Output 23.4.1 Net Change in Total Assets and Liabilities by Income Quintiles: All Consumer Units**Retrieve Consumer Expenditure Survey Annual Data**

Obs	year	period	periodName	CXUCHGASLILB0101M	CXUCHGASLILB0101M_12	CXUCHGASLILB0101M%12
1	2010	A01	Annual	-4458	958	-17.7
2	2011	A01	Annual	-1826	2632	-59.0
3	2012	A01	Annual	-5092	-3266	178.9
4	2013	A01	Annual	11404	16496	-324.0
5	2014	A01	Annual	7431	-3973	-34.8
6	2015	A01	Annual	6937	-494	-6.6
7	2016	A01	Annual	8721	1784	25.7
8	2017	A01	Annual	14049	5328	61.1
9	2018	A01	Annual	12679	-1370	-9.8
10	2019	A01	Annual	10971	-1708	-13.5
11	2020	A01	Annual	3584	-7387	-67.3

Example 23.5: Retrieving Current Employment Statistics National Survey (CE) Data

This example shows how to retrieve data from the Current Employment Statistics Survey (CE). You can view these data by using the following URL:

<https://www.bls.gov/ces/data/>

INSERIES is an input SAS data set that lists the IDs of the time series to be retrieved from the Current Employment Statistics (CES) database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series to retrieve using their IDs. In this example, two time series, CES0000000001 and CEU0500000001, are requested, along with any calculations. Both time series have 1-month, 3-month, 6-month, and 12-month net and percentage changes.

The date range of the data starts with 2003, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2004, to include in the range of data to retrieve. The FREQ=MONTH option specifies the selection of monthly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include the net change calculation and percentage change calculation statistics. The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.5.1](#) shows the data that are retrieved. [Output 23.5.2](#) shows the catalog information for the retrieved data.

```

options validvarname=any;

DATA inseries;
  length seriesid $ 32;
  seriesid = 'CES0000000001'; output; /* All employees */
  seriesid = 'CEU0500000001'; output;

RUN;

title 'Retrieve Current Employment Statistics Data';
libname bls sasebls "%sysget(BLS) "
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=CES
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex05
  STARTYEAR='2003'
  ENDYEAR='2004'
  FREQ=Month
  CALCULATIONS=true
  CATALOG=true
  FORMAT=json;

data myCES;
  set bls.blsex05 ;
run;

proc print data=myCES; run;

```

Output 23.5.1 All Seasonally Adjusted Nonfarm Employees and All Not Seasonally Adjusted Private Employees**Retrieve Current Employment Statistics Data**

Obs	year	period	periodName	CES00000000001	CES00000000001_1	CES00000000001_3	CES00000000001_6
1	2003	M01	January	130579	109	-42	-4
2	2003	M02	February	130443	-136	-175	-144
3	2003	M03	March	130233	-210	-237	-265
4	2003	M04	April	130176	-57	-403	-445
5	2003	M05	May	130197	21	-246	-421
6	2003	M06	June	130192	-5	-41	-278
7	2003	M07	July	130183	-9	7	-396
8	2003	M08	August	130153	-30	-44	-290
9	2003	M09	September	130252	99	60	19
10	2003	M10	October	130439	187	256	263

Obs	CES00000000001_12	CES00000000001%1	CES00000000001%3	CES00000000001%6	CES00000000001%12
1	-273	0.1	0.0	0.0	-0.2
2	-289	-0.1	-0.1	-0.1	-0.2
3	-488	-0.2	-0.2	-0.2	-0.4
4	-441	0.0	-0.3	-0.3	-0.3
5	-436	0.0	-0.2	-0.3	-0.3
6	-491	0.0	0.0	-0.2	-0.4
7	-400	0.0	0.0	-0.3	-0.3
8	-434	0.0	0.0	-0.2	-0.3
9	-246	0.1	0.0	0.0	-0.2
10	-182	0.1	0.2	0.2	-0.1

Obs	CEU05000000001	CEU05000000001_1	CEU05000000001_3	CEU05000000001_6	CEU05000000001_12
1	107019	-2266	-2633	-2870	-558
2	107052	33	-2583	-2893	-584
3	107457	405	-1828	-2037	-662
4	108131	674	1112	-1521	-566
5	108906	775	1854	-729	-461
6	109634	728	2177	349	-519
7	109438	-196	1307	2419	-451
8	109533	95	627	2481	-412
9	109262	-271	-372	1805	-232
10	109512	250	74	1381	-140

Obs	CEU05000000001%1	CEU05000000001%3	CEU05000000001%6	CEU05000000001%12
1	-2.1	-2.4	-2.6	-0.5
2	0.0	-2.4	-2.6	-0.5
3	0.4	-1.7	-1.9	-0.6
4	0.6	1.0	-1.4	-0.5
5	0.7	1.7	-0.7	-0.4
6	0.7	2.0	0.3	-0.5
7	-0.2	1.2	2.3	-0.4
8	0.1	0.6	2.3	-0.4
9	-0.2	-0.3	1.7	-0.2
10	0.2	0.1	1.3	-0.1

Output 23.5.1 continued

Retrieve Current Employment Statistics Data

Obs	year	period	periodName	CES0000000001	CES0000000001_1	CES0000000001_3	CES0000000001_6
11	2003	M11	November	130482	43	329	285
12	2003	M12	December	130595	113	343	403
13	2004	M01	January	130765	170	326	582
14	2004	M02	February	130822	57	340	669
15	2004	M03	March	131139	317	544	887
16	2004	M04	April	131410	271	645	971
17	2004	M05	May	131694	284	872	1212
18	2004	M06	June	131789	95	650	1194
19	2004	M07	July	131849	60	439	1084
20	2004	M08	August	131936	87	242	1114

Obs	CES0000000001_12	CES0000000001%1	CES0000000001%3	CES0000000001%6	CES0000000001%12
11	-136	0.0	0.3	0.2	-0.1
12	125	0.1	0.3	0.3	0.1
13	186	0.1	0.2	0.4	0.1
14	379	0.0	0.3	0.5	0.3
15	906	0.2	0.4	0.7	0.7
16	1234	0.2	0.5	0.7	0.9
17	1497	0.2	0.7	0.9	1.1
18	1597	0.1	0.5	0.9	1.2
19	1666	0.0	0.3	0.8	1.3
20	1783	0.1	0.2	0.9	1.4

Obs	CEU0500000001	CEU0500000001_1	CEU0500000001_3	CEU0500000001_6	CEU0500000001_12
11	109569	57	36	663	-66
12	109454	-115	192	-180	169
13	107266	-2188	-2246	-2172	247
14	107474	208	-2095	-2059	422
15	108369	895	-1085	-893	912
16	109468	1099	2202	-44	1337
17	110392	924	2918	823	1486
18	111280	888	2911	1826	1646
19	111231	-49	1763	3965	1793
20	111232	1	840	3758	1699

Obs	CEU0500000001%1	CEU0500000001%3	CEU0500000001%6	CEU0500000001%12
11	0.1	0.0	0.6	-0.1
12	-0.1	0.2	-0.2	0.2
13	-2.0	-2.1	-2.0	0.2
14	0.2	-1.9	-1.9	0.4
15	0.8	-1.0	-0.8	0.8
16	1.0	2.1	0.0	1.2
17	0.8	2.7	0.8	1.4
18	0.8	2.7	1.7	1.5
19	0.0	1.6	3.7	1.6
20	0.0	0.8	3.5	1.6

Output 23.5.1 *continued***Retrieve Current Employment Statistics Data**

Obs	year	period	periodName	CES0000000001	CES0000000001_1	CES0000000001_3	CES0000000001_6
21	2004	M09	September	132082	146	293	943
22	2004	M10	October	132433	351	584	1023
23	2004	M11	November	132509	76	573	815
24	2004	M12	December	132633	124	551	844

Obs	CES0000000001_12	CES0000000001%1	CES0000000001%3	CES0000000001%6	CES0000000001%12
21	1830	0.1	0.2	0.7	1.4
22	1994	0.3	0.4	0.8	1.5
23	2027	0.1	0.4	0.6	1.6
24	2038	0.1	0.4	0.6	1.6

Obs	CEU0500000001	CEU0500000001_1	CEU0500000001_3	CEU0500000001_6	CEU0500000001_12
21	110901	-331	-379	2532	1639
22	111357	456	126	1889	1845
23	111457	100	225	1065	1888
24	111349	-108	448	69	1895

Obs	CEU0500000001%1	CEU0500000001%3	CEU0500000001%6	CEU0500000001%12
21	-0.3	-0.3	2.3	1.5
22	0.4	0.1	1.7	1.7
23	0.1	0.2	1.0	1.7
24	-0.1	0.4	0.1	1.7

The following statements print the catalog information shown in [Output 23.5.2](#):

```
libname blscat "%sysget (BLS) ";

data myCAT;
  set blscat.blsex05_catalog;
run;

proc print data=myCAT; run;
```

Output 23.5.2 Catalog for Series CES0000000001 and CEU0500000001**Retrieve Current Employment Statistics Data**

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality	survey_name
1	1	1	All employees, thousands, total nonfarm, seasonally adjusted	CES0000000001	Seasonally Adjusted	Employment, Hours, and Earnings from the Current Employment Statistics survey (National)
2	2	2	All employees, thousands, total private, not seasonally adjusted	CEU0500000001	Not Seasonally Adjusted	Employment, Hours, and Earnings from the Current Employment Statistics survey (National)

Obs	survey_abbreviation	measure_data_type	commerce_industry	commerce_sector
1	CE	ALL EMPLOYEES, THOUSANDS	Total nonfarm	Total nonfarm
2	CE	ALL EMPLOYEES, THOUSANDS	Total private	Total private

Example 23.6: Retrieving State and Metro Area Current Employment (SM) Data

This example shows how to retrieve data from the State and Metro Area Current Employment Survey (SM). You can view these data by using the following URL:

<https://www.bls.gov/sae/data/>

INSERIES is an input SAS data set that lists the IDs of the time series to be retrieved from the State and Metro Area Current Employment Statistics (SAE) database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series to retrieve using their IDs. In this example, two time series, SMU78000009092000001 and SMS39496600000000001, are requested, along with any calculations. Both time series have 1-month, 3-month, 6-month, and 12-month net and percentage changes.

The date range of the data starts with 2021, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2022, to include in the range of data to retrieve. The FREQ=MONTH option specifies the selection of monthly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include the net change calculation and percentage change calculation statistics. The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.6.1](#) shows the first seven observations of data that are retrieved. [Output 23.6.2](#) shows the catalog information for the retrieved data.

```

options validvarname=any;

DATA inseries;
    length seriesid $ 32;
    seriesid = 'SMU78000009092000001 '; output;
    seriesid = 'SMS39496600000000001'; output;
RUN;

title 'Retrieve State and Metro Area Current Employment Statistics Data';
libname bls sasebls "%sysget(BLS) "
    USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    SETNAME=SAE
    INSET=inseries
    PAYPATH= "%sysget(BLS)payload"
    OUTJSON=blsex06
    STARTYEAR='2021'
    ENDYEAR='2022'
    FREQ=Month
    CALCULATIONS=true
    CATALOG=true
    FORMAT=json;

data mySAE;
    set bls.blsex06 ;
run;

proc contents data=mySAE; run;
proc print data=mySAE(obs=7); run;

```

Output 23.6.1 State and Area Employment, Hours, and Earnings: Total Nonfarm and State Government**Retrieve State and Metro Area Current Employment Statistics Data**

Obs	year	period	periodName	SMS394966000000000001	SMS394966000000000001_1	SMS394966000000000001_3
1	2021	M01	January	199.7	2.1	0.1
2	2021	M02	February	199.2	-0.5	0.3
3	2021	M03	March	200.9	1.7	3.3
4	2021	M04	April	201.7	0.8	2.0
5	2021	M05	May	201.4	-0.3	2.2
6	2021	M06	June	202.1	0.7	1.2
7	2021	M07	July	203.3	1.2	1.6

Obs	SMS394966000000000001_6	SMS394966000000000001_12	SMS394966000000000001%1	SMS394966000000000001%3
1	3.2	-16.0	1.1	0.1
2	1.3	-16.0	-0.3	0.2
3	1.6	-11.4	0.9	1.7
4	2.1	28.1	0.4	1.0
5	2.5	19.5	-0.1	1.1
6	4.5	9.5	0.3	0.6
7	3.6	6.8	0.6	0.8

Obs	SMS394966000000000001%6	SMS394966000000000001%12	SMU78000009092000001	SMU78000009092000001_1
1	1.6	-7.4	9.4	-0.1
2	0.7	-7.4	9.6	0.2
3	0.8	-5.4	9.6	0.0
4	1.1	16.2	9.7	0.1
5	1.3	10.7	9.6	-0.1
6	2.3	4.9	9.7	0.1
7	1.8	3.5	10.1	0.4

Obs	SMU78000009092000001_3	SMU78000009092000001_6	SMU78000009092000001_12	SMU78000009092000001%1
1	-0.8	-1.0	-0.1	-1.1
2	-0.6	-0.4	-0.1	2.1
3	0.1	-0.4	-0.5	0.0
4	0.3	-0.5	-0.6	1.0
5	0.0	-0.6	-0.6	-1.0
6	0.1	0.2	0.0	1.0
7	0.4	0.7	-0.3	4.1

Obs	SMU78000009092000001%3	SMU78000009092000001%6	SMU78000009092000001%12
1	-7.8	-9.6	-1.1
2	-5.9	-4.0	-1.0
3	1.1	-4.0	-5.0
4	3.2	-4.9	-5.8
5	0.0	-5.9	-5.9
6	1.0	2.1	0.0
7	4.1	7.4	-2.9

The following statements print the catalog information shown in [Output 23.6.2](#):

```
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

data myCAT;
    set blscat.blsex06_catalog;
run;
```

```
proc print data=blscat.blsex06_catalog; run;
```

Output 23.6.2 Catalog for State and Area Employment, Hours, and Earnings: Total Nonfarm and State Government

Retrieve State and Metro Area Current Employment Statistics Data

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality	survey_name
1	1	1	Employed and Office of Employment and Unemployment Statistics : Total Nonfarm - Total Nonfarm	SMS39496600000000001	Seasonally Adjusted	State and Area Employment, Hours, and Earnings
2	2	2	Employed and Office of Employment and Unemployment Statistics : Government - State Government	SMU78000009092000001	Not Seasonally Adjusted	State and Area Employment, Hours, and Earnings

Obs	survey_abbreviation	measure_data_type	commerce_industry	commerce_sector	area
1	SM	All Employees, In Thousands	Total Nonfarm	Total Nonfarm	Youngstown-Warren-Boardman, OH-PA
2	SM	All Employees, In Thousands	State Government	Government	Virgin Islands

Example 23.7: Retrieving Labor Force Current Population Statistics (CPS) Data

This example shows how to retrieve data from the Weekly Hours and Earnings Survey, LE, and the Labor Force Statistics Survey (includes the national unemployment rate), LN, both from the Current Population Statistics program data. You can view these data by using the following URL:

<https://www.bls.gov/cps/>

INSERIES is an input SAS data set that lists the IDs of the time series to be retrieved from the Current Population Statistics database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option

and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series to retrieve using their IDs. In this example, four time series are requested, along with any calculations.

The date range of the data starts with 2021, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2022, to include in the range of data to retrieve. The FREQ=QUARTERLY option specifies the selection of quarterly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include the net change calculation and percentage change calculation statistics. The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.7.1](#) shows the data that are retrieved. [Output 23.7.2](#) shows the catalog information for the retrieved data.

```
options validvarname=any;
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

DATA inseries;
  length seriesid $ 32;
  seriesid = 'LNS11000000Q'; output;
  seriesid = 'LNS11000012Q'; output;
  seriesid = 'LES1252881500'; output;
  seriesid = 'LES1252882800'; output;
RUN;

title 'Retrieve Labor Force Current Population Statistics Data';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=CPS
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex07
  STARTYEAR='2021'
  ENDYEAR='2022'
  FREQ=Quarterly
  CALCULATIONS=true
  CATALOG=true
  FORMAT=json;

data myCPS;
  set bls.blsex07 ;
run;

proc contents data=myCPS; run;
proc print data=myCPS; run;
```

Output 23.7.1 Weekly Earnings and Civilian Labor Force Level: Current Population Statistics**Retrieve Labor Force Current Population Statistics Data**

Obs	year	period	periodName	LES1252881500	LES1252881500_3	LES1252881500_6	LES1252881500_12
1	2021	Q01	1st Quarter	983	0	-11	32
2	2021	Q02	2nd Quarter	996	13	13	-12
3	2021	Q03	3rd Quarter	1001	5	18	7
4	2021	Q04	4th Quarter	1010	9	14	27
5	2022	Q01	1st Quarter	1032	22	31	49
6	2022	Q02	2nd Quarter	1048	16	38	52

Obs	LES1252881500%3	LES1252881500%6	LES1252881500%12	LES1252882800	LES1252882800_3	LES1252882800_6
1	0.0	-1.1	3.4	340	-2	-7
2	1.3	1.3	-1.2	337	-3	-5
3	0.5	1.8	0.7	335	-2	-5
4	0.9	1.4	2.7	333	-2	-4
5	2.2	3.1	5.0	329	-4	-6
6	1.6	3.8	5.2	326	-3	-7

Obs	LES1252882800_12	LES1252882800%3	LES1252882800%6	LES1252882800%12	LNS11000000Q	LNS11000000Q_3
1	10	-0.6	-2.0	3.0	160225	-571
2	-22	-0.9	-1.5	-6.1	160947	722
3	-12	-0.6	-1.5	-3.5	161521	574
4	-9	-0.6	-1.2	-2.6	162155	634
5	-11	-1.2	-1.8	-3.2	163932	1777
6	-11	-0.9	-2.1	-3.3	164077	145

Obs	LNS11000000Q_6	LNS11000000Q_12	LNS11000000Q%3	LNS11000000Q%6	LNS11000000Q%12	LNS11000012Q
1	-124	-3588	-0.4	-0.1	-2.2	5938
2	151	2855	0.5	0.1	1.8	5993
3	1296	1172	0.4	0.8	0.7	5949
4	1208	1359	0.4	0.8	0.8	5974
5	2411	3707	1.1	1.5	2.3	6215
6	1922	3130	0.1	1.2	1.9	6243

Obs	LNS11000012Q_3	LNS11000012Q_6	LNS11000012Q_12	LNS11000012Q%3	LNS11000012Q%6	LNS11000012Q%12
1	-12	268	-77	-0.2	4.7	-1.3
2	55	43	666	0.9	0.7	12.5
3	-44	11	279	-0.7	0.2	4.9
4	25	-19	24	0.4	-0.3	0.4
5	241	266	277	4.0	4.5	4.7
6	28	269	250	0.5	4.5	4.2

Output 23.7.1 *continued***Retrieve Labor Force Current Population Statistics Data**

Obs	year	period	periodName	LES1252881500	LES1252881500_3	LES1252881500_6	LES1252881500_12
7	2022	Q03	3rd Quarter	1070	22	38	69
8	2022	Q04	4th Quarter	1084	14	36	74

Obs	LES1252881500%3	LES1252881500%6	LES1252881500%12	LES1252882800	LES1252882800_3	LES1252882800_6
7	2.1	3.7	6.9	327	1	-2
8	1.3	3.4	7.3	327	0	1

Obs	LES1252882800_12	LES1252882800%3	LES1252882800%6	LES1252882800%12	LNS11000000Q	LNS11000000Q_3
7	-8	0.3	-0.6	-2.4	164441	364
8	-6	0.0	0.3	-1.8	164713	272

Obs	LNS11000000Q_6	LNS11000000Q_12	LNS11000000Q%3	LNS11000000Q%6	LNS11000000Q%12	LNS11000012Q
7	509	2920	0.2	0.3	1.8	6280
8	636	2558	0.2	0.4	1.6	6367

Obs	LNS11000012Q_3	LNS11000012Q_6	LNS11000012Q_12	LNS11000012Q%3	LNS11000012Q%6	LNS11000012Q%12
7	37	65	331	0.6	1.0	5.6
8	87	124	393	1.4	2.0	6.6

The following statements print the catalog information shown in [Output 23.7.2](#):

```
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

data myCAT;
  set blscat.blsex07_catalog;
run;

proc print data=blscat.blsex07_catalog; run;
```

Output 23.7.2 Catalog for Weekly Earnings and Civilian Labor Force Level: Current Population Statistics**Retrieve Labor Force Current Population Statistics Data**

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality
1	1	1	(Seas)- Median usual weekly earnings (second quartile), Employed full time, Wage and salary workers	LES1252881500	Seasonally Adjusted
2	2	2	(Seas)- Constant (1982-84) dollar adjusted to CPI-U- Median usual weekly earnings, Employed full time, Wage and salary workers, Women	LES1252882800	Seasonally Adjusted

Obs	survey_name	survey_abbreviation	measure_data_type	commerce_industry	occupation
1	Weekly and hourly earnings data from the Current Population Survey	LE	Median usual weekly earnings - in current dollars (second quartile)	All Industries	All Occupations
2	Weekly and hourly earnings data from the Current Population Survey	LE	Median usual weekly earnings - in constant (1982-84) dollars	All Industries	All Occupations

Obs	occupation_work_class	cps_labor_force_status	demographic_age	demographic_ethnic_origin	demographic_race
1	Wage and salary workers, excluding incorporated self employed	Employed full time	16 years and over	All Origins	All Races
2	Wage and salary workers, excluding incorporated self employed	Employed full time	16 years and over	All Origins	All Races

Obs	demographic_gender	demographic_education
1	Both Sexes	All educational levels
2	Women	All educational levels

Output 23.7.2 *continued***Retrieve Labor Force Current Population Statistics Data**

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality
3	3	3	(Seas) Civilian Labor Force Level	LNS1100000Q	Seasonally Adjusted
4	4	4	(Seas) Civilian Labor Force Level - 16-19 yrs.	LNS11000012Q	Seasonally Adjusted

Obs	survey_name	survey_abbreviation	measure_data_type	commerce_industry	occupation
3	Labor Force Statistics from the Current Population Survey	LN	Number in thousands	All Industries	All Occupations
4	Labor Force Statistics from the Current Population Survey	LN	Number in thousands	All Industries	All Occupations

Obs	occupation_work_class	cps_labor_force_status	demographic_age	demographic_ethnic_origin	demographic_race
3		Civilian labor force	16 years and over	All Origins	All Races
4		Civilian labor force	16 to 19 years	All Origins	All Races

Obs	demographic_gender	demographic_education
3	Both Sexes	All educational levels
4	Both Sexes	All educational levels

Example 23.8: Retrieving Job Openings and Labor Turnover Statistics (JOLT) Data

This example shows how to retrieve data from the Job Openings and Labor Turnover Surveys, JT and JL. You can view these data by using the following URL:

<https://www.bls.gov/jlt>

INSERTS is an input SAS data set that lists the IDs of the time series to be retrieved from the Job Openings and Labor Turnover Statistics database. In this example, the input SAS data set is named in the INSET=INSERTS option. To create the INSERTS data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERTS'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON=

option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series to retrieve using their IDs. In this example, four time series are requested, along with any calculations.

The date range of the data starts with 2002, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2003, to include in the range of data to retrieve. The FREQ=MONTH option specifies the selection of monthly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include the net change calculation and percentage change calculation statistics. There is no catalog for the JOLT data set, so the description of each statistical time series can be obtained by the seriesid format described at the following URL:

<https://www.bls.gov/help/hlpforma.htm#jt>

These descriptions can be placed in the label for each time series as shown in the label statements of the myJOLT DATA STEP.

The FORMAT= option specifies that the data are to be retrieved using the JSON format. For brevity, only the first four observations are shown in [Output 23.8.1](#).

```
options validvarname=any;

DATA inseries;
  length seriesid $ 32;
  seriesid = 'JLU000000MWHIL'; output; /* Total Nonfarm Hires (SIC based) */
  seriesid = 'JLU91000000TSR'; output; /* Federal Total Separations (SIC based)*/
  seriesid = 'JTS000000000000000HIL'; output; /* Total Nonfarm Hires */
  seriesid = 'JTU929000000000000LDL'; output; /* Govt. Layoffs and Discharges */
RUN;

title 'Retrieve Job Openings and Labor Turnover Statistics Data';
libname bls sasebls "%sysget(BLS) "
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=JOLT
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex08
  STARTYEAR='2002'
  ENDYEAR='2003'
  FREQ=Month
  CALCULATIONS=true
  FORMAT=json;

data myJOLT;
  set bls.blsex08 ;
  label JLU000000MWHIL='Total Nonfarm Hires (SIC based)';
  label JLU91000000TSR='Federal Total Separations (SIC based)';
  label JTS000000000000000HIL='Total Nonfarm Hires';
  label JTU929000000000000LDL='State and Local Government Layoffs and Discharges';
run;
```

```
proc contents data=myJOLT; run;
proc print data=myJOLT(obs=4); run;
```

Output 23.8.1 Job Openings and Labor Turnover Statistics

Retrieve Job Openings and Labor Turnover Statistics Data

Obs	year	period	periodName	JLU000000MWHIL	JLU000000MWHIL_1	JLU000000MWHIL_3	JLU000000MWHIL_6
1	2002	M01	January	738	18	-345	-343
2	2002	M02	February	744	6	-38	-534

Obs	JLU000000MWHIL_12	JLU000000MWHIL%1	JLU000000MWHIL%3	JLU000000MWHIL%6	JLU000000MWHIL%12
1	-294	2.5	-31.9	-31.7	-28.5
2	-162	0.8	-4.9	-41.8	-17.9

Obs	JLU91000000TSR	JLU91000000TSR_1	JLU91000000TSR_3	JLU91000000TSR_6	JLU91000000TSR_12
1	1.5	-0.2	0.0	-0.1	-0.4
2	1.3	-0.2	0.2	-0.4	-0.1

Obs	JLU91000000TSR%1	JLU91000000TSR%3	JLU91000000TSR%6	JLU91000000TSR%12	JTS0000000000000000HIL
1	-11.8	0.0	-6.3	-21.1	4855
2	-13.3	18.2	-23.5	-7.1	4801

Obs	JTS0000000000000000HIL_1	JTS0000000000000000HIL_3	JTS0000000000000000HIL_6	JTS0000000000000000HIL_12
1	26	-223	-389	-867
2	-54	-113	-422	-502

Obs	JTS0000000000000000HIL%1	JTS0000000000000000HIL%3	JTS0000000000000000HIL%6	JTS0000000000000000HIL%12
1	0.5	-4.4	-7.4	-15.2
2	-1.1	-2.3	-8.1	-9.5

Obs	JTU9290000000000000LDL	JTU9290000000000000LDL_1	JTU9290000000000000LDL_3	JTU9290000000000000LDL_6
1	31	-3	-36	1
2	29	-2	-5	-114

Obs	JTU9290000000000000LDL_12	JTU9290000000000000LDL%1	JTU9290000000000000LDL%3	JTU9290000000000000LDL%6
1	-2	-8.8	-53.7	3.3
2	2	-6.5	-14.7	-79.7

Obs	JTU9290000000000000LDL%12
1	-6.1
2	7.4

Output 23.8.1 *continued***Retrieve Job Openings and Labor Turnover Statistics Data**

Obs	year	period	periodName	JLU000000MWHIL	JLU000000MWHIL_1	JLU000000MWHIL_3	JLU000000MWHIL_6
3	2002	M03	March	832	88	112	-462
4	2002	M04	April	1173	341	435	90
Obs	JLU000000MWHIL_12	JLU000000MWHIL%1	JLU000000MWHIL%3	JLU000000MWHIL%6	JLU000000MWHIL%12		
3	-270	11.8	15.6	-35.7	-24.5		
4	-76	41.0	58.9	8.3	-6.1		
Obs	JLU91000000TSR	JLU91000000TSR_1	JLU91000000TSR_3	JLU91000000TSR_6	JLU91000000TSR_12		
3	1.3	0.0	-0.4	-0.4	0.4		
4	1.2	-0.1	-0.3	-0.3	0.1		
Obs	JLU91000000TSR%1	JLU91000000TSR%3	JLU91000000TSR%6	JLU91000000TSR%12	JTS000000000000000HIL		
3	0.0	-23.5	-23.5	44.4	4706		
4	-7.7	-20.0	-20.0	9.1	4928		
Obs	JTS000000000000000HIL_1	JTS000000000000000HIL_3	JTS000000000000000HIL_6	JTS000000000000000HIL_12			
3	-95	-123	-248	-822			
4	222	73	-150	-276			
Obs	JTS000000000000000HIL%1	JTS000000000000000HIL%3	JTS000000000000000HIL%6	JTS000000000000000HIL%12			
3	-2.0	-2.5	-5.0	-14.9			
4	4.7	1.5	-3.0	-5.3			
Obs	JTU929000000000000LDL	JTU929000000000000LDL_1	JTU929000000000000LDL_3	JTU929000000000000LDL_6			
3	21	-8	-13	-83			
4	43	22	12	-24			
Obs	JTU929000000000000LDL_12	JTU929000000000000LDL%1	JTU929000000000000LDL%3	JTU929000000000000LDL%6			
3	-5	-27.6	-38.2	-79.8			
4	-5	104.8	38.7	-35.8			
Obs	JTU929000000000000LDL%12						
3	-19.2						
4	-10.4						

Example 23.9: Retrieving Quarterly Census of Employment and Wages (QCEW) Data

This example shows how to retrieve data from the Quarterly Census of Employment and Wages surveys. You can view these data by using the following URL:

<https://www.bls.gov/cew/data-overview.htm>

INSERIES is an input SAS data set that lists the IDs of the time series to be retrieved from the Quarterly Census of Employment and Wages (QCEW) database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series to retrieve using their IDs. In this example, four time series are requested, along with any calculations.

The date range of the data starts with 2019, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2020, to include in the range of data to retrieve. The FREQ=MONTH option specifies the selection of monthly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include the net change calculation and percentage change calculation statistics. No catalog is available for the selected QCEW data in this example, so the description of each time series can be obtained by using the series ID format described at the following URL:

<https://www.bls.gov/help/hlpforma.htm#EN>

These descriptions can be placed in the label for each time series, as shown in the LABEL statements in the myQCEW DATA step.

The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.9.1](#) shows the data that are retrieved.

```
options validvarname=any;

DATA inseries;
  length seriesid $ 32;
  seriesid = 'ENU0603710010'; output; /* Los Angeles County */
  seriesid = 'ENU0607310010'; output; /* San Diego County */
RUN;

title 'Retrieve State and County Employment and Wages Data';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=QCEW
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex09
```

```
STARTYEAR='2019'
ENDYEAR='2020'
FREQ=Month
CALCULATIONS=true
FORMAT=json;

data myQCEW;
  set bls.blsex09;
  label ENU0603710010=
    'All employees, industries, and establishment sizes in Los Angeles County, NSA';
  label ENU0607310010=
    'All employees, industries, and establishment sizes in San Diego County, NSA';
run;

proc contents data=myQCEW; run;
proc print data=myQCEW; run;
```


Output 23.9.1 State and County Employment and Wages, Los Angeles and San Diego**Retrieve State and County Employment and Wages Data**

Obs	year	period	periodName	ENU0603710010	ENU0603710010_1	ENU0603710010_3	ENU0603710010_6
1	2019	M01	January	4441945	-84249	-42840	47637
2	2019	M02	February	4503959	62014	-24444	74733
3	2019	M03	March	4488329	-15630	-37865	37896
4	2019	M04	April	4476946	-11383	35001	-7839
5	2019	M05	May	4504394	27448	435	-24009
6	2019	M06	June	4501170	-3224	12841	-25024
7	2019	M07	July	4453145	-48025	-23801	11200
8	2019	M08	August	4485820	32675	-18574	-18139
9	2019	M09	September	4512698	26878	11528	24369
10	2019	M10	October	4547597	34899	94452	70651

Obs	ENU0603710010_12	ENU0603710010%1	ENU0603710010%3	ENU0603710010%6	ENU0603710010%12
1	58390	-1.9	-1.0	1.1	1.3
2	66479	1.4	-0.5	1.7	1.5
3	66546	-0.3	-0.8	0.9	1.5
4	57017	-0.3	0.8	-0.2	1.3
5	72814	0.6	0.0	-0.5	1.6
6	59703	-0.1	0.3	-0.6	1.3
7	58837	-1.1	-0.5	0.3	1.3
8	56594	0.7	-0.4	-0.4	1.3
9	62265	0.6	0.3	0.5	1.4
10	62812	0.8	2.1	1.6	1.4

Obs	ENU0607310010	ENU0607310010_1	ENU0607310010_3	ENU0607310010_6	ENU0607310010_12
1	1461494	-24183	-15211	3363	23556
2	1470634	9140	-14836	9012	21659
3	1474284	3650	-11393	8050	22357
4	1482234	7950	20740	5529	21656
5	1487525	5291	16891	2055	23090
6	1492516	4991	18232	6839	20705
7	1479095	-13421	-3139	17601	20964
8	1488846	9751	1321	18212	27224
9	1491789	2943	-727	17505	25555
10	1503857	12068	24762	21623	27152

Obs	ENU0607310010%1	ENU0607310010%3	ENU0607310010%6	ENU0607310010%12
1	-1.6	-1.0	0.2	1.6
2	0.6	-1.0	0.6	1.5
3	0.2	-0.8	0.5	1.5
4	0.5	1.4	0.4	1.5
5	0.4	1.1	0.1	1.6
6	0.3	1.2	0.5	1.4
7	-0.9	-0.2	1.2	1.4
8	0.7	0.1	1.2	1.9
9	0.2	0.0	1.2	1.7
10	0.8	1.7	1.5	1.8

Output 23.9.1 *continued*

Retrieve State and County Employment and Wages Data

Obs	year	period	periodName	ENU0603710010	ENU0603710010_1	ENU0603710010_3	ENU0603710010_6
11	2019	M11	November	4608275	60678	122455	103881
12	2019	M12	December	4594584	-13691	81886	93414
13	2020	M01	January	4509613	-84971	-37984	56468
14	2020	M02	February	4577042	67429	-31233	91222
15	2020	M03	March	4483235	-93807	-111349	-29463
16	2020	M04	April	3768970	-714265	-740643	-778627
17	2020	M05	May	3833794	64824	-743248	-774481
18	2020	M06	June	3935879	102085	-547356	-658705
19	2020	M07	July	3911445	-24434	142475	-598168
20	2020	M08	August	3930529	19084	96735	-646513

Obs	ENU0603710010_12	ENU0603710010%1	ENU0603710010%3	ENU0603710010%6	ENU0603710010%12
11	79872	1.3	2.7	2.3	1.8
12	68390	-0.3	1.8	2.1	1.5
13	67668	-1.8	-0.8	1.3	1.5
14	73083	1.5	-0.7	2.0	1.6
15	-5094	-2.0	-2.4	-0.7	-0.1
16	-707976	-15.9	-16.4	-17.1	-15.8
17	-670600	1.7	-16.2	-16.8	-14.9
18	-565291	2.7	-12.2	-14.3	-12.6
19	-541700	-0.6	3.8	-13.3	-12.2
20	-555291	0.5	2.5	-14.1	-12.4

Obs	ENU0607310010	ENU0607310010_1	ENU0607310010_3	ENU0607310010_6	ENU0607310010_12
11	1515570	11713	26724	28045	30100
12	1511523	-4047	19734	19007	25846
13	1487837	-23686	-16020	8742	26343
14	1500138	12301	-15432	11292	29504
15	1484848	-15290	-26675	-6941	10564
16	1248648	-236200	-239189	-255209	-233586
17	1262659	14011	-237479	-252911	-224866
18	1321639	58980	-163209	-189884	-170877
19	1313914	-7725	65266	-173923	-165181
20	1329040	15126	66381	-171098	-159806

Obs	ENU0607310010%1	ENU0607310010%3	ENU0607310010%6	ENU0607310010%12
11	0.8	1.8	1.9	2.0
12	-0.3	1.3	1.3	1.7
13	-1.6	-1.1	0.6	1.8
14	0.8	-1.0	0.8	2.0
15	-1.0	-1.8	-0.5	0.7
16	-15.9	-16.1	-17.0	-15.8
17	1.1	-15.8	-16.7	-15.1
18	4.7	-11.0	-12.6	-11.4
19	-0.6	5.2	-11.7	-11.2
20	1.2	5.3	-11.4	-10.7

Output 23.9.1 *continued***Retrieve State and County Employment and Wages Data**

Obs	year	period	periodName	ENU0603710010	ENU0603710010_1	ENU0603710010_3	ENU0603710010_6
21	2020	M09	September	3991837	61308	55958	-491398
22	2020	M10	October	4098577	106740	187132	329607
23	2020	M11	November	4155559	56982	225030	321765
24	2020	M12	December	4115123	-40436	123286	179244

Obs	ENU0603710010_12	ENU0603710010%1	ENU0603710010%3	ENU0603710010%6	ENU0603710010%12
21	-520861	1.6	1.4	-11.0	-11.5
22	-449020	2.7	4.8	8.7	-9.9
23	-452716	1.4	5.7	8.4	-9.8
24	-479461	-1.0	3.1	4.6	-10.4

Obs	ENU0607310010	ENU0607310010_1	ENU0607310010_3	ENU0607310010_6	ENU0607310010_12
21	1349606	20566	27967	-135242	-142183
22	1378764	29158	64850	130116	-125093
23	1387139	8375	58099	124480	-128431
24	1370022	-17117	20416	48383	-141501

Obs	ENU0607310010%1	ENU0607310010%3	ENU0607310010%6	ENU0607310010%12
21	1.5	2.1	-9.1	-9.5
22	2.2	4.9	10.4	-8.3
23	0.6	4.4	9.9	-8.5
24	-1.2	1.5	3.7	-9.4

Example 23.10: Retrieving Business Employment Dynamics Statistics (BDM) Data

This example shows how to retrieve data from the Business Employment Dynamics Statistics survey. You can view these data by using the following URL:

<https://www.bls.gov/bdm/data.htm>

INSERIES is an input SAS data set that lists the IDs of the time series to be retrieved from the Business Employment Dynamics Statistics (BDM) database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series to retrieve using their IDs. In this example, four time series are requested, along with any calculations.

The date range of the data starts with 2019, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2020, to include in the range of data to retrieve. The FREQ=QUARTERLY (or FREQ=Q) option specifies the selection of quarterly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include the net change calculation and percentage change calculation statistics. The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.10.1](#) shows the data that are retrieved. [Output 23.10.2](#) shows the catalog information for the retrieved data.

```
options validvarname=any;
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

DATA inseries;
  length seriesid $ 32;
  seriesid = 'BDS00000000000100030110004LQ5'; output;
  seriesid = 'BDS00000000000200050110001LQ5'; output;
  seriesid = 'BDS00000000000100000110001LQ5'; output;
  seriesid = 'BDS00000000000200000110001LQ5'; output;
RUN;

title 'Retrieve Business Employment Dynamics Statistics Data';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=BDM
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex10
  STARTYEAR='2019'
  ENDYEAR='2020'
  FREQ=Quarterly
  CALCULATIONS=true
  CATALOG=true
  FORMAT=json;

data myBDM;
  set bls.blsex10;
run;

proc contents data=myBDM; run;
proc print data=myBDM; run;
```

Output 23.10.1 Business Employment Dynamics Statistics**Retrieve Business Employment Dynamics Statistics Data**

Obs	year	period	periodName	BDS0000000000100000110001LQ5	BDS0000000000100000110001LQ5_3
1	2019	Q01	1st Quarter	1356	-40
2	2019	Q02	2nd Quarter	1378	22
3	2019	Q03	3rd Quarter	1254	-124

Obs	BDS0000000000100000110001LQ5_6	BDS0000000000100000110001LQ5_12	BDS0000000000100000110001LQ5%3
1	39	-56	-2.9
2	-18	-38	1.6
3	-102	-63	-9.0

Obs	BDS0000000000100000110001LQ5%6	BDS0000000000100000110001LQ5%12	BDS0000000000100030110004LQ5
1	3.0	-4.0	385
2	-1.3	-2.7	424
3	-7.5	-4.8	437

Obs	BDS0000000000100030110004LQ5_3	BDS0000000000100030110004LQ5_6	BDS0000000000100030110004LQ5_12
1	3	-21	23
2	39	42	48
3	13	52	31

Obs	BDS0000000000100030110004LQ5%3	BDS0000000000100030110004LQ5%6	BDS0000000000100030110004LQ5%12
1	0.8	-5.2	6.4
2	10.1	11.0	12.8
3	3.1	13.5	7.6

Obs	BDS0000000000200000110001LQ5	BDS0000000000200000110001LQ5_3	BDS0000000000200000110001LQ5_6
1	6108	-220	-75
2	6275	167	-53
3	6144	-131	36

Obs	BDS0000000000200000110001LQ5_12	BDS0000000000200000110001LQ5%3	BDS0000000000200000110001LQ5%6
1	23	-3.5	-1.2
2	18	2.7	-0.8
3	-39	-2.1	0.6

Obs	BDS0000000000200000110001LQ5%12	BDS0000000000200050110001LQ5	BDS0000000000200050110001LQ5_3
1	0.4	164	20
2	0.3	164	0
3	-0.6	161	-3

Obs	BDS0000000000200050110001LQ5_6	BDS0000000000200050110001LQ5_12	BDS0000000000200050110001LQ5%3
1	17	6	13.9
2	20	4	0.0
3	-3	14	-1.8

Obs	BDS0000000000200050110001LQ5%6	BDS0000000000200050110001LQ5%12
1	11.6	3.8
2	13.9	2.5
3	-1.8	9.5

Output 23.10.1 continued

Retrieve Business Employment Dynamics Statistics Data

Obs	year	period	periodName	BDS0000000000100000110001LQ5	BDS0000000000100000110001LQ5_3
4	2019	Q04	4th Quarter	1310	56
5	2020	Q01	1st Quarter	1296	-14
6	2020	Q02	2nd Quarter	1045	-251

Obs	BDS0000000000100000110001LQ5_6	BDS0000000000100000110001LQ5_12	BDS0000000000100000110001LQ5%3
4	-68	-86	4.5
5	42	-60	-1.1
6	-265	-333	-19.4

Obs	BDS0000000000100000110001LQ5%6	BDS0000000000100000110001LQ5%12	BDS0000000000100030110004LQ5
4	-4.9	-6.2	410
5	3.3	-4.4	426
6	-20.2	-24.2	1153

Obs	BDS0000000000100030110004LQ5_3	BDS0000000000100030110004LQ5_6	BDS0000000000100030110004LQ5_12
4	-27	-14	28
5	16	-11	41
6	727	743	729

Obs	BDS0000000000100030110004LQ5%3	BDS0000000000100030110004LQ5%6	BDS0000000000100030110004LQ5%12
4	-6.2	-3.3	7.3
5	3.9	-2.5	10.6
6	170.7	181.2	171.9

Obs	BDS0000000000200000110001LQ5	BDS0000000000200000110001LQ5_3	BDS0000000000200000110001LQ5_6
4	6521	377	246
5	5739	-782	-405
6	4674	-1065	-1847

Obs	BDS0000000000200000110001LQ5_12	BDS0000000000200000110001LQ5%3	BDS0000000000200000110001LQ5%6
4	193	6.1	3.9
5	-369	-12.0	-6.6
6	-1601	-18.6	-28.3

Obs	BDS0000000000200000110001LQ5%12	BDS0000000000200050110001LQ5	BDS0000000000200050110001LQ5_3
4	3.0	155	-6
5	-6.0	153	-2
6	-25.5	99	-54

Obs	BDS0000000000200050110001LQ5_6	BDS0000000000200050110001LQ5_12	BDS0000000000200050110001LQ5%3
4	-9	11	-3.7
5	-8	-11	-1.3
6	-56	-65	-35.3

Obs	BDS0000000000200050110001LQ5%6	BDS0000000000200050110001LQ5%12
4	-5.5	7.6
5	-5.0	-6.7
6	-36.1	-39.6

Output 23.10.1 *continued***Retrieve Business Employment Dynamics Statistics Data**

Obs	year	period	periodName	BDS0000000000100000110001LQ5	BDS0000000000100000110001LQ5_3
7	2020	Q03	3rd Quarter	1491	446
8	2020	Q04	4th Quarter	1463	-28
Obs	BDS0000000000100000110001LQ5_6 BDS0000000000100000110001LQ5_12 BDS0000000000100000110001LQ5%3				
7	195			237	42.7
8	418			153	-1.9
Obs	BDS0000000000100000110001LQ5%6 BDS0000000000100000110001LQ5%12 BDS0000000000100030110004LQ5				
7	15.0			18.9	468
8	40.0			11.7	379
Obs	BDS0000000000100030110004LQ5_3 BDS0000000000100030110004LQ5_6 BDS0000000000100030110004LQ5_12				
7	-685			42	31
8	-89			-774	-31
Obs	BDS0000000000100030110004LQ5%3 BDS0000000000100030110004LQ5%6 BDS0000000000100030110004LQ5%12				
7	-59.4			9.9	7.1
8	-19.0			-67.1	-7.6
Obs	BDS0000000000200000110001LQ5 BDS0000000000200000110001LQ5_3 BDS0000000000200000110001LQ5_6				
7	9383			4709	3644
8	7334			-2049	2660
Obs	BDS0000000000200000110001LQ5_12 BDS0000000000200000110001LQ5%3 BDS0000000000200000110001LQ5%6				
7	3239			100.7	63.5
8	813			-21.8	56.9
Obs	BDS0000000000200050110001LQ5%12 BDS0000000000200050110001LQ5 BDS0000000000200050110001LQ5_3				
7	52.7			206	107
8	12.5			174	-32
Obs	BDS0000000000200050110001LQ5_6 BDS0000000000200050110001LQ5_12 BDS0000000000200050110001LQ5%3				
7	53			45	108.1
8	75			19	-15.5
Obs	BDS0000000000200050110001LQ5%6 BDS0000000000200050110001LQ5%12				
7	34.6			28.0	
8	75.8			12.3	

The following statements print the catalog data shown in [Output 23.10.2](#):

```
libname blscat "%sysget(BLS)";

data myCAT;
    set blscat.blsex10_catalog;
run;
```

```
proc print data=myCat; run;
```

Output 23.10.2 Catalog for Business Dynamics Statistics

Retrieve Business Employment Dynamics Statistics Data

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality
1	1	1	Gross job gains for the goods-producing sector in the U.S. (rounded to the nearest thousands)	BDS0000000000100000110001LQ5	Seasonally Adjusted
2	2	2	Gross job losses for the manufacturing sector in the U.S. (rounded to the nearest thousands)	BDS0000000000100030110004LQ5	Seasonally Adjusted
3	3	3	Gross job gains for the service-providing sector in the U.S. (rounded to the nearest thousands)	BDS0000000000200000110001LQ5	Seasonally Adjusted
4	4	4	Gross job gains for the information sector in the U.S. (rounded to the nearest thousands)	BDS0000000000200050110001LQ5	Seasonally Adjusted

Obs	survey_name	survey_abbreviation	measure_data_type	commerce_industry	commerce_sector	area
1	Business Employment Dynamics	BD	Gross Job Gains	Goods-producing	Private Sector	U.S. totals
2	Business Employment Dynamics	BD	Gross Job Losses	Manufacturing	Private Sector	U.S. totals
3	Business Employment Dynamics	BD	Gross Job Gains	Service-providing	Private Sector	U.S. totals
4	Business Employment Dynamics	BD	Gross Job Gains	Information	Private Sector	U.S. totals

Example 23.11: Retrieving Occupational Employment and Wage Statistics (OEWS) Data

This example shows how to retrieve Occupational Employment and Wage Statistics (OEWS) data. You can view these data by using the following URL:

<https://www.bls.gov/oes/data.htm>

INSERIES is an input SAS data set that lists the IDs of the time series to be read from the OEWS database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing the ID of each series to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series to retrieve using their IDs. In this example, one time series is requested, along with any calculations.

The date range of the data starts with 2011, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2021, to include in the range of data to retrieve. The FREQ=ANNUAL (or FREQ=A) option specifies the selection of annual data. The CALCULATIONS=TRUE option is specified, but the data that are returned do not contain any net change calculation or percentage change calculation statistics, so no calculations exist in the SAS Output data set. The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.11.1](#) shows the data that are retrieved. [Output 23.11.2](#) shows the catalog information for the retrieved data.

```
options validvarname=any;
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

DATA inseries;
  length seriesid $ 32;
  seriesid = 'OEUN0000000000000000000004'; output;
RUN;

title 'Retrieve Occupational Employment and Wage Statistics Data';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=OES
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex11
  STARTYEAR='2011'
  ENDYEAR='2021'
  FREQ=Annual
  CALCULATIONS=true
  CATALOG=true
  FORMAT=json;

data myOES;
  set bls.blsex11;
run;

proc contents data=myOES; run;
proc print data=myOES; run;
```

Output 23.11.1 Occupational Employment and Wage Statistics

Retrieve Occupational Employment and Wage Statistics Data

Obs	year	period	periodName	OEUN0000000000000000000004
1	2021	A01	Annual	58260

The following statements print the catalog data shown in [Output 23.11.2](#):

```
libname blscat "%sysget(BLS)";

data myCAT;
  set blscat.blsex11_catalog;
run;
```

```
proc print data=myCat; run;
```

Output 23.11.2 Catalog for Occupational Employment and Wage Statistics

Retrieve Occupational Employment and Wage Statistics Data

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality	survey_name
1	1	1	Annual mean wage for All Occupations in All Industries in the United States	OEUN00000000000000000004	Not Seasonally Adjusted	Occupational Employment and Wage Statistics

Obs	survey_abbreviation	measure_data_type	commerce_industry	occupation	area	area_type
1	OE	Annual mean wage	Cross-industry, Private, Federal, State, and Local Government	All Occupations	National	National

Example 23.12: Retrieving American Time Use Survey (ATUS) Data

This example shows how to retrieve American Time Use Survey (ATUS) data. You can view these data by using the following URL:

<https://www.bls.gov/tus/database.htm>

INSERIES is an input SAS data set that lists the IDs of the time series to be read from the ATUS database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing each series ID to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series to retrieve using their IDs. In this example, four time series are requested.

The date range of the data starts with 2009, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2019, to include in the range of data to retrieve. The FREQ=ANNUAL (or FREQ=A) option specifies the selection of annual data. The ASPECTS=TRUE option is specified, and the data that are retrieved show the standard error, name_aspects, and the value, value_aspects, that are associated with each observation of each time series. The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.12.1](#) shows the data that are retrieved. [Output 23.12.2](#) shows the catalog information for the retrieved data. [Output 23.12.3](#) shows the aspects information for the retrieved data.

```
options validvarname=any;
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

DATA inseries;
```

```

length seriesid $ 32;
seriesid = 'TUU10101AA01001181'; output;
seriesid = 'TUU10101AA01001260'; output;
seriesid = 'TUU20101AA01002999'; output;
seriesid = 'TUU20101AA01003084'; output;
RUN;

title 'Retrieve American Time Use Survey Data';
libname bls sasebls "%sysget(BLS)";
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=ATUS
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex12
  STARTYEAR='2009'
  ENDYEAR='2019'
  FREQ=Annual
  ASPECTS=true
  CATALOG=true
  FORMAT=json;

data myTUS;
  set bls.blsex12;
run;

proc contents data=myTUS; run;
proc print data=myTUS; run;

```

Output 23.12.1 American Time Use Survey Data**Retrieve American Time Use Survey Data**

Obs	year	period	periodName	TUU10101AA01001181	TUU10101AA01001260
1	2009	A01	Annual	0.29	0.77
2	2010	A01	Annual	0.32	0.79
3	2011	A01	Annual	0.31	0.79
4	2012	A01	Annual	0.28	0.75
5	2013	A01	Annual	0.33	0.80
6	2014	A01	Annual	0.34	0.82
7	2015	A01	Annual	0.35	0.83
8	2016	A01	Annual	0.35	0.82
9	2017	A01	Annual	0.36	0.83
10	2018	A01	Annual	0.37	0.80
11	2019	A01	Annual	0.39	0.79

Obs	TUU20101AA01002999	TUU20101AA01003084
1	7.90	6.97
2	7.82	7.14
3	8.00	7.21
4	8.07	7.16
5	7.97	7.08
6	8.14	7.28
7	7.91	7.21
8	8.04	7.11
9	8.05	7.24
10	7.88	7.31
11	7.97	7.17

The following statements print the catalog data shown in [Output 23.12.2](#):

```
libname blscat "%sysget(BLS)";

data myCAT;
    set blscat.blsex12_catalog;
run;

proc print data=myCAT; run;
```

Output 23.12.2 Catalog for American Time Use Survey Data**Retrieve American Time Use Survey Data**

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality
1	1	1	Avg hrs per day - Food preparation and cleanup, Men	TUU10101AA01001181	Not Seasonally Adjusted
2	2	2	Avg hrs per day - Food preparation and cleanup, Women	TUU10101AA01001260	Not Seasonally Adjusted
3	3	3	Avg hrs per day for participants - Working, Employed, Men	TUU20101AA01002999	Not Seasonally Adjusted
4	4	4	Avg hrs per day for participants - Working, Employed, Women	TUU20101AA01003084	Not Seasonally Adjusted

Obs	survey_name	survey_abbreviation	measure_data_type	cps_labor_force_status	demographic_age
1	American Time Use	TU	Average hours per day	All persons	15 years and over
2	American Time Use	TU	Average hours per day	All persons	15 years and over
3	American Time Use	TU	Average hours per day for participants in an activity	Employed	15 years and over
4	American Time Use	TU	Average hours per day for participants in an activity	Employed	15 years and over

Obs	demographic_race	demographic_gender	demographic_children	demographic_education
1	All races	Men	All persons	All education levels
2	All races	Women	All persons	All education levels
3	All races	Men	All persons	All education levels
4	All races	Women	All persons	All education levels

Obs	cps_family_children	cps_activity	day_of_week	earnings
1	All persons	Food preparation and cleanup	All days	All persons
2	All persons	Food preparation and cleanup	All days	All persons
3	All persons	Working	All days	All persons
4	All persons	Working	All days	All persons

The following statements print the aspects data as shown in [Output 23.12.3](#). Each time series has its own corresponding name aspects and value aspects time series. The first series, TUU10101AA01001181, has the corresponding name aspects time series TUU10101AA01001181_name_aspects, and it also has the corresponding value aspects time series TUU10101AA01001181_value_aspects. This naming convention, where the time series name also appears as a prefix in the name aspects and value aspects series names, is used for each of the retrieved series.

```
libname blscat "%sysget(BLS)";

data myASPECTS;
    set blscat.blsex12_data_aspects;
```

```
run;
```

```
proc print data=myASPECTS; run;
```

Output 23.12.3 Aspects Output for American Time Use Survey Data**Retrieve American Time Use Survey Data**

Obs	year	period	periodName	TUU10101AA01001181	TUU10101AA01001181_name_asp	TUU10101AA01001181_val_asp
1	2009	A01	Annual	0.29	Standard Error	0.010
2	2010	A01	Annual	0.32	Standard Error	0.009
3	2011	A01	Annual	0.31	Standard Error	0.010
4	2012	A01	Annual	0.28	Standard Error	0.009
5	2013	A01	Annual	0.33	Standard Error	0.011
6	2014	A01	Annual	0.34	Standard Error	0.011
7	2015	A01	Annual	0.35	Standard Error	0.013
8	2016	A01	Annual	0.35	Standard Error	0.012
9	2017	A01	Annual	0.36	Standard Error	0.012

Obs	TUU10101AA01001260	TUU10101AA01001260_name_asp	TUU10101AA01001260_val_asp	TUU20101AA01002999
1	0.77	Standard Error	0.013	7.90
2	0.79	Standard Error	0.015	7.82
3	0.79	Standard Error	0.017	8.00
4	0.75	Standard Error	0.014	8.07
5	0.80	Standard Error	0.014	7.97
6	0.82	Standard Error	0.016	8.14
7	0.83	Standard Error	0.016	7.91
8	0.82	Standard Error	0.017	8.04
9	0.83	Standard Error	0.022	8.05

Obs	TUU20101AA01002999_name_asp	TUU20101AA01002999_val_asp	TUU20101AA01003084
1	Standard Error	0.076	6.97
2	Standard Error	0.072	7.14
3	Standard Error	0.074	7.21
4	Standard Error	0.079	7.16
5	Standard Error	0.078	7.08
6	Standard Error	0.080	7.28
7	Standard Error	0.078	7.21
8	Standard Error	0.074	7.11
9	Standard Error	0.072	7.24

Obs	TUU20101AA01003084_name_asp	TUU20101AA01003084_val_asp
1	Standard Error	0.059
2	Standard Error	0.079
3	Standard Error	0.067
4	Standard Error	0.077
5	Standard Error	0.072
6	Standard Error	0.069
7	Standard Error	0.083
8	Standard Error	0.080
9	Standard Error	0.088

Output 23.12.3 *continued***Retrieve American Time Use Survey Data**

Obs	year	period	periodName	TUU10101AA01001181	TUU10101AA01001181_name_asp	TUU10101AA01001181_val_asp
10	2018	A01	Annual	0.37	Standard Error	0.013
11	2019	A01	Annual	0.39	Standard Error	0.012

Obs	TUU10101AA01001260	TUU10101AA01001260_name_asp	TUU10101AA01001260_val_asp	TUU20101AA01002999
10	0.80	Standard Error	0.017	7.88
11	0.79	Standard Error	0.016	7.97

Obs	TUU20101AA01002999_name_asp	TUU20101AA01002999_val_asp	TUU20101AA01003084
10	Standard Error	0.083	7.31
11	Standard Error	0.083	7.17

Obs	TUU20101AA01003084_name_asp	TUU20101AA01003084_val_asp
10	Standard Error	0.081
11	Standard Error	0.091

Example 23.13: Retrieving Employment Projections (EPP) Data

This example shows how to retrieve Employment Projections (EPP) data. You can view these data by using the following URL:

<https://www.bls.gov/emp/data.htm>

INSERTIES is an input SAS data set that lists the IDs of the time series to be read from the EP database. In this example, the input SAS data set is named in the INSET=INSERTIES option. To create the INSERTIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERTIES'. Next, the length of the series ID is specified as 32 bytes prior to listing each series ID to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERTIES data set for the list of series to retrieve using their IDs. In this example, four time series are requested, along with any calculations. Employment Projections data are available for only one year, so the date range is specified for that year. The date range of the data starts with 2021, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2021, to include in the range of data to retrieve. The FREQ=ANNUAL (or FREQ=A) option specifies the selection of annual data. There is no catalog for the EPP data set, so the description of each time series can be obtained from the series_title column at the following URL:

<https://download.bls.gov/pub/time.series/ep/ep.series>

You can place these descriptions in the label for each time series, as shown in the LABEL statements of the myEPP DATA step.

The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.13.1](#) shows the data that are retrieved.


```

options validvarname=any;

DATA inseries;
  length seriesid $ 32;
  seriesid = 'EPU000000110000'; output;
  seriesid = 'EPU000000220000'; output;
  seriesid = 'EPU000000230000'; output;
  seriesid = 'EPU000000324000'; output;
RUN;

title 'Retrieve Employment Projections Data';
libname bls sasebls "%sysget(BLS)";
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=EPP
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex13
  STARTYEAR='2021'
  ENDYEAR='2021'
  FREQ=Annual
  FORMAT=json;

data myEPP;
  set bls.blsex13;
  label EPU000000110000=
    'Total, all occupations in mining, quarrying, oil and gas extraction';
  label EPU000000220000='Total, all occupations in Utilities';
  label EPU000000230000='Total, all occupations in Construction';
  label EPU000000324000='Total, all occupations in Petroleum and coal';
run;

proc contents data=myEPP; run;
proc print data=myEPP; run;

```

Output 23.13.1 Employment Projections Data

Retrieve Employment Projections Data

Obs	year	period	periodName	EPU000000110000	EPU000000220000	EPU000000230000	EPU000000324000
1	2021	A01	Annual	1460.2	540.8	7413.1	105.6

Example 23.14: Retrieving International Labor Comparisons (ILC) Data

This example shows how to retrieve International Labor Comparisons (ILC) data. You can view these data by using the following URL:

`https://www.bls.gov/fls/data.htm`

INSERIES is an input SAS data set that lists the time series IDs to be retrieved from the IN database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing each series ID to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The SASEBLS LIBNAME statement in the code that follows specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series IDs to retrieve. In this example, three time series are requested, along with any calculations.

The date range of the data starts with 2013, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2013, to include in the range of data to retrieve. The FREQ=MONTHLY option specifies the selection of monthly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include the net change calculation and percentage change calculation statistics. There is no catalog for the IN data set, so the description of each time series can be obtained by using the BLS series report tool at the following URL:

`https://data.bls.gov/cgi-bin/srgate`

These descriptions can be placed in the label for each time series as shown in the LABEL statements of the myINS DATA step.

The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.14.1](#) shows the data that are retrieved.

```
options validvarname=any;

DATA inseries;
  length seriesid $ 32;
  seriesid = 'INS00220000JP0M'; output; /* Japan monthly */
  seriesid = 'INS00220000US0M'; output; /* US monthly */
  seriesid = 'INS00220000CA0M'; output; /* Canada monthly*/
RUN;

title 'Retrieve International Civilian Unemployment Rate Data';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=ILC
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex14
  STARTYEAR='2013'
```

```
ENDYEAR='2013'
FREQ=Monthly
CALCULATIONS=true
FORMAT=json;

data myINS;
  set bls.blsex14;
  label INS00220000JP0M=
    'Civilian Labor Force Unemployment Rates for Whole Economy, Japan';
  label INS00220000US0M=
    'Civilian Labor Force Unemployment Rates for Whole Economy, United States';
  label INS00220000CA0M=
    'Civilian Labor Force Unemployment Rates for Whole Economy, Canada';
run;

proc contents data=myINS; run;
proc print data=myINS label; run;
```

Output 23.14.1 International Labor Survey

Retrieve International Civilian Unemployment Rate Data

Obs	Year of Observation	Period of Observation	periodName	Civilian Labor Force Unemployment Rates for Whole Economy, Canada	1-Month Net Change for INS00220000CA0M	3-Month Net Change for INS00220000CA0M	6-Month Net Change for INS00220000CA0M
1	2013	M01	January	6.0	-0.2	-0.4	-0.3
2	2013	M02	February	6.1	0.1	-0.1	-0.1
3	2013	M03	March	6.3	0.2	0.1	0.0

Obs	12-Month Net Change for INS00220000CA0M	1-Month % Change for INS00220000CA0M	3-Month % Change for INS00220000CA0M	6-Month % Change for INS00220000CA0M	12-Month % Change for INS00220000CA0M	Civilian Labor Force Unemployment Rates for Whole Economy, Japan
1	-0.5	-3.2	-6.3	-4.8	-7.7	3.8
2	-0.2	1.7	-1.6	-1.6	-3.2	3.8
3	0.1	3.3	1.6	0.0	1.6	3.6

Obs	1-Month Net Change for INS00220000JP0M	3-Month Net Change for INS00220000JP0M	6-Month Net Change for INS00220000JP0M	12-Month Net Change for INS00220000JP0M	1-Month % Change for INS00220000JP0M	3-Month % Change for INS00220000JP0M
1	0.0	0.1	0.0	-0.2	0.0	2.7
2	0.0	0.1	0.0	-0.2	0.0	2.7
3	-0.2	-0.2	-0.2	-0.4	-5.3	-5.3

Obs	6-Month % Change for INS00220000JP0M	12-Month % Change for INS00220000JP0M	Civilian Labor Force Unemployment Rates for Whole Economy, United States	1-Month Net Change for INS00220000US0M	3-Month Net Change for INS00220000US0M	6-Month Net Change for INS00220000US0M
1	0.0	-5.0	7.9	0.1	0.0	-0.3
2	0.0	-5.0	7.7	-0.2	-0.1	-0.4
3	-5.3	-10.0	7.6	-0.1	-0.2	-0.2

Obs	12-Month Net Change for INS00220000US0M	1-Month % Change for INS00220000US0M	3-Month % Change for INS00220000US0M	6-Month % Change for INS00220000US0M	12-Month % Change for INS00220000US0M
1	-0.4	1.3	0.0	-3.7	-4.8
2	-0.6	-2.5	-1.3	-4.9	-7.2
3	-0.6	-1.3	-2.6	-2.6	-7.3

Output 23.14.1 *continued*

Retrieve International Civilian Unemployment Rate Data

Obs	Year of Observation	Period of Observation	periodName	Civilian Labor Force Unemployment Rates for Whole Economy, Canada	1-Month Net Change for INS00220000CA0M	3-Month Net Change for INS00220000CA0M	6-Month Net Change for INS00220000CA0M
4	2013	M04	April	6.2	-0.1	0.2	-0.2
5	2013	M05	May	6.1	-0.1	0.0	-0.1
6	2013	M06	June	6.2	0.1	-0.1	0.0

Obs	12-Month Net Change for INS00220000CA0M	1-Month % Change for INS00220000CA0M	3-Month % Change for INS00220000CA0M	6-Month % Change for INS00220000CA0M	12-Month % Change for INS00220000CA0M	Civilian Labor Force Unemployment Rates for Whole Economy, Japan
4	-0.2	-1.6	3.3	-3.1	-3.1	3.6
5	-0.3	-1.6	0.0	-1.6	-4.7	3.6
6	-0.1	1.6	-1.6	0.0	-1.6	3.4

Obs	1-Month Net Change for INS00220000JP0M	3-Month Net Change for INS00220000JP0M	6-Month Net Change for INS00220000JP0M	12-Month Net Change for INS00220000JP0M	1-Month % Change for INS00220000JP0M	3-Month % Change for INS00220000JP0M
4	0.0	-0.2	-0.1	-0.4	0.0	-5.3
5	0.0	-0.2	-0.1	-0.3	0.0	-5.3
6	-0.2	-0.2	-0.4	-0.4	-5.6	-5.6

Obs	6-Month % Change for INS00220000JP0M	12-Month % Change for INS00220000JP0M	Civilian Labor Force Unemployment Rates for Whole Economy, United States	1-Month Net Change for INS00220000US0M	3-Month Net Change for INS00220000US0M	6-Month Net Change for INS00220000US0M
4	-2.7	-10.0	7.5	-0.1	-0.4	-0.4
5	-2.7	-7.7	7.6	0.1	-0.1	-0.2
6	-10.5	-10.5	7.6	0.0	0.0	-0.2

Obs	12-Month Net Change for INS00220000US0M	1-Month % Change for INS00220000US0M	3-Month % Change for INS00220000US0M	6-Month % Change for INS00220000US0M	12-Month % Change for INS00220000US0M
4	-0.6	-1.3	-5.1	-5.1	-7.4
5	-0.6	1.3	-1.3	-2.6	-7.3
6	-0.6	0.0	0.0	-2.6	-7.3

Example 23.15: Retrieving Occupational Requirements Survey (ORS) Data

This example shows how to retrieve data from the Occupational Requirements Survey (ORS). You can view these data by using the following URL:

<https://www.bls.gov/ors/data.htm>

INSERTIES is an input SAS data set that lists the time series IDs to be read from the OR database. In this example, the input SAS data set is named in the INSET=INSERTIES option. To create the INSERTIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERTIES'. Next, the length of the series ID is specified as 32 bytes prior to listing each series ID to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERTIES data set for the list of series IDs to retrieve. In this example, two time series are requested, along with any calculations. The date range of the data starts with 2022, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2022, to include in the range of data to retrieve. The FREQ=ANNUAL (or FREQ=A) option specifies the selection of annual data, and the CATALOG=TRUE option requests that catalog information be retrieved. The FORMAT= option specifies that the data are to be retrieved using the JSON format. [Output 23.15.1](#) shows the data that are retrieved. [Output 23.15.2](#) shows the catalog information for the retrieved data.

```
options validvarname=any;
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

DATA inseries;
  length seriesid $ 32;
  seriesid = 'ORUC1000000000001167'; output;
  seriesid = 'ORUC1000000000001164'; output;
RUN;

title 'Retrieve Occupational Requirements Survey Data';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=ORS
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex15
  STARTYEAR='2022'
  ENDYEAR='2022'
  FREQ=Annual
  CATALOG=true
  FORMAT=json;

data myORS;
  set bls.blsex15;
run;
```

```
proc contents data=myORS; run;
proc print data=myORS; run;
```

Output 23.15.1 Occupational Requirements Survey Data

Retrieve Occupational Requirements Survey Data

Obs	year	period	periodName	ORUC1000000000001164	ORUC1000000000001167
1	2022	A01	Annual	2.4	17.1

The following statements print the catalog information shown in [Output 23.15.2](#):

```
proc contents data=blscat.blsex15_catalog; run;
proc print data=blscat.blsex15_catalog; run;
```

Output 23.15.2 Catalog for Occupational Requirements Survey Data

Retrieve Occupational Requirements Survey Data

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality	survey_name
1	1	1	Percent of civilian workers; workload is controlled by machinery, equipment, or software	ORUC1000000000001164	Not Seasonally Adjusted	Occupational Requirements Survey
2	2	2	Percent of civilian workers; workload is self-paced	ORUC1000000000001167	Not Seasonally Adjusted	Occupational Requirements Survey

Obs	survey_abbreviation	measure_data_type	commerce_industry	occupation	occupation_work_class
1	OR	Percentage	All workers	All workers	All workers
2	OR	Percentage	All workers	All workers	All workers

Obs	requirements	type_of_cases
1	Cognitive and mental requirements	Pace: Control of workload
2	Cognitive and mental requirements	Pace: Control of workload

Example 23.16: Retrieving Work Stoppages (WS) Data

This example shows how to retrieve data from the Work Stoppages program (WS). You can view these data by using the following URL:

<https://www.bls.gov/wsp/data>

INseries is an input SAS data set that lists the time series IDs to be read from the Work Stoppages program (WSP) database. In this example, the input SAS data set is named in the INSET=INseries option. To create the INseries data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INseries'. Next, the length of the series ID is specified as 32 bytes prior to listing each series ID to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INseries data set for the list of series IDs to retrieve. In this example, two time series are requested, along with any calculations.

The date range of the data starts with 2022, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2023, to include in the range of data to retrieve. The FREQ=MONTHLY option specifies the selection of monthly data, but because the CALCULATIONS=TRUE option is specified, the retrieved data also include available net change calculation and percentage change calculation statistics.

Output 23.16.1 shows the selected monthly data that are available. The FORMAT= option specifies that the data are to be retrieved using the JSON format. Output 23.16.1 shows the data that are retrieved. Output 23.16.2 shows the catalog information for the retrieved data.

```
options validvarname=any;
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

DATA inseries;
    length seriesid $ 32;
    seriesid = 'WSU100'; output;
    seriesid = 'WSU010'; output;
    seriesid = 'WSU001'; output;
RUN;

title 'Retrieve Work Stoppages Data';
libname bls sasebls "%sysget(BLS)"
    USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    SETNAME=WSP
    INSET=inseries
    PAYPATH= "%sysget(BLS)payload"
    OUTJSON=blsex16
    STARTYEAR='2022'
    ENDYEAR='2023'
    FREQ=Monthly
    CALCULATIONS=true
    CATALOG=true
    FORMAT=json;
```



```
data myWSP;  
    set bls.blsex16;  
run;  
  
proc print data=myWSP; run;
```

Output 23.16.1 Work Stoppages Data**Retrieve Work Stoppages Data**

Obs	year	period	periodName	WSU001	WSU001_1	WSU001_3	WSU001_6	WSU001_12	WSU001%1
1	2022	M01	January	145.0	-6.9	-116.0	30.8	105.2	-4.5
2	2022	M02	February	55.1	-89.9	-258.9	-17.7	20.9	-62.0
3	2022	M03	March	166.4	111.3	14.5	71.5	83.4	202.0
4	2022	M04	April	97.6	-68.8	-47.4	-163.4	-40.8	-41.3
5	2022	M05	May	32.0	-65.6	-23.1	-282.0	-74.0	-67.2
6	2022	M06	June	22.2	-9.8	-144.2	-129.7	-120.4	-30.6
7	2022	M07	July	20.0	-2.2	-77.6	-125.0	-94.2	-9.9
8	2022	M08	August	88.0	68.0	56.0	32.9	15.2	340.0
9	2022	M09	September	92.6	4.6	70.4	-73.8	-2.3	5.2
10	2022	M10	October	73.8	-18.8	53.8	-23.8	-187.2	-20.3

Obs	WSU001%3	WSU001%6	WSU001%12	WSU010	WSU010_1	WSU010_3	WSU010_6	WSU010_12
1	-44.4	27.0	264.3	8	1.0	-7.4	6.6	6.6
2	-82.5	-24.3	61.1	0.0	-8.0	-40.0	-1.0	0.0
3	9.5	75.3	100.5	9.1	9.1	2.1	7.1	4.8
4	-32.7	-62.6	-29.5	16.5	7.4	8.5	1.1	10.3
5	-41.9	-89.8	-69.8	4.2	-12.3	4.2	-35.8	4.2
6	-86.7	-85.4	-84.4	1.0	-3.2	-8.1	-6.0	-1.0
7	-79.5	-86.2	-82.5	0	-1.0	-16.5	-8	-1.4
8	175.0	59.7	20.9	10.2	10.2	6.0	10.2	9.2
9	317.1	-44.4	-2.4	17.2	7.0	16.2	8.1	15.2
10	269.0	-24.4	-71.7	1.8	-15.4	1.8	-14.7	-13.6

Obs	WSU010%1	WSU010%3	WSU010%6	WSU010%12	WSU100	WSU100_1	WSU100_3	WSU100_6
1	14.3	-48.1	471.4	471.4	1	0	-3	0
2	-100.0	-100.0	-100.0		0	-1	-1	-1
3		30.0	355.0	111.6	2	2	1	1
4	81.3	106.3	7.1	166.1	4	2	3	0
5	-74.5		-89.5		3	-1	3	2
6	-76.2	-89.0	-85.7	-50.0	1	-2	-1	0
7	-100.0	-100.0	-100.0	-100.0	0	-1	-4	-1
8		142.9		920.0	4	4	1	4
9	68.6	1620.0	89.0	760.0	3	-1	2	1
10	-89.5		-89.1	-88.3	1	-2	1	-3

Obs	WSU100_12	WSU100%1	WSU100%3	WSU100%6	WSU100%12
1	0	0.0	-75.0	0.0	0.0
2	0	-100.0	-100.0	-100.0	
3	0		100.0	100.0	0.0
4	1	100.0	300.0	0.0	33.3
5	3	-25.0		200.0	
6	0	-66.7	-50.0	0.0	0.0
7	-1	-100.0	-100.0	-100.0	-100.0
8	3		33.3		300.0
9	2	-25.0	200.0	50.0	200.0
10	-3	-66.7		-75.0	-75.0

Output 23.16.1 *continued***Retrieve Work Stoppages Data**

Obs	year	period	periodName	WSU001	WSU001_1	WSU001_3	WSU001_6	WSU001_12	WSU001%1
11	2022	M11	November	613.0	539.2	525.0	581.0	299.0	730.6
12	2022	M12	December	737.2	124.2	644.6	715.0	585.3	20.3
13	2023	M01	January	34	-703.2	-39.8	14.0	-111.0	-95.4
14	2023	M02	February	0.0	-34.0	-613.0	-88.0	-55.1	-100.0
15	2023	M03	March	199.4	199.4	-537.8	106.8	33.0	

Obs	WSU001%3	WSU001%6	WSU001%12	WSU010	WSU010_1	WSU010_3	WSU010_6	WSU010_12
11	596.6	1815.6	95.2	49.7	47.9	39.5	45.5	9.7
12	696.1	3220.7	385.3	2.9	-46.8	-14.3	1.9	-4.1
13	-53.9	70.0	-76.6	7	4.1	5.2	7	-1
14	-100.0	-100.0	-100.0	0.0	-7.0	-49.7	-10.2	0.0
15	-73.0	115.3	19.8	67.2	67.2	64.3	50.0	58.1

Obs	WSU010%1	WSU010%3	WSU010%6	WSU010%12	WSU100	WSU100_1	WSU100_3	WSU100_6
11	2661.1	387.3	1083.3	24.3	2	1	-2	-1
12	-94.2	-83.1	190.0	-58.6	2	0	-1	1
13	141.4	288.9		-12.5	1	-1	0	1
14	-100.0	-100.0	-100.0		0	-1	-2	-4
15		2217.2	290.7	638.5	2	2	0	-1

Obs	WSU100_12	WSU100%1	WSU100%3	WSU100%6	WSU100%12
11	1	100.0	-50.0	-33.3	100.0
12	1	0.0	-33.3	100.0	100.0
13	0	-50.0	0.0		0.0
14	0	-100.0	-100.0	-100.0	
15	0		0.0	-33.3	0.0

The following statements print the catalog information shown in [Output 23.16.2](#):

```
libname blscat "%sysget(BLS)";

data myCAT;
  set blscat.blsex16_catalog;
run;

proc print data=myCAT; run;
```

Output 23.16.2 Catalog for Work Stoppages Data**Retrieve Work Stoppages Data**

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality	survey_name	survey_abbreviation
1	1	1	Days of idleness (in 1,000s) from all work stoppages in effect in the period	WSU001	Not Seasonally Adjusted	Work Stoppage Data	WS
2	2	2	Number of workers (in 1,000s) involved in all work stoppages beginning in the period	WSU010	Not Seasonally Adjusted	Work Stoppage Data	WS
3	3	3	Number of work stoppages involving 1,000 workers or more beginning in the period	WSU100	Not Seasonally Adjusted	Work Stoppage Data	WS

Obs	measure_data_type
1	Days of idleness (in 1,000s) from all work stoppages in effect in the period
2	Number of workers (in 1,000s) involved in all work stoppages beginning in the period
3	Number of work stoppages involving 1,000 workers or more beginning in the period

Example 23.17: Retrieving Local Area Unemployment Statistics (LAUS) Data

This example shows how to retrieve data from the Local Area Unemployment Statistics (LAUS) program. You can view these data by using the following URL:

`https://www.bls.gov/lau/data.htm`

INSERIES is an input SAS data set that lists the time series IDs to be read from the Local Area Unemployment Statistics (LAUS) program database. In this example, the input SAS data set is named in the INSET=INSERIES option. To create the INSERIES data set, use the DATA step. In the DATA statement, the name of the input data set is specified as 'INSERIES'. Next, the length of the series ID is specified as 32 bytes prior to listing each series ID to be accessed. Each time series is listed on its own line, along with the OUTPUT statement. When the series IDs have been listed, the DATA step is ended with a RUN statement. The following SASEBLS LIBNAME statement specifies the libref (BLS), followed by the engine name (SASEBLS) and the physical path where the retrieved BLS data are to be stored. The OUTJSON= option names the file where the JSON data are stored. The output data set is created using the OUTJSON= option and placed in the folder that is specified in the SASEBLS LIBNAME statement. The INSET= option specifies the INSERIES data set for the list of series IDs to retrieve. In this example, three time series are requested, along with the catalog (CATALOG=TRUE option).

The date range of the data starts with 2022, as specified by the STARTYEAR= option. The ENDYEAR= option specifies the last year, 2023, to include in the range of data to retrieve. The FREQ=MONTHLY option specifies the selection of monthly data, and because the CATALOG=TRUE option is specified, the retrieved data also include available catalog information.

The `FORMAT=` option specifies that the data are to be retrieved using the JSON format. [Output 23.17.1](#) shows the selected monthly data that are retrieved. [Output 23.17.2](#) shows the catalog information for the retrieved data. The data from the three retrieved time series come from three different metropolitan statistical areas (MSAs).

```
options validvarname=any;
libname blscat "%sysget(BLS)";      /* for Catalog data set*/

DATA inseries;
  length seriesid $ 32;
  seriesid = 'LAUMT373958000000003'; output; /* Raleigh, NC MSA*/
  seriesid = 'LAUMT064174000000003'; output; /* San Diego-Carlsbad, CA MSA*/
  seriesid = 'LAUMT364506000000003'; output; /* Syracuse, NY MSA*/
RUN;

title 'Retrieve Local Area Unemployment Statistics Data';
libname bls sasebls "%sysget(BLS)"
  USER='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  SETNAME=LAUS
  INSET=inseries
  PAYPATH= "%sysget(BLS)payload"
  OUTJSON=blsex17
  STARTYEAR='2022'
  ENDYEAR='2023'
  FREQ=Monthly
  CATALOG=true
  FORMAT=json;

data myLAUS;
  set bls.blsex17 ;
  label LAUMT373958000000003='Raleigh, NC MSA, unemployment rate';
  label LAUMT064174000000003='San Diego-Carlsbad, CA MSA, unemployment rate';
  label LAUMT364506000000003='Syracuse, NY MSA, unemployment rate';
run;

proc print data=myLAUS; run;
```

Output 23.17.1 Local Area Unemployment Rate**Retrieve Local Area Unemployment Statistics Data**

Obs	year	period	periodName	LAUMT0641740000000003	LAUMT3645060000000003	LAUMT3739580000000003
1	2022	M01	January	4.6	4.0	3.3
2	2022	M02	February	4.1	4.2	3.1
3	2022	M03	March	3.6	3.7	2.9
4	2022	M04	April	3.2	3.2	2.8
5	2022	M05	May	2.9	3.1	3.1
6	2022	M06	June	3.4	3.4	3.4
7	2022	M07	July	3.4	3.6	3.3
8	2022	M08	August	3.5	3.7	3.6
9	2022	M09	September	3.1	3.0	3.0
10	2022	M10	October	3.3	2.8	3.2
11	2022	M11	November	3.3	3.0	3.1
12	2022	M12	December	3.0	3.2	2.7
13	2023	M01	January	3.7	4.1	3.1
14	2023	M02	February	3.7	3.8	3.1
15	2023	M03	March	3.7	3.4	3.1
16	2023	M04	April	3.3	2.6	2.8

The following statements print the catalog information shown in [Output 23.17.2](#):

```
libname blscat "%sysget(BLS)";

data myCAT;
    set blscat.blsex17_catalog;
run;

proc print data=myCAT; run;
```

Output 23.17.2 Catalog for Local Area Unemployment Rate
Retrieve Local Area Unemployment Statistics Data

Obs	ordinal_series	ordinal_catalog	series_title	series_id	seasonality	survey_name
1	1	1	Unemployment Rate: San Diego-Carlsbad, CA Metropolitan Statistical Area (U)	LAUMT0641740000000003	Not Seasonally Adjusted	Local Area Unemployment Statistics
2	2	2	Unemployment Rate: Syracuse, NY Metropolitan Statistical Area (U)	LAUMT3645060000000003	Not Seasonally Adjusted	Local Area Unemployment Statistics
3	3	3	Unemployment Rate: Raleigh, NC Metropolitan Statistical Area (U)	LAUMT3739580000000003	Not Seasonally Adjusted	Local Area Unemployment Statistics

Obs	survey_abbreviation	measure_data_type	area	area_type
1	LA	unemployment rate	San Diego-Carlsbad, CA Metropolitan Statistical Area	Metropolitan areas
2	LA	unemployment rate	Syracuse, NY Metropolitan Statistical Area	Metropolitan areas
3	LA	unemployment rate	Raleigh, NC Metropolitan Statistical Area	Metropolitan areas

References

- Bureau of Labor Statistics (2020). "BLS Public Data API Signatures (Version 2.0)." Accessed May 5, 2022. https://www.bls.gov/developers/api_signature_v2.htm.
- Bureau of Labor Statistics (2022). "Download.bls.gov - /pub/time.series/." Accessed May 5, 2022. <https://download.bls.gov/pub/time.series/>.

Chapter 24

SASEMOOD Interface Engine

Contents

Overview: SASEMOOD Interface Engine	1672
Using CAS Sessions and CAS Engine Librefs	1672
Getting Started: SASEMOOD Interface Engine	1673
Syntax: SASEMOOD Interface Engine	1675
The LIBNAME <i>libref</i> SASEMOOD Statement	1675
Details: SASEMOOD Interface Engine	1681
SAS Output Data Set	1681
Examples: SASEMOOD Interface Engine	1682
Example 24.1: Retrieving Consumer Price Index Data from Moody’s Analytics Data Buffet	1682
Example 24.2: Retrieving Quarterly Gross Domestic Product Data by Industry from the Bureau of Economic Analysis (BEA)	1683
Example 24.3: Retrieving Weekly Unemployment Insurance Claim Statistics for the United States	1684
Example 24.4: Retrieving Daily Data from the S&P 500	1685
Example 24.5: Retrieving GDP by Industry: Chain-Type Price Index for Value Added for the United States	1687
Example 24.6: Retrieving Unemployment Rate Data for the United States	1688
Example 24.7: Retrieving Employment Data: Total, Nonfarm, for the United States	1690
Example 24.8: Retrieving Unemployment Data and Household Debt to Personal Disposable Income Ratio for the United States	1692
Example 24.9: Retrieving National Income and Product Account Data	1694
References	1695

Overview: SASEMOOD Interface Engine

The SASEMOOD interface engine enables SAS users to retrieve time series data from the Moody's Analytics Data Buffet (Economy.com) website. This website offers access to over 600 sources of global historical statistical data at the regional, national, and subnational level, in addition to 40 forecast database offerings. Over 220 million time series are updated promptly after release, on topics such as housing, labor, demographics, finance, industry, housing prices, and consumer credit performance forecasts. Time series are offered in yearly, semiannual, quarterly, monthly, semimonthly, biweekly, weekly, workday (business), daily, and indexed frequencies.

The SASEMOOD interface engine uses the LIBNAME statement to enable you to download Moody's online economic data by using the Data Buffet API described at the following URL:

<https://www.economy.com/products/tools/api>

It also enables you to specify which time series you want to retrieve, by using the mnemonic code that corresponds to the data set that you select. You choose the time range of the retrieved data by specifying a start date and an end date. You can then use the SAS DATA step to perform further subsetting, retrieve the data, and store the resulting time series in a SAS data set and in a CAS table (if desired).

You can view the catalog of available Moody's databases on the web page at the following URL:

<https://www.economy.com/databuffet/>

The SASEMOOD interface engine supports Linux X64 (LAX) and Windows.

Although the SASEMOOD engine uses Moody's Analytics Data Buffet online API, it is not endorsed or certified by Moody's Analytics. By using the SASEMOOD interface engine, you are agreeing to comply with the terms of use, which are described on the web page at the following URL:

<https://www.economy.com/about/terms-of-use>

For more information see the Moody's Analytics (2019) Data Buffet API User Guide.

To get started using the SASEMOOD engine, follow the steps in the section “[Getting Started: SASEMOOD Interface Engine](#)” on page 1673, which enable you to view the GDP\$ (Gross Domestic Product in the National Income and Product Accounts) database to retrieve the GDP\$ time series. The sample SAS code for accessing the Moody's GDP\$ time series appears at the end of the section.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).” For more information about how to store your Moody’s data in a CAS table, see [Example 24.9](#) in this chapter.

Getting Started: SASEMOOD Interface Engine

You need a subscription to use Moody’s Analytics Data Buffet web service, and you must have the proper credentials to access this service. For more information, see “Using Data Buffet: Data Buffet API v1.0” (Thorne 2019b). After you log on, you will see your access key and encryption key in your account information at the following URL:

<https://www.economy.com/myeconomy/api-key-info>

For more information about getting started using Data Buffet, see “Using Data Buffet: Which Series Can You Download?” (Thorne 2019f).

The web page at the following URL describes how to use the Moody’s Analytics Data Buffet API (Moody’s Analytics 2019):

<https://github.com/moodysanalytics/databuffet.api.codesamples/blob/master/databuffet-api-userguide.pdf>

The statements at the end of this section enable you to access the mnemonic GDP\$.IUSA, which is described as “Gross domestic product, (Mil. Ch 2012 USD, SAAR) for United States.”

The MNEMONIC= option specifies the time series mnemonic code(s) that you need in order to retrieve the data from Moody’s Analytics Data Buffet. GDP\$.IUSA is the time series mnemonic code in the sample code shown at the end of this section.

The OUT= option specifies both the name of the resulting JSON file(s) and the name of the SAS data set.

The range of the retrieved data is determined by the START= and END= options. You specify the start and end dates in ‘YYYY-MM-DD’ format.

The JSON data that the Moody's Analytics Data Buffet website returns are placed in a file that is named by the OUT= option—in this case, *GDP1.json*. **NOTE:** The SASEMOOD engine appends a numeral to the JSON file name, and the file extension (.json) is excluded from the file name that appears in the OUT= option. When the SET statement is executed, the JSON data are read (and merged) into a SAS data set named GDP.sas7bdat, which resides in the location that is specified inside the string enclosed in double quotation marks in the LIBNAME statement.

The result is named in the DATA step in the SET statement and is shown in [Output 24.1](#).

For another example that uses more LIBNAME statement options, see the section “[Examples: SASEMOOD Interface Engine](#)” on page 1682.

```
options validvarname=any;

title 'Retrieve Gross Domestic Product Data for United States';

libname mylib "/sasusr/mood/";

title1 "National Income and Product Accounts Databank from Moody's Analytics Data Buffet";
title2 "Request US Gross Domestic Product";
libname moody sasemood "%sysget(MOOD_DATA)"
  mnemonic="GDP$.IUSA"      /* steps 2 and 3 */
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' /* access key */
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' /* encryption key */
  freq=quarterly
  conv=2
  outjson=GDP
  start='2017-01-01'
  end='2017-12-01'
  ;

data mylib.myGDP;
  set moody.GDP;           /* GDP is specified in the OUT= option */
run;

title3 "The mylib.myGDP Data Set";
proc print data=mylib.myGDP; run;
```

Figure 24.1 US Gross Domestic Product

**National Income and Product Accounts Databank from Moody's Analytics Data Buffet
Request US Gross Domestic Product
The mylib.myGDP Data Set**

Obs	date	GDP\$.IUSA
1	2017-03-31	17925256
2	2017-06-30	18021048
3	2017-09-30	18163558
4	2017-12-31	18322464

Syntax: SASEMOOD Interface Engine

The SASEMOOD interface engine uses standard engine syntax to read the observations or data values for one or more time series from Moody's Analytics Data Buffet.

Table 24.1 summarizes the options that the SASEMOOD engine uses.

Table 24.1 Summary of SASEMOOD Interface Engine Options

Option	Description
CASLIB=	Specifies the name of the CASLIB where the in-memory CAS table containing the MOODY's data is stored
CASOUT=	Specifies the name of the in-memory CAS table, which contains the data that the SASEMOOD interface engine returns
CONV=	Specifies the method to aggregate (or disaggregate) the frequency to
DEBUG=	Specifies whether to include diagnostic message logging in the SAS log window
END=	Specifies the end date of the retrieved data
FREQ=	Specifies the frequency to convert the time series to (if a conversion is necessary)
GOCAS=	Specifies whether to generate an in-memory CAS table for the Moody's data
MNEMONIC=	Specifies the required Moody's data set mnemonic code (concept code and geography code) that enables you to specify which data to select from Moody's Analytics Data Buffet
OUT=	Specifies the name of the SAS data set and the JSON file, which contains the data that the SASEMOOD interface engine returns
PASS=	Specifies the required Moody's encryption key that enables you to access Moody's Analytics Data Buffet
START=	Specifies the start date of the retrieved data
TRANS=	Specifies the transformation function to be applied to the selected data sets
USER=	Specifies the required Moody's access key to authenticate the request for the retrieved data

The LIBNAME libref SASEMOOD Statement

LIBNAME libref **SASEMOOD** '*physical-name*' options ;

The LIBNAME statement assigns a SAS library reference (libref) to the physical path of the directory where the SASEMOOD interface engine stores the Moody's data. The required *physical-name* argument specifies the final location of the folder where your Moody's data reside in a SAS data set named by the OUT= option. The *physical-name* should end with a backslash if you are in a Windows environment or a forward slash if you are in a UNIX environment.

You can specify the following *options*:

CASLIB=*mood_caslib_name*

specifies the name of the CAS library (caslib) where the in-memory CAS table is stored when the JSON data from Moody's Analytics Data Buffet are read into SAS. You can use this option to name your caslib where the in-memory CAS table is stored. The default for this option is the active caslib. The GOCAS=ON option is required when you specify the CASLIB= option.

CASOUT=*mood_castable_name***OUTCAS=***mood_castable_name*

specifies the name of the in-memory CAS table that is created when the JSON data from Moody's Analytics Data Buffet are read into SAS. You can use this option to name your in-memory CAS table. The default for this option is the same name that you specify in the OUT= option. The in-memory table is created only when you also specify the GOCAS=ON option.

CONV=CONSTANT | CUBIC | DISCRETE | LINEAR**CONVERSION=CONSTANT | CUBIC | DISCRETE | LINEAR**

specifies the technique to use to convert the native frequency to the desired frequency. You can specify the following values:

CONSTANT (or **0**) converts the native frequency to a cubic frequency.

CUBIC (or **2**) converts the native frequency to a constant frequency.

DISCRETE (or **3**) converts the native frequency to a discrete frequency.

LINEAR (or **1**) converts the native frequency to a linear frequency.

By default, CONV=CUBIC. For more about native frequency and frequency conversion, see "Using Data Buffet: The Convert Function" (Thorne 2019d).

For more information about frequency conversion, see "Using Data Buffet: How Frequency Conversion Works" (Zandi 2019) on the web page at the following URL:

<https://www.economy.com/support/blog/buffet.aspx?did=FE8BEE0D-53E5-4F6A-AF55-8A9E765DCE68>

DEBUG=ON | OFF

specifies whether or not to include diagnostic message logging in the SAS log window. This information can be useful for troubleshooting a problem.

END=*mood_endDate***ENDDATE=***mood_endDate***ENDATE=***mood_endDate***END_TIME=***mood_endDate***ENDTIME=***mood_endDate***OBSERVATION_END=***mood_endDate*

specifies the end date of the Moody's data that you request. Specify the *mood_endDate* in 'YYYY-MM-DD' format. The valid data range of available data varies with each Moody's data set.

FREQ=*mood_frequency***FREQUENCY**=*mood_frequency***INTERVAL**=*mood_frequency***PERIOD**=*mood_frequency*

specifies the desired frequency for the time series that are listed in the MNEMONIC= option. You can specify the *mood_frequency* as annual, quarterly, monthly, weekly, daily, and so on. Native is the default frequency when the FREQ= option is not specified. For more information about frequency conversion, see “Using Data Buffet: Caveats When Converting Low Frequencies” (Thorne 2019a). For a complete list of frequencies and their frequency codes, see [Table 24.2](#).

Table 24.2 Frequency Codes for Moody’s Analytics Data and the FREQ= Option

Interval or Frequency	Moody’s Frequency Code	Description
Yearly, year, annual, ann, anndec	204	Annual, year ending in December
Annnov	203	Annual, year ending in November
Annoct	202	Annual, year ending in October
Annsep	201	Annual, year ending in September
Annaug	200	Annual, year ending in August
Annjul	199	Annual, year ending in July
Annjun	198	Annual, year ending in June
Annmay	197	Annual, year ending in May
Annapr	196	Annual, year ending in April
Annmar	195	Annual, year ending in March
Annfeb	194	Annual, year ending in February
Annjan	193	Annual, year ending in January
Semiannual	188	Semiannual, year ending in December
Semnov	187	Semiannual, year ending in November
Semoct	186	Semiannual, year ending in October
Semsep	185	Semiannual, year ending in September
Semaug	184	Semiannual, year ending in August
Semjul	183	Semiannual, year ending in July
Quarterly, Qtr, Qtrdec, Quarter	172	Quarterly, year ending in December
Qtrnov	171	Quarterly, year ending in November
Qtroct	170	Quarterly, year ending in October
Bimdec	156	Bimonthly, year ending in December
Bimnov	155	Bimonthly, year ending in November
Monthly, mon, month	128	Monthly
Semmon	112	Semimonthly, 15th and end of month
Bwsat2	110	Biweekly, ending on alternating Saturdays, starting Jan. 26, 1850, Feb. 9, 1850, ..., Dec. 30, 2017, Jan. 13, 2018, Jan. 27, 2018
Bwfri2	109	Biweekly, ending on alternating Fridays, starting Jan. 25, 1850, Feb. 8, 1850, ..., Dec. 29, 2017, Jan. 12, 2018, Jan. 26, 2018

Table 24.2 *continued*

Interval or Frequency	Moody's Frequency Code	Description
Bwthu2	108	Biweekly, ending on alternating Thursdays, starting Jan. 24, 1850, Feb. 7, 1850, ..., Dec. 28, 2017, Jan. 11, 2018, Jan. 25, 2018
Bwwed2	107	Biweekly, ending on alternating Wednesdays, starting Jan. 23, 1850, Feb. 6, 1850, ..., Dec. 27, 2017, Jan. 10, 2018, Jan. 24, 2018
Bwtue2	106	Biweekly, ending on alternating Tuesdays, starting Jan. 22, 1850, Feb. 5, 1850, ..., Dec. 26, 2017, Jan. 9, 2018, Jan. 23, 2018
Bwmon2	105	Biweekly, ending on alternating Mondays, starting Jan. 21, 1850, Feb. 4, 1850, ..., Dec. 25, 2017, Jan. 8, 2018, Jan. 22, 2018
Bwsun2	104	Biweekly, ending on alternating Sundays, starting Jan. 20, 1850, Feb. 3, 1850, ..., Dec. 24, 2017, Jan. 7, 2018, Jan. 21, 2018
Bwsat1	103	Biweekly, ending on alternating Saturdays, starting Jan. 19, 1850, Feb. 2, 1850, ..., Dec. 23, 2017, Jan. 6, 2018, Jan. 20, 2018
Bwfri1	102	Biweekly, ending on alternating Fridays, starting Jan. 18, 1850, Feb. 1, 1850, ..., Dec. 22, 2017, Jan. 5, 2018, Jan. 19, 2018
Bwthu1	101	Biweekly, ending on alternating Thursdays, starting Jan. 17, 1850, Jan. 31, 1850, ..., Dec. 21, 2017, Jan. 4, 2018, Jan. 18, 2018
Bwwed1	100	Biweekly, ending on alternating Wednesdays, starting Jan. 16, 1850, Jan. 30, 1850, ..., Dec. 20, 2017, Jan. 3, 2018, Jan. 17, 2018
Bwtue1	99	Biweekly, ending on alternating Tuesdays, starting Jan. 15, 1850, Jan. 29, 1850, ..., Dec. 19, 2017, Jan. 2, 2018, Jan. 16, 2018
Bwmon1	98	Biweekly, ending on alternating Mondays, starting Jan. 14, 1850, Jan. 28, 1850, ..., Dec. 18, 2017, Jan. 1, 2018, Jan. 15, 2018
Bwsun1	97	Biweekly, ending on alternating Sundays, starting Jan. 27, 1850, Feb. 10, 1850, ..., Dec. 31, 2017, Jan. 14, 2018, Jan. 28, 2018
Tenday	80	Three times a month: 10th, 20th, and end of month
Weekly, week, wsaturday	71	Weekly, ending Saturday
Wfriday, wfri	70	Weekly, ending Friday
Wthursday, wthu	69	Weekly, ending Thursday
Wwednesday, wwed	68	Weekly, ending Wednesday

Table 24.2 continued

Interval or Frequency	Moody's Frequency Code	Description
Wtuesday, wtue	67	Weekly, ending Tuesday
WMonday, wmon	66	Weekly, ending Monday
WSunday, wsun	65	Weekly, ending Sunday
Business, busin, bus	50	Business daily, Monday through Friday
Daily, day	49	Daily
Index	16	INDEX
Default	0	Native

GOCAS=ON | OFF

specifies whether to create (ON) or not create (OFF) an in-memory CAS table of the Moody's data (in addition to the SAS data set that is created). When GOCAS=ON, the SASEMOOD engine assumes that there is an active CAS session running when it executes the LIBNAME statement. By default, GOCAS=OFF.

MNEMONIC='mood_mnemonic_list'

ID='mood_mnemonic_list'

IDS='mood_mnemonic_list'

specifies the Moody's mnemonic code (or codes) that enables you to access the data set that corresponds to that code. Each mnemonic code consists of a concept code and a geography code, separated by a period (.). When you want more than one mnemonic code, specify the mnemonic codes in a list, separated by semicolons. There are too many codes to list here; you can view the catalog of the available list by visiting the web page at the following URL:

<https://www.economy.com/databuffet/>

OUT=mood_jsonfile

OUTJSON=mood_jsonfile

JSON=mood_jsonfile

specifies the name of both the JSON file (downloaded) and the SAS data set that is created when the JSON data are read into SAS. You can use the OUT= option to name your JSON data file. It is stored in the SAS Work library. The SAS data set that is created (when the JSON data are read into SAS) is stored in the folder specified by *physical-name*, and you can refer to it by using the myLib libref in your LIBNAME statement.

PASS='mood_password'

ENCRYPTKEY='mood_password'

specifies the unique PASS (password) encryption key that Moody's Analytics assigns to the user (subscriber) to provide security for accessing the Moody's Analytics Data Buffet online data.

USER='mood_user'

ACCESSKEYID='mood_user'

specifies the unique USER access key that Moody's Analytics assigns to the user (subscriber) to allow access to the Moody's Analytics Data Buffet API service.

START=mood_start

STARTDATE=mood_start

STDATE=mood_start

START_TIME=mood_start

STARTTIME=mood_start

BEGIN=mood_start

OBSERVATION_START=mood_start

specifies the start date for the Moody's data that you request. Specify the data in 'YYYY-MM-DD' format. The valid date range for available data varies with each Moody's data set.

TRANS=0 | 1 | 2 | 3 | 4 | 8

TRANSFORMATION=0 | 1 | 2 | 3 | 4 | 8

specifies the transformation function to be performed on the selected series. You can specify the following values:

0 indicates none.

1 indicates year-over-year percentage change.

2 indicates simple difference.

3 indicates annualized growth.

4 indicates percentage change.

8 indicates year-over-year difference.

By default, TRANS=0 (no transformation). For more information about Moody's transformations, see "Using Data Buffet: Transformations" (Thorne 2019e) at the following URL:

<https://www.economy.com/support/blog/buffet.aspx?did=783658E9-93DA-4F85-A206-9D14FC44904F>

Details: SASEMOOD Interface Engine

The SASEMOOD interface engine enables SAS users to access time series data that are stored in Moody's Analytics Data Buffet data sets. Each Moody's time series is identified by a unique mnemonic code. The mnemonic code consists of a concept code and a geography code. For more information about mnemonic codes, see "Using Data Buffet: Mnemonic, concept code and geo code" (Thorne 2019c) on the web page at the following URL:

<https://www.economy.com/support/blog/buffet.aspx?did=FC5702E3-32B7-4068-AC3B-5D18D3400416>

SAS Output Data Set

You can use a SAS DATA step to write the selected Moody's time series data to a SAS data set. This enables you to use SAS software to easily analyze the data. If you specify the name of the output data set in the DATA statement, the SAS engine supervisor creates a SAS data set that has the specified name in the location specified by the SASEMOOD engine libref's *physical-name*.

The contents of the SAS data set include the date of each observation and the name of each time series that is read from the Moody's website.

You can use the PRINT and CONTENTS procedures to print your output data set and its contents. Alternatively, you can view your SAS output observations by opening the desired output data set in a SAS Explorer window. You can also use the SQL procedure with your SASEMOOD engine libref to create a custom view of your data.

Because each SASEMOOD engine libref results in retrieving the requested data from the Moody's Analytics Data Buffet website, it is best to use a DATA step to store the data. You should avoid the inefficient use of the SASEMOOD engine libref that follows:

```
proc print data=moody.GDP; run;
```

Here the member name, GDP, is used in the PROC PRINT statement that invokes the Moody libref to run the SASEMOOD engine. It is more efficient to refer to the SAS data set myGDP repeatedly than to invoke the interface engine repeatedly. This use of the member name, GDP, corresponds to specifying the OUT=GDP option. Although using this statement might seem easier, it is not as efficient, because every time you use the SASEMOOD engine libref, the SASEMOOD engine reads the entire JSON file into SAS again.

Examples: SASEMOOD Interface Engine

Example 24.1: Retrieving Consumer Price Index Data from Moody's Analytics Data Buffet

This example requests Consumer Price Index data (all items, total for United States) from the OECD Main Economic Indicators database. The MNEMONIC= option specifies the Moody's data to retrieve, which in this example is OCMCPALTT01IXOBAM.IUSA. The FREQ= option requests monthly data. The range is specified using the START= and END= options. The OUTJSON= option names the JSON file and the SAS output data set that contains the time series data from the downloaded JSON file. The CONV= option specifies cubic conversion to the desired frequency. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set.

The SAS code is shown after the next paragraph, and it is followed by the output, shown in [Output 24.1.1](#), for the selected date range.

The SET statement reads observations from the input data set Moody.monCPI, and the DATA step stores them in a SAS data set named myMONCPI. When all the data are retrieved, they are placed in a SAS data set that is named by the OUT= option and that is located in the folder specified by the *physical-name* in the LIBNAME statement.

```
options validvarname=any;

title 'Request Consumer Price Index (Total - All Items - Index 2015-100)';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
    mnemonic="OCMCPALTT01IXOBAM.IUSA"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    freq=monthly
    conv=2
    outjson=moncpi
    start='2017-01-01'
    end='2018-05-01'
    format=json;

data myMONCPI;
    set moody.moncpi;
run;

proc print data=myMONCPI; run;
```

Output 24.1.1 US Monthly CPI Data in OECD Main Economic Indicators Database**Request Consumer Price Index (Total - All Items - Index 2015=100)**

Obs	date	OCMCPALTT01IXOBAM.IUSA
1	2017-01-31	102.827
2	2017-02-28	102.958
3	2017-03-31	102.828
4	2017-04-30	102.971
5	2017-05-31	102.915
6	2017-06-30	103.022
7	2017-07-31	103.085
8	2017-08-31	103.496
9	2017-09-30	103.978
10	2017-10-31	104.031
11	2017-11-30	104.352
12	2017-12-31	104.569
13	2018-01-31	104.978
14	2018-02-28	105.256
15	2018-03-31	105.230
16	2018-04-30	105.460
17	2018-05-31	105.748

Example 24.2: Retrieving Quarterly Gross Domestic Product Data by Industry from the Bureau of Economic Analysis (BEA)

This example shows how to retrieve gross domestic product data by industry, starting in the second quarter of 2016 and ending in the third quarter of 2018. The output, which is shown in [Output 24.2.1](#), contains two variables, Date and PDPGPOD1Q.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character ‘.’ in the mnemonic code; the same mnemonic code is used to name the SAS variable’s series name in the SAS output data set.

The MNEMONIC= option specifies the Moody’s time series data to retrieve, which in this example is PDPGPOD1Q.IUSA. The FREQ= option selects the quarterly frequency, and the START= and END= options define the date range of the retrieved data.

```
options validvarname=any;

title 'Request Gross Domestic Product by Industry, Quarterly Data';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
  mnemonic="PDPGPOD1Q.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=quarterly
  conv=cubic
  outjson=GDPI
  start='2016-01-01'
  end='2018-06-01'
```

```

format=json
;

data myIGDPQ;
  set moody.GDPI;
run;

proc print data=myIGDPQ; run;

```

Output 24.2.1 GDP by Industry: Chain-Type Price Index for Value Added – All Industries
Request Gross Domestic Product by Industry, Quarterly Data

Obs	date	PDPGPOD1Q.IUSA
1	2016-03-31	104.933
2	2016-06-30	105.618
3	2016-09-30	105.987
4	2016-12-31	106.543
5	2017-03-31	107.040
6	2017-06-30	107.394
7	2017-09-30	108.032
8	2017-12-31	108.715
9	2018-03-31	109.341
10	2018-06-30	110.209

Example 24.3: Retrieving Weekly Unemployment Insurance Claim Statistics for the United States

This example shows how to retrieve weekly unemployment insurance claim statistics data for the United States, starting January 1, 2018, and ending May 1, 2018. The output is shown in [Output 24.3.1](#). The MNEMONIC= option specifies the Moody's data set to retrieve; its mnemonic code is UNIW.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The START= and END= options define the date range of the selected data.

```

options validvarname=any;

title 'Request Unemployment Insurance Claim Statistics, Weekly Data';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
  mnemonic="UNIW.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=weekly
  conv=cubic
  outjson=uniweek
  start='2018-01-01'
  end='2018-05-01'
  format=json;

data myUNIWK;
  set moody.uniweek;
run;

```

```
proc contents data=myUNIWK; run;
proc print data=myUNIWK; run;
```

Output 24.3.1 US Unemployment Insurance Claim Weekly Statistics

Request Unemployment Insurance Claim Statistics, Weekly Data

Obs	date	UNIWK.IUSA
1	2018-01-06	251000
2	2018-01-13	229000
3	2018-01-20	237000
4	2018-01-27	221000
5	2018-02-03	221000
6	2018-02-10	224000
7	2018-02-17	220000
8	2018-02-24	215000
9	2018-03-03	226000
10	2018-03-10	223000
11	2018-03-17	224000
12	2018-03-24	220000
13	2018-03-31	237000
14	2018-04-07	234000
15	2018-04-14	235000
16	2018-04-21	207000
17	2018-04-28	212000
18	2018-05-05	210000

Example 24.4: Retrieving Daily Data from the S&P 500

This example shows how to retrieve Standard & Poor's Dow Jones Indices data, starting in January 2018 and ending in February 2018. The output is shown in [Output 24.4.1](#). The MNEMONIC= option specifies the Moody's data set to retrieve the data from; its mnemonic code is SP500D.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The START= and END= options define the date range.

```
options validvarname=any;

title 'Request S&P 500 Data, Daily Data';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
    mnemonic="SP500D.IUSA"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    freq=daily
    conv=cubic
    outjson=sp500
    start='2018-01-01'
    end='2018-02-01'
```

```

format=json
;

data mySP500;
    set moody.sp500;
run;

proc print data=work.mySP500; run;

```

Output 24.4.1 Stock Price Index: S&P 500 (Index 1941 to 1943=10, NSA)

Request S&P 500 Data, Daily Data

Obs	date	SP500D.IUSA
1	2018-01-01	2688.01
2	2018-01-02	2695.81
3	2018-01-03	2713.06
4	2018-01-04	2723.99
5	2018-01-05	2743.15
6	2018-01-06	2746.41
7	2018-01-07	2747.02
8	2018-01-08	2749.70
9	2018-01-09	2751.29
10	2018-01-10	2748.23
11	2018-01-11	2767.56
12	2018-01-12	2786.24
13	2018-01-13	2784.92
14	2018-01-14	2780.31
15	2018-01-15	2775.69
16	2018-01-16	2776.42
17	2018-01-17	2802.56
18	2018-01-18	2798.03
19	2018-01-19	2810.30
20	2018-01-20	2824.21
21	2018-01-21	2836.14
22	2018-01-22	2838.57
23	2018-01-23	2839.13
24	2018-01-24	2837.54
25	2018-01-25	2839.25
26	2018-01-26	2872.87
27	2018-01-27	2868.00
28	2018-01-28	2854.91
29	2018-01-29	2837.68
30	2018-01-30	2822.43
31	2018-01-31	2823.81
32	2018-02-01	2821.98

Example 24.5: Retrieving GDP by Industry: Chain-Type Price Index for Value Added for the United States

This example shows how to retrieve multiple time series for GDP by industry—utilities, manufacturing, finance, and all industries—starting January 1, 2016, and ending June 1, 2018. The MNEMONIC= option specifies the Moody's data to retrieve. This example lists four mnemonic codes, separated by semicolons. The FREQ= option specifies quarterly data intervals, and the START= and END= options define the date range of the data. The output is shown in [Output 24.5.1](#), which contains five variables: Date, PDPGPOD1Q.IUSA, PDPGPO22D1Q.IUSA, PDPGPOMFD1Q.IUSA, and PDPGPOFID1Q.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic name list; this list is used to name the SAS variables (time series) in the SAS output data set.

```
options validvarname=any;

title 'Request Gross Domestic Product by Industry for Multiple Time Series';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
  mnemonic="PDPGPOD1Q.IUSA;PDPGPO22D1Q.IUSA;PDPGPOMFD1Q.IUSA;PDPGPOFID1Q.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  mnemonic="PDPGPOD1Q.IUSA;PDPGPO22D1Q.IUSA;PDPGPOMFD1Q.IUSA;PDPGPOFID1Q.IUSA"
  freq=quarterly
  conv=cubic
  outjson=incGDP
  start='2016-01-01'
  end='2018-06-01'
  format=json;

data myIGDPQ;
  set moody.incGDP;
run;

proc print data=work.myIGDPQ; run;
```

Output 24.5.1 Four Time Series for GDP by Industry: Utilities, Manufacturing, Insurance, and All Industries

Request Gross Domestic Product by Industry for Multiple Time Series

Obs	date	PDPGPOD1Q.IUSA	PDPGPO22D1Q.IUSA	PDPGPOMFD1Q.IUSA	PDPGPOFID1Q.IUSA
1	2016-03-31	104.933	107.363	103.553	112.061
2	2016-06-30	105.618	107.974	103.663	113.373
3	2016-09-30	105.987	109.396	103.388	114.571
4	2016-12-31	106.543	108.743	104.032	115.582
5	2017-03-31	107.040	109.983	104.439	115.557
6	2017-06-30	107.394	111.066	104.183	116.405
7	2017-09-30	108.032	111.542	105.476	117.472
8	2017-12-31	108.715	111.480	106.157	119.072
9	2018-03-31	109.341	115.945	106.367	120.243
10	2018-06-30	110.209	113.648	107.424	121.325

Example 24.6: Retrieving Unemployment Rate Data for the United States

This example shows how to retrieve unemployment rate data for United States (total), starting in January 1973 and ending in January 2019. The output is shown in [Output 24.6.1](#). The MNEMONIC= option specifies the Moody's data set to retrieve the data from; its mnemonic code is LBR.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The START= and END= options define the date range. Although the native frequency of this series is monthly, the request is for an annual frequency, using the cubic conversion in the CONV= option.

```
options validvarname=any;

title 'Request Unemployment Data, Annual Frequency Using Cubic Conversion';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
    mnemonic="LBR.IUSA"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    freq=annual      /* Convert from monthly to annual frequency */
    conv=cubic
    outjson=labor
    start='1973-01-01'
    end='2019-01-01'
    format=json
    ;

data myLabor;
    set moody.labor;
run;

proc print data=work.myLabor; run;
```

Output 24.6.1 US Unemployment Rate: Total (% , SA)**Request Unemployment Data, Annual Frequency Using Cubic Conversion**

Obs	date	LBR.IUSA
1	1973-12-31	4.85833
2	1974-12-31	5.64167
3	1975-12-31	8.47500
4	1976-12-31	7.70000
5	1977-12-31	7.05000
6	1978-12-31	6.06667
7	1979-12-31	5.85000
8	1980-12-31	7.17500
9	1981-12-31	7.61667
10	1982-12-31	9.70833
11	1983-12-31	9.60000
12	1984-12-31	7.50833
13	1985-12-31	7.19167
14	1986-12-31	7.00000
15	1987-12-31	6.17500
16	1988-12-31	5.49167
17	1989-12-31	5.25833
18	1990-12-31	5.61667
19	1991-12-31	6.85000
20	1992-12-31	7.49167
21	1993-12-31	6.90833
22	1994-12-31	6.10000
23	1995-12-31	5.59167
24	1996-12-31	5.40833
25	1997-12-31	4.94167
26	1998-12-31	4.50000
27	1999-12-31	4.21667
28	2000-12-31	3.96667
29	2001-12-31	4.74167
30	2002-12-31	5.78333
31	2003-12-31	5.99167
32	2004-12-31	5.54167
33	2005-12-31	5.08333
34	2006-12-31	4.60833
35	2007-12-31	4.61667
36	2008-12-31	5.80000
37	2009-12-31	9.28333
38	2010-12-31	9.60833
39	2011-12-31	8.93333
40	2012-12-31	8.07500
41	2013-12-31	7.35833
42	2014-12-31	6.15833
43	2015-12-31	5.27500
44	2016-12-31	4.87500
45	2017-12-31	4.34167
46	2018-12-31	3.89167
47	2019-12-31	3.66667

Example 24.7: Retrieving Employment Data: Total, Nonfarm, for the United States

This example shows how to retrieve employment data for the United States (total, nonfarm), starting in June 2015 and ending in January 2019. The TRANS= option specifies the year-over-year percentage change transformation on the time series.

The output is shown in [Output 24.7.1](#). The MNEMONIC= option specifies the Moody's data set to retrieve the data from; its mnemonic code is LBR.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The START= and END= options define the date range.

```
options validvarname=any;

title 'Request Employment Data: Total Nonfarm Using Transformation';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
  mnemonic="ET.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=monthly
  conv=2
  trans=1          /* year-over-year percentage change*/
  outjson=etrans
  start='2015-06-01'
  end='2019-01-01'
  format=json
  ;

data myETrans;
  set moody.etrans;
run;

proc print data=work.myETrans; run;
```

Output 24.7.1 US Employment: Total, Nonfarm (Year-over-Year % Transform)**Request Employment Data: Total Nonfarm Using Transformation**

Obs	date	ET.IUSA
1	2015-06-30	2.06141
2	2015-07-31	2.10831
3	2015-08-31	2.06022
4	2015-09-30	1.94532
5	2015-10-31	1.98043
6	2015-11-30	1.93778
7	2015-12-31	1.93764
8	2016-01-31	1.85106
9	2016-02-29	1.84182
10	2016-03-31	1.95302
11	2016-04-30	1.88039
12	2016-05-31	1.67179
13	2016-06-30	1.73537
14	2016-07-31	1.76830
15	2016-08-31	1.77378
16	2016-09-30	1.85197
17	2016-10-31	1.73508
18	2016-11-30	1.67199
19	2016-12-31	1.63875
20	2017-01-31	1.71614
21	2017-02-28	1.66071
22	2017-03-31	1.58845
23	2017-04-30	1.59334
24	2017-05-31	1.67140
25	2017-06-30	1.63293
26	2017-07-31	1.53274
27	2017-08-31	1.56519
28	2017-09-30	1.38908
29	2017-10-31	1.47179
30	2017-11-30	1.52122
31	2017-12-31	1.45006
32	2018-01-31	1.40427
33	2018-02-28	1.55197
34	2018-03-31	1.58280
35	2018-04-30	1.53961
36	2018-05-31	1.62205
37	2018-06-30	1.62171
38	2018-07-31	1.56549
39	2018-08-31	1.60437
40	2018-09-30	1.64637
41	2018-10-31	1.59855
42	2018-11-30	1.53438
43	2018-12-31	1.56827
44	2019-01-31	1.66721

Example 24.8: Retrieving Unemployment Data and Household Debt to Personal Disposable Income Ratio for the United States

This example shows how to retrieve unemployment data and household debt data for the United States, starting in January 2009 and ending in January 2019. The `FREQ=` option specifies quarterly frequency conversion, using the cubic conversion in the `CONV=` option.

The output is shown in [Output 24.8.1](#). The `MNEMONIC=` option specifies the Moody's data sets to retrieve the data from; their mnemonic codes are `LBR.IUSA` and `DM%CHHDPIAQ.IUSA`. The SAS system option `VALIDVARNAME=ANY` allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The `START=` and `END=` options define the date range.

```
options validvarname=any;

title 'Request Unemployment Data and Household Debt to Personal Disposable Income Ratio';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
  mnemonic="LBR.IUSA;DM%CHHDPIAQ.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=quarterly
  conv=2
  outjson=labdebt
  start='2009-01-01'
  end='2019-01-01'
  format=json
  ;

data myLabD;
  set moody.labdebt;
run;

proc print data=work.myLabD; run;
```

Output 24.8.1 US Unemployment Data and Household Debt to Personal Disposable Income Ratio

Request Unemployment Data and Household Debt to Personal Disposable Income Ratio

Obs	date	LBR.IUSA	DM%CHHDPIA.Q.IUSA
1	2009-03-31	8.26667	130.182
2	2009-06-30	9.30000	128.033
3	2009-09-30	9.63333	128.243
4	2009-12-31	9.93333	127.003
5	2010-03-31	9.83333	125.126
6	2010-06-30	9.63333	122.552
7	2010-09-30	9.46667	120.727
8	2010-12-31	9.50000	119.304
9	2011-03-31	9.03333	117.152
10	2011-06-30	9.06667	115.851
11	2011-09-30	9.00000	113.993
12	2011-12-31	8.63333	113.136
13	2012-03-31	8.26667	110.363
14	2012-06-30	8.20000	108.800
15	2012-09-30	8.03333	109.268
16	2012-12-31	7.80000	105.969
17	2013-03-31	7.73333	110.175
18	2013-06-30	7.53333	109.342
19	2013-09-30	7.23333	108.739
20	2013-12-31	6.93333	108.409
21	2014-03-31	6.66667	106.576
22	2014-06-30	6.20000	105.504
23	2014-09-30	6.06667	104.458
24	2014-12-31	5.70000	103.637
25	2015-03-31	5.53333	103.246
26	2015-06-30	5.43333	102.856
27	2015-09-30	5.10000	101.941
28	2015-12-31	5.03333	101.779
29	2016-03-31	4.93333	101.568
30	2016-06-30	4.90000	101.973
31	2016-09-30	4.90000	102.106
32	2016-12-31	4.76667	101.663
33	2017-03-31	4.56667	100.824
34	2017-06-30	4.36667	100.949
35	2017-09-30	4.30000	100.546
36	2017-12-31	4.13333	100.210
37	2018-03-31	4.06667	98.648
38	2018-06-30	3.93333	98.247
39	2018-09-30	3.76667	97.922
40	2018-12-31	3.80000	97.565
41	2019-03-31	3.86667	96.891

Example 24.9: Retrieving National Income and Product Account Data

This example shows how to retrieve the National Income and Product Account (NIPA) data for two time series—exports of goods data and NIPA exports of goods and services data—starting January 1, 2017, and ending December 1, 2017. The data are quarterly, because each mnemonic code ends in “Q.IUSA”.

The output is shown in [Output 24.9.1](#). The MNEMONIC= option specifies the Moody’s data sets to retrieve the data from; their mnemonic codes are NCU18_B020RCQ.IUSA and NCU18_A253RCQ.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character ‘.’ in the mnemonic code; the same mnemonic code is used to name the SAS variable’s series name in the SAS output data set. The START= and END= options define the date range. The GOCAS=ON option specifies the data to be stored in a CAS in-memory table, which is named M2MNIPA in the CASOUT= option. The CASLIB= option specifies the CAS library name where the in-memory table, M2MNIPA, is stored.

```
options validvarname=any;

title 'Request NIPA Data';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
  gocas=on
  caslib=casuserhdfs /* CAS library for MOODY's data */
  mnemonic="NCU18_B020RCQ.IUSA;NCU18_A253RCQ.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=0
  conv=2
  outjson=YNIPA /* SAS data set name for the incoming NIPA data */
  casout=M2MNIPA /* in-memory CAS table name for requested NIPA data */
  start='2017-01-01'
  end='2017-12-01'
  format=json
;

data myYNIPA;
  set moody.ynipa;
run;

proc print data=work.myYNIPA; run;
```

Output 24.9.1 US Exports of Goods and Exports of Goods and Services

Request NIPA Data

Obs	date	NCU18_B020RCQ.IUSA	NCU18_A253RCQ.IUSA
1	2017-03-31	2303282	1503009
2	2017-06-30	2313183	1508669
3	2017-09-30	2360093	1535383
4	2017-12-31	2450348	1606363

In the preceding code, you can reference the in-memory CAS table, M2MNIPA, later in the same CAS session to save the data in a SASHDAT file, but do not specify the SASHDAT extension in the file name:


```
proc casutil;
    save casdata="M2MNIPA" casout="<physical path and filename>" replace;
quit;
```

To list the tables in the active caslib, specify the following statements:

```
proc casutil; list; quit;
```

References

- Moody's Analytics (2019). "Moody's Analytics Data Buffet API User Guide." Accessed July 17, 2019. <https://github.com/moodysanalytics/databuffet.api.codesamples/blob/master/dataBuffet-api-userguide.md>.
- Thorne, P. (2019a). "Using Data Buffet: Caveats When Converting Low Frequencies." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=D6427BAA-1803-4C88-A7C5-D1F6F791642A>.
- Thorne, P. (2019b). "Using Data Buffet: Data Buffet API v1.0." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=0978F3F5-60FF-41A3-B984-1DA75CD4BBE2>.
- Thorne, P. (2019c). "Using Data Buffet: Mnemonic, Concept Code and Geo Code." Accessed August 16, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=FC5702E3-32B7-4068-AC3B-5D18D3400416>.
- Thorne, P. (2019d). "Using Data Buffet: The CONVERT Function." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=A35042C7-BB22-4647-9570-4B634E9A952C>.
- Thorne, P. (2019e). "Using Data Buffet: Transformations." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=783658E9-93DA-4F85-A206-9D14FC44904F>.
- Thorne, P. (2019f). "Using Data Buffet: Which Series Can You Download?" Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=A00FDE45-EC0B-4140-8378-0FA3474D85FA>.
- Zandi, K. (2019). "Using Data Buffet: How Frequency Conversion Works." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=FE8BEE0D-53E5-4F6A-AF55-8A9E765DCE68>.

Chapter 25

SASEUSCB Interface Engine

Contents

Overview: SASEUSCB Interface Engine	1698
Using CAS Sessions and CAS Engine Librefs	1698
Getting Started: SASEUSCB Interface Engine	1699
Syntax: SASEUSCB Interface Engine	1701
The LIBNAME <i>libref</i> SASEUSCB Statement	1702
Details: SASEUSCB Interface Engine	1706
Data Elements Reference: SASEUSCB Interface Engine	1707
ADVM3 Data Set: Advance Report on Durable Goods, Manufacturers' Ship- ments, Inventories, and Orders	1708
BFS Data Set: Business Formation Statistics	1709
FTD Data Set: International Trade in Goods and Services	1711
SAS Output Data Set	1711
SAS OUTJSON File	1712
Examples: SASEUSCB Interface Engine	1712
Example 25.1: Retrieving (ADVM3) Data for Category MDM (Durable Goods) . . .	1712
Example 25.2: Retrieving Business Formation Statistics (BFS) Data for Business Ap- plications (BA_BA) Data Type	1714
Example 25.3: Retrieving International Trade in Goods and Services (FTD) Data for All Data Types	1719
References	1720

Overview: SASEUSCB Interface Engine

The SASEUSCB interface engine enables SAS users to retrieve economic data from the United States Census Bureau (USCB) website (www.census.gov), which is hosted by the principal agency in the US Department of Commerce that measures demographic, economic, and population data. The economic time series on the USCB website contain observation or measurement periods that are associated with data values.

The SASEUSCB interface engine uses the LIBNAME statement to enable you to specify how to subset your USCB data and select data according to their frequency and range. You can then use a SAS DATA step to perform further subsetting, retrieve the data, and store the resulting time series in a SAS data set and in a CAS data table (if desired).

The SASEUSCB interface engine supports 64-bit Windows and Linux X64 (LAX) platforms.

Note that the SASEUSCB engine uses the US Census Bureau API, but it is not endorsed or certified by the US Census Bureau, and that by using the SASEUSCB interface, you are agreeing to comply with the USCB terms of use. They are found on the web page at the following URL: <https://www.census.gov/data/developers/about/terms-of-service.html>.

To get started using the SASEUSCB engine, see the section “Getting Started: SASEUSCB Interface Engine” on page 1699, which shows how to view the time series data. The sample SAS code for accessing these data appears at the end of that section.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the `CASHOST` and `CASPORT` of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the `TERMINATE` option in the `CAS` statement as follows:

```
cas mysess terminate;
```

For more information about the `CAS` statement and the `LIBNAME` statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about `CAS` and `CAS` sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Getting Started: SASEUSCB Interface Engine

You can query the US Census Bureau (USCB) databases to retrieve the observations or data values for a list of economic time series by specifying the series ID of each time series that you want to read into SAS and by specifying your unique USCB API key. To obtain your own unique API key, start by visiting the USCB website:

<https://www.census.gov/data/developers.html>

Then click on the **Request a Key** box at the lower left of that page. Alternatively, you can go directly to the following site:

https://api.census.gov/data/key_signup.html

The USCB API key is a 40-character alphanumeric lowercase string, such as 'abcdefghijklmnopqrstuvwxyz12345612345678', and is represented by 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' in the `USER=` option in the following example. In addition, the example URLs in this section and in the section “[Details: SASEUSCB Interface Engine](#)” on page 1706 use the same USCB API key as the argument `your_uscb_apikey`.

Now that you are informed about the terms of service of the USCB data, you can use your USCB API registration key to access the USCB data, as shown in the example that follows.

The following statements enable you to access monthly time series from the EITS (Economic Indicator Time Series) ADVDM3 (Advanced Report on Durable Goods) database. The time series for defense aircraft and parts (`CATEGORY=DAP`) that are selected from the time period starting September 2012 (`STARTDATE='2012-09'`) and ending April 2013 (`ENDDATE='2013-04'`) are retrieved by using the following SAS DATA step and shown in [Figure 25.1](#):

```
options validvarname=any;

title 'Retrieve Monthly Data from EITS ADVDM3 (Advanced Report on Durable Goods)';
libname _all_ clear;

libname uscb saseuscb "%sysget(USCB) "
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    setname=eits_advdm3
    for='us:*'
    category=DAP
    startdate='2012-09'
    enddate='2013-04'
```

```

outjson=ADVM3
format=json;

data my_gsa;
  set uscb.ADVM3 ;
run;

proc contents data=my_gsa; run;
proc print data=my_gsa; run;

```

Figure 25.1 Advanced Report on Durable Goods Time Series: my_gsa

Retrieve Monthly Data from EITS ADVM3 (Advanced Report on Durable Goods)

Obs	date	DAP_MPCNO_S	DAP_MPCTI_S	DAP_MPCUO_S	DAP_MPCVS_S	DAP_NO_S	DAP_TI_S
1	2012-09	31.3	-3.1	1.9	3.8	6234	12637
2	2012-10	-7.7	-0.4	1.6	-5.0	5752	12587
3	2012-11	-2.5	2.6	0.9	7.1	5607	12909
4	2012-12	50.9	0.8	4.4	5.9	8462	13009
5	2013-01	-46.1	-1.4	0.1	-15.3	4561	12830
6	2013-02	-8.0	4.1	-1.0	10.8	4195	13354
7	2013-03	-15.7	-2.0	-2.0	2.4	3535	13086
8	2013-04	38.2	-1.7	0.1	-4.2	4886	12861

Obs	DAP_UO_S	DAP_VS_S	DAP_MPCNO_U	DAP_MPCTI_U	DAP_MPCUO_U	DAP_MPCVS_U
1	70626	4891	99.9	-3.5	2.5	15.2
2	71733	4645	-20.7	-0.2	1.9	-19.9
3	72367	4973	-2.0	2.4	0.7	15.9
4	75561	5268	102.8	-0.6	7.2	21.5
5	75662	4460	-59.4	-0.4	1.5	-43.6
6	74915	4942	-28.5	4.7	-1.4	24.9
7	73391	5059	32.1	-2.5	-2.4	42.2
8	73432	4845	-16.3	-1.4	-0.8	-32.1

Obs	DAP_NO_U	DAP_TI_U	DAP_UO_U	DAP_VS_U
1	7177	12641	69404	5470
2	5688	12610	70711	4381
3	5576	12909	71209	5078
4	11306	12829	76345	6170
5	4591	12782	77457	3479
6	3282	13389	76393	4346
7	4334	13049	74547	6180
8	3629	12861	73980	4196

The JSON data that the USCB website returns are placed in a file that is named by the OUTJSON= option—in this case, *ADVM3.json*. Note that the JSON file extension is excluded from the file name that is specified in the OUTJSON= option. When the SET statement is executed, the JSON data are read into a SAS data set named *ADVM3.sas7bdat*, which resides in the location that is specified inside the string enclosed in double quotation marks in the SASEUSCB LIBNAME statement. So, in the preceding example, if the USCB environment variable is set to */sasusr/uscb/test/*, then the SAS data set that is created by reading the downloaded JSON file is placed in the following file: */sasusr/uscb/test/ADVM3.sas7bdat*.

An equivalent LIBNAME statement that does not use any environment variables could be as follows:

```
libname uscb saseuscb "/sasusr/uscb/test/"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  setname=eits_advm3
  for='us:*'
  category=DAP
  startdate='2012-09'
  enddate='2013-04'
  outjson=ADVM3
  format=json;
```

You could also use either a SAS macro variable or a system environment variable to store the value of your USCB API registration key so that the key does not appear explicitly in your SAS code. The SETNAME= option names the USCB data set that you are retrieving data from.

The result, MY_GSA, is named in the DATA step and shown in [Figure 25.1](#). It is more efficient to use the DATA step to store your USCB data in a SAS data set and then refer to the SAS data set directly in your PROC PRINT or PROC SGLOT statement, but you can also refer to the SASEUSCB libref directly, as in the following statement:

```
proc print data=uscb.ADVM3; run;
```

The PROC PRINT statement uses the member name, ADVM3; this usage corresponds to specifying the OUTJSON=ADVM3 option. Although using this statement might seem easier, it is not as efficient, because every time you use the SASEUSCB libref, the USCB interface engine reads the entire JSON file into SAS again. It is better to refer to the SAS data set repeatedly than to invoke the interface engine repeatedly. For another example that uses more SASEUSCB LIBNAME statement options, see the section “[Details: SASEUSCB Interface Engine](#)” on page 1706.

The variable name for each ADVM3 time series is made up of three parts, separated by underscores ('_'): the category code, the data type code, and an 'S' for seasonal data or a 'U' for nonseasonal data. See [Table 25.7](#) and [Table 25.8](#) for the complete list of categories and data type codes, respectively, in the ADVM3 data set.

Syntax: SASEUSCB Interface Engine

The SASEUSCB interface engine uses standard engine syntax to read the observations or data values for one or more economic time series. [Table 25.1](#) summarizes the options that the interface engine uses. Two options are required: USER='uscb_apikey' and SETNAME='uscb_input_dataset_name'.

Table 25.1 Summary of LIBNAME libref SASEUSCB Options

Option	Description
CASLIB=	Specifies the name of the caslib where the in-memory CAS table containing the USCB data is stored
CASOUT=	Specifies the name of the in-memory CAS table, which contains the data that the SASEUSCB interface engine retrieves
CATEGORY=	Specifies the name of a category which is used to keep or select the desired variables based on their category

Table 25.1 *continued*

Option	Description
CONNECT=	Specifies whether or not to use the connect method for a secure connection via a proxy server. You must specify the PROXY= option when you use the CONNECT=ON option.
DATATYPE=	Specifies the name of the data type of the variables to keep
DEBUG=	Specifies whether or not to use diagnostic message logging in the SAS log window
ENDDATE=	Specifies the ending date for the observation period
FOR=	Specifies the demographic category that you want to retrieve data for
FORMAT=	Specifies a file extension that indicates the type of file to retrieve. Only the JSON format is supported.
GEO=	Specifies the geographic region (or geo-level) of the data that you want to retrieve: ‘US’ for United States, ‘NE’ for Northeast, ‘MW’ for Midwest, ‘SO’ for South, ‘WE’ for West. For US states, some data sets let you use the standard two-letter postal codes, such as AL for Alabama. For the complete list of geo-level codes, see Table 25.2 .
GOCAS=	Specifies whether to generate an in-memory CAS table for the USCB data
OUTJSON=	Specifies the name of the output SAS data set and the JSON file for the retrieved data
PROXY=	Specifies the proxy server that you want to use (if you have trouble connecting without specifying a proxy). If you also need the connect method for a secure connection, specify the CONNECT=ON option in addition to the PROXY= option. (See the CONNECT= option.)
SETNAME=	Specifies the name of the data set that you want to access; see Table 25.3 .
STARTDATE=	Specifies the starting date for the observation period
USER=	Specifies the required USCB registration key that enables you to access data on the USCB website

The LIBNAME *libref* SASEUSCB Statement

LIBNAME *libref* **SASEUSCB** '*physical-name*' *options* ;

The LIBNAME statement assigns a SAS library reference (*libref*) to the physical path of the directory where the SASEUSCB interface engine stores the USCB economic data. The required *physical-name* argument specifies the final location of the folder where your USCB data reside in a SAS data set that is named in the OUT= option. The *physical-name* should end with a backslash (for example, C:*physical_name*\) when you are in a Windows environment or a forward slash (for example, /*user/physical_name*/) when you are in a UNIX environment.

You can specify the following *options*:

CASLIB=*uscb_caslib_name*

specifies the name of the CAS library (caslib) where the in-memory CAS table is stored when the JSON data from the USCB website are read into SAS. The default value is the name of the active caslib. When you specify this option, you must also specify the GOCAS=ON option.

CASOUT=*libref.data-table***OUTCAS=***libref.data-table*

specifies the name of the in-memory CAS table that is created when the JSON data from USCB are read into SAS. The default value is the name that you specify in the OUT= option. The in-memory table is created only when you also specify the GOCAS=ON option. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section “Using CAS Sessions and CAS Engine Librefs” on page 1698.

CATEGORY=*uscb_category_code*

specifies the category of time series variables to keep. All categories are kept when the CATEGORY= option is omitted. See Table 25.6 for more information about categories.

CONNECT=ON | OFF

specifies whether to use (ON) or not use (OFF) the connect method along with the PROXY= option. **NOTE:** To use the connect method, you must use the PROXY= option and specify your proxy server in addition to the CONNECT=ON option. For more information about a secure connection, see the PROXY= option.

DATATYPE=*uscb_datatype_code*

specifies the data type of time series variables to keep. All data types are kept when the DATATYPE= option is omitted. See Table 25.6 for more information about data types.

DEBUG=ON | OFF

specifies whether or not to include diagnostic message logging in the SAS log window. This information can be useful for troubleshooting. Specify DEBUG=ON to see all the log messages (including debugging information) in the SAS log. By default, DEBUG=OFF.

ENDDATE=*'uscb_enddate'*

specifies the ending year of the time series in the format 'YYYY', the ending quarter and year for quarterly data in the format 'YYYY-Qx' (where x = 1, 2, 3, or 4), or the ending month and year for monthly data in the format 'YYYY-nn' (where nn = 01, 02, ..., 10, 11, or 12).

FOR=*uscb_for_param*

specifies the demographic category that you want to retrieve data for. The Advanced Report on Durable Goods, Manufacturers' Shipments, Inventories, and Orders (ADVM3) data set, which is the only data set supported in this release, requires this option. By

default, FOR=US (United States).

See the following web page for more information: <https://www.census.gov/data/developers/guidance/api-user-guide/core-concepts.html>.

FORMAT=uscb_jsonformat

specifies the format of the file to be retrieved from the USCB website. Although USCB can report data in many formats, the SASEUSCB engine supports only the JSON format (the default).

GEO='uscb_geo_level'

specifies the geographic region (or geo-level) that you want to retrieve data for. See [Table 25.2](#) for a complete list of geo-level codes, which identify the regions. By default, GEO=US (United States Total). Note that geo-level codes for US states are usually the same as the standard two-letter postal codes, such as AL for Alabama.

GOCAS=ON | OFF

specifies whether to create (ON) or not create (OFF) an in-memory CAS table of the USCB data (in addition to the SAS data set that is created). When GOCAS=ON, the SASEUSCB engine assumes that there is an active CAS session running when it executes the LIBNAME statement. By default, GOCAS=OFF.

OUTJSON=uscb_json_file_name

specifies the name of the JSON file (which is downloaded to the SAS Work library) and the name of the SAS data set that is created when the JSON data are read into SAS. The SAS data set is stored in the folder that is specified by *physical-name*.

PROXY='uscb_proxyserver'

specifies which proxy server to use. This option is not required. The specified proxy server is used only when a connection-refused or connection-timed-out error occurs. For *uscb_proxyserver*, specify the server's HTTP address, followed by a colon and the port number, and enclose that string in double quotation marks; for example, PROXY="http://inetgw.unx.sas.com:8118". (See also the [CONNECT=](#) option.)

SETNAME=uscb_setname**DATASETNAME=uscb_setname****DATASET=uscb_setname****SETID=uscb_setname**

specifies the name of the USCB data set to retrieve data from. [Table 25.3](#) shows the list of available USCB data sets. **NOTE:** This version of the SASEUSCB interface engine supports only the data set that is shown in the first row of [Table 25.3](#) (EITS_ADVM3).

STARTDATE='uscb_startdate'

specifies the starting date of the time series in the format 'YYYY', the ending quarter and year for quarterly data in the format 'YYYY-Qx' (where x = 1, 2, 3, or 4), or the ending month and year for monthly data in the format 'YYYY-nn' (where nn = 01, 02, ..., 10, 11, or 12).

USER='uscb_user'**APIKEY='uscb_user'****REGISTRATIONKEY='uscb_user'**

specifies the unique user registration key, enclosed in quotation marks, that the USCB assigns to the user (subscriber) to allow access to the data on the USCB website. The USCB access key is a 40-character alphanumeric lowercase string that you can request. To obtain your own unique API key, visit the USCB website at the following URL: <https://www.census.gov/data/developers.html>. Then click on the **Request a Key** box at the lower left of that page. Alternatively, you can go directly to the following site: https://api.census.gov/data/key_signup.html.

Table 25.2 USCB Geo-level Region Codes

Geo-level Code	Description	Geo-level Code	Description
US	United States Total	MO	Missouri
NO	Northeast	MT	Montana
MW	Midwest	MW	Midwest
SO	South	NE	Nebraska
WE	West	NV	Nevada
AK	Alaska	NH	New Hampshire
AL	Alabama	NJ	New Jersey
AR	Arkansas	NM	New Mexico
AZ	Arizona	NY	New York
CA	California	NC	North Carolina
CO	Colorado	ND	North Dakota
CT	Connecticut	NE	Northeast
DE	Delaware	OH	Ohio
DC	District of Columbia	OK	Oklahoma
FL	Florida	OR	Oregon
GA	Georgia	PA	Pennsylvania
HI	Hawaii	RI	Rhode Island
ID	Idaho	SC	South Carolina
IL	Illinois	SD	South Dakota
IN	Indiana	SO	South
IA	Iowa	TN	Tennessee
KS	Kansas	TX	Texas
KY	Kentucky	UT	Utah
LA	Louisiana	VT	Vermont
ME	Maine	VA	Virginia
MD	Maryland	WA	Washington
MA	Massachusetts	WE	West
MI	Michigan	WV	West Virginia
MN	Minnesota	WI	Wisconsin
MS	Mississippi	WY	Wyoming

Details: SASEUSCB Interface Engine

The SASEUSCB interface engine enables you to access time series data from the USCB data sets shown in Table 25.3. To specify a USCB data set, use one of the supported data set names that are listed in the table.

NOTE: This version of the SASEUSCB engine supports only the ADVDM3 data set, the BFS data set, and the FTD data set. The Advance Report on Durable Goods, Manufacturers' Shipments, Inventories, and Orders (ADVDM3) data set is listed in the first row of the table. The Business Formation Statistics (BFS) data set is listed in the fourth row of the table. The International Trade in Goods and Services (FTD) data set is listed in the fifth row of the table.

To access the data set (the data set name is EITS_ADVM3), specify SETNAME=<data set name>, as shown in the SASEUSCB LIBNAME statement in the section “Getting Started: SASEUSCB Interface Engine” on page 1699.

Table 25.3 USCB Data Sets

Data Set	Starting Date	Long Name
EITS_ADVM3	1992	Advance Report on Durable Goods, Manufacturers' Shipments, Inventories, and Orders
EITS_ADVMRTS	1992	Advance Retail Inventories
EITS_ADVMWTS	1992	Advance Wholesale Inventories
EITS_BFS	2004	Business Formation Statistics
EITS_FTD	1992	International Trade in Goods and Services
EITS_HV	1956	Housing Vacancies and Homeownership
EITS_M3	1992	Manufacturers' Shipments, Inventories, and Orders
EITS_MARTS	1992	Advance Monthly Sales for Retail and Food Service
EITS_MHS	1959	Manufactured Homes Survey
EITS_MHS2	1959	Manufactured Housing Survey
EITS_MRTS	1992	Monthly Retail Trade and Food Services
EITS_MTIS	1992	Manufacturing and Trade Inventories and Services
EITS_MWTS	1992	Monthly Wholesale Trade: Sales and Inventories
EITS_QFR	2000	Quarterly Financial Report Survey
EITS_QPR	1968	Quarterly Survey of Public Pensions
EITS_QSS	2003	Quarterly Services Survey
EITS_QTAX	1992	Quarterly Summary of State and Local Taxes
EITS_RESCONST	1959	New Residential Construction
EITS_RESSALES	1963	New Home Sales
EITS_VIP	2002	Construction Spending

Data Elements Reference: SASEUSCB Interface Engine

Table 25.4 gives useful information about the supported data set: reporting frequency and allowed parameter values for required options. The default geo-level code, which identifies the geographic region that the data apply to, is US (United States Total). **NOTE:** This version of the SASEUSCB engine supports only the ADVM3 data set, the BFS data set, and the FTD data set.

Table 25.4 Supported USCB Data Sets

Data Set	Reporting Frequency	Required Options
ADVM3	Monthly	FOR=US; START= and END= <YYYYnn>
BFS	Quarterly	FOR=US; START= and END= <YYYYQn>
FTD	Monthly	START= and END= <YYYYnn>

Table 25.5 shows where you can find a summary of USCB data set details. To obtain a list of the queries that you can use for a data set, see the tables for that data set in the second column. Each USCB data set has its own list of allowed queries.

Table 25.5 USCB Data Set Details

Data Set	Where to Find Details for Specifying Options
ADVM3	Table 25.7 and Table 25.8
BFS	Table 25.9 , Table 25.10 , and Table 25.11
FTD	Table 25.12 and Table 25.13

The CATEGORY=, DATATYPE=, and GEO= options enable you to select the data to keep in your SAS output data set. Table 25.6 shows the available options for the supported data set and where to find the allowed values for each option. The GEO= option is not allowed for the ADVM3 data set or the FTD data set. **NOTE:** This version of the SASEUSCB engine supports only the ADVM3 data set, the BFS data set, and the FTD data set.

Table 25.6 Values for USCB Data Set Options

Data Set	Option	Option Value Table
ADVM3	CATEGORY=	Table 25.7
	DATATYPE=	Table 25.8
BFS	CATEGORY=	Table 25.9
	DATATYPE=	Table 25.10
	GEO=	Table 25.11
FTD	CATEGORY=	Table 25.12
	DATATYPE=	Table 25.13

ADVM3 Data Set: Advance Report on Durable Goods, Manufacturers' Shipments, Inventories, and Orders

Table 25.7 lists the codes for the supported categories in the ADVM3 data set and the description of each category.

Table 25.7 ADVM3 Data Set Categories

Category Code	Description
MDM	Durable Goods
DXT	Durable Goods Excluding Transportation
DXD	Durable Goods Excluding Defense
MTU	Manufacturing with Unfilled Orders
31S	Primary Metals
32S	Fabricated Metal Products
33S	Machinery
34S	Computers and Electronic Products
CRP	Computers and Related Products
34X	Communications Equipment
35S	Electrical Equipment, Appliances, and Components
36S	Transportation Equipment
MVP	Motor Vehicles and Parts
NAP	Nondefense Aircraft and Parts
DAP	Defense Aircraft and Parts
ODG	All Other Durable Goods
TCG	Capital Goods
NDE	Nondefense Capital Goods
NXA	Nondefense Capital Goods Excluding Aircraft
DEF	Defense Capital Goods

Table 25.8 lists the codes for the supported data types in the ADVM3 data set and the description of each data type. The description includes the unit of measurement in millions of dollars (MLN\$) or percentage change monthly (PCT).

Table 25.8 ADVM3 Data Set Data Types

Data Type Code	Description
VS	Value of Shipments (MLN\$)
NO	New Orders (MLN\$)
UO	Unfilled Orders (MLN\$)
TI	Total Inventories (MLN\$)
MPCVS	Value of Shipments Percent Change Monthly (PCT)
MPCNO	New Orders Percent Change Monthly (PCT)
MPCUO	Unfilled Orders Percent Change Monthly (PCT)
MPCTI	Total Inventories Percent Change Monthly (PCT)

BFS Data Set: Business Formation Statistics

Table 25.9 lists the codes for the supported categories in the BFS data set and the description of each category.

Table 25.9 BFS Data Set Categories

Category Code	Description
TOTAL	Total for All NAICS
NAICS11	Agriculture
NAICS21	Mining
NAICS22	Utilities
NAICS23	Construction
NAICSMNF	Manufacturing
NAICS42	Wholesale Trade
NAICSRET	Retail Trade
NAICSTW	Transportation and Warehousing
NAICS51	Information
NAICS52	Finance and Insurance
NAICS53	Real Estate
NAICS54	Professional Services
NAICS55	Management of Companies
NAICS56	Administrative and Support
NAICS61	Educational Services
NAICS62	Health Care and Social Assistance
NAICS71	Arts and Entertainment
NAICS72	Accommodation and Food Services
NAICS81	Other Services
NAICS92	Public Administration
NONAICS	No NAICS Assigned

Table 25.10 lists the codes for the supported data types in the BFS data set and the description of each data type. The description includes the unit of measurement in number of units (UNITS).

Table 25.10 BFS Data Set Data Types

Data Type Code	Description
BA_BA	Business Applications (UNITS)
BA_CBA	Business Applications from Corporations (UNITS)
BA_HBA	High-Propensity Business Applications (UNITS)
BA_WBA	Business Applications with Planned Wages (UNITS)
BF_BF4Q	Business Formations within Four Quarters (UNITS)
BF_BF8Q	Business Formations within Eight Quarters (UNITS)
BF_PBF4Q	Projected Business Formations within Four Quarters (UNITS)
BF_PBF8Q	Projected Business Formations within Eight Quarters (UNITS)
BF_SBF4Q	Spliced Business Formations within Four Quarters (UNITS)
BF_SBF8Q	Spliced Business Formations within Eight Quarters (UNITS)

Table 25.10 *continued*

Data Type Code	Description
BF_DUR4Q	Average Duration (in Quarters) from Business Application to Formation within Four Quarters (UNITS)
BF_DUR8Q	Average Duration (in Quarters) from Business Application to Formation within Eight Quarters (UNITS)

Table 25.11 lists the supported geographic regions (using geo-level codes) for the BFS data set and the description of each region. Note that for states, the geo-level codes are the same as the standard two-letter postal codes, such as AL for Alabama.

Table 25.11 BFS Data Set Geo-level Codes

Geo-level Code	Description	Geo-level Code	Description
US	United States Total	MN	Minnesota
*	All Geographies	MS	Mississippi
NO	Northeast	MO	Missouri
MW	Midwest	MT	Montana
SO	South	NE	Nebraska
WE	West	NV	Nevada
AK	Alaska	NH	New Hampshire
AL	Alabama	NJ	New Jersey
AR	Arkansas	NM	New Mexico
AZ	Arizona	NY	New York
CA	California	NC	North Carolina
CO	Colorado	ND	North Dakota
CT	Connecticut	OH	Ohio
DE	Delaware	OK	Oklahoma
DC	District of Columbia	OR	Oregon
FL	Florida	PA	Pennsylvania
GA	Georgia	RI	Rhode Island
HI	Hawaii	SC	South Carolina
ID	Idaho	SD	South Dakota
IL	Illinois	TN	Tennessee
IN	Indiana	TX	Texas
IA	Iowa	UT	Utah
KS	Kansas	VT	Vermont
KY	Kentucky	VA	Virginia
LA	Louisiana	WA	Washington
ME	Maine	WV	West Virginia
MD	Maryland	WI	Wisconsin
MA	Massachusetts	WY	Wyoming
MI	Michigan		

FTD Data Set: International Trade in Goods and Services

Table 25.12 lists the codes for the supported categories in the FTD data set and the description of each category.

Table 25.12 FTD Data Set Categories

Category Code	Description
BOPG	Balance of Payment, Goods
BOPGS	Balance of Payment, Goods and Services

Table 25.13 lists the codes for the supported data types in the FTD data set and the description of each data type. The description includes the unit of measurement in millions of dollars (MLN\$).

Table 25.13 FTD Data Set Data Types

Data Type Code	Description
BAL	Balance (MLN\$)
EXP	Exports (MLN\$)
IMP	Imports (MLN\$)

SAS Output Data Set

You can use the SAS DATA step to write the selected USCB data to a SAS data set. This enables you to use SAS software to easily analyze the data. If you specify the name of the output data set in the DATA statement, the engine supervisor creates a SAS data set that has the specified name in either the SAS Work library or, if specified, the SAS User library. The SASEUSCB engine also keeps a copy of the output data set in the location that is specified by the *physical-name* in the LIBNAME *libref* SASEUSCB statement, which is described in the section “[The LIBNAME libref SASEUSCB Statement](#)” on page 1702.

The contents of the SAS data set include the date of each observation and the name of each series that is read from the USCB data source. The series names for each Economic Indicator Time Series (EITS) data set are made up of four parts, separated by underscores (‘_’): the category code, the data type code, the geo-level code, and an ‘S’ for seasonal data or a ‘U’ for nonseasonal data. When there is only one geo-level code, the series name might not include the geo-level code in the series name. The default geographic region for a time series name that does not include the geo-level code is the United States. For example, NXA_TI_S is a series name that has the following meaning: a seasonal time series in the category NXA, with the data type TI, for the United States. (See [Table 25.7](#) and [Table 25.8](#) for a list of the currently supported USCB data set’s categories and data types.)

You can use the PRINT and CONTENTS procedures to print your output data set and its contents. Alternatively, you can view your SAS output observations by opening the chosen output data set in a SAS Explorer window. You can also use the SQL procedure with your SASEUSCB libref to create a custom view of the data.

SAS OUTJSON File

The SAS JSON data (in JSON format) that the USCB website retrieves are placed in a file that you name in the OUTJSON= option. The SAS JSON data file is placed in the SAS Work directory. But the SAS data set that is created by reading the JSON data into SAS is placed in the location that is specified by the *physical-name* in the LIBNAME *libref* SASEUSCB statement, which is described in the section “Data Elements Reference: SASEUSCB Interface Engine” on page 1707.

Examples: SASEUSCB Interface Engine

Example 25.1: Retrieving (ADVM3) Data for Category MDM (Durable Goods)

This example shows how to retrieve data from the Advance Report on Durable Goods Manufacturers’ Shipments, Inventories, and Orders data set from the US Census Bureau website. You can view these data at the following web page:

[https://www.census.gov/econ/currentdata/dbsearch?programCode=M3ADV&startYear=2020&endYear=2022&categories\[\]=MDM&dataType=NO&geoLevel=US&adjusted=1¬Adjusted=1&errorData=0](https://www.census.gov/econ/currentdata/dbsearch?programCode=M3ADV&startYear=2020&endYear=2022&categories[]=MDM&dataType=NO&geoLevel=US&adjusted=1¬Adjusted=1&errorData=0)

The following SASEUSCB LIBNAME statement specifies the libref (USCB), followed by the engine name (SASEUSCB) and the physical path where the retrieved USCB data are to be stored. The SETNAME= option specifies the name of the data set to retrieve data from. In this example, SETNAME=EITS_ADVM3 retrieves data from the Advance Report on Durable Goods Manufacturers’ Shipments, Inventories, and Orders data set. The CATEGORY=MDM option selects the time series for the durable goods category (MDM). See Table 25.7 for a list of available categories and Table 25.8 for a list of available data types in this data set. Each Census Bureau data set has its own specific list of categories and data types to choose from. If the CATEGORY= option is omitted, then all the categories of data are selected for retrieval. The USER= option specifies your unique USCB API key. Without a user API key (which you can obtain free of charge), you cannot retrieve data from the US Census Bureau. The OUTJSON= option names the file where the JSON data are stored. The output data set is created by using the OUTJSON= option and placed in the folder that you specify in the SASEUSCB LIBNAME statement. The date range of the data starts with 2004-09, as specified by the STARTDATE= option. The ENDDATE= option specifies the last period (2005-04) to include in the range of data to retrieve. The FORMAT= option specifies that the data are to be retrieved in the JSON format. Output 25.1.1 shows the data that are retrieved; only the time series in the durable goods category (MDM) are shown.

```
options validvarname=any;

title 'Retrieve ADVM3 Advance Report for Durable Goods Category Statistics';
libname uscb saseuscb "%sysget(USCB)"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    setname=EITS_ADVM3
    for='US:*'
    category=MDM
```

```

outjson=uscbex01
startdate='2004-09'
enddate='2005-04'
format=json;

data myAM3;
  set uscb.uscbex01;
run;

proc contents data=myAM3; run;
proc print data=myAM3; run;

```

Output 25.1.1 Durable Goods (CATEGORY=MDM) in the ADVM3 Economic Indicators Data Set

Retrieve ADVM3 Advance Report for Durable Goods Category Statistics

Obs	date	MDM_MPCNO_S	MDM_MPCTI_S	MDM_MPCUO_S	MDM_MPCVS_S	MDM_NO_S
1	2004-09	2.6	0.4	0.8	0.3	188444
2	2004-10	-1.7	0.8	0.0	0.7	185187
3	2004-11	4.0	1.1	1.2	0.3	192545
4	2004-12	0.5	0.2	0.7	2.2	193578
5	2005-01	0.7	1.3	0.2	1.6	194875
6	2005-02	0.8	1.1	0.9	-1.0	196475
7	2005-03	-5.0	0.8	-0.5	-1.1	186608
8	2005-04	4.4	0.3	-0.1	3.1	194847

Obs	MDM_TI_S	MDM_UO_S	MDM_VS_S	MDM_MPCNO_U	MDM_MPCTI_U	MDM_MPCUO_U
1	259354	548017	190135	6.8	-0.9	-0.1
2	261417	547928	191474	-5.2	1.5	-0.2
3	264380	554750	192033	0.6	1.2	1.2
4	264895	558457	196330	6.9	-2.7	1.9
5	268396	559852	199389	-11.7	4.1	1.4
6	271422	564784	197451	6.4	2.4	1.3
7	273699	562236	195296	11.0	-0.5	0.3
8	274498	561491	201311	-9.0	1.1	-0.2

Obs	MDM_MPCVS_U	MDM_NO_U	MDM_TI_U	MDM_UO_U	MDM_VS_U
1	5.8	197167	257887	539876	204079
2	-4.8	186918	261805	538716	194247
3	-3.1	188131	264821	545152	188136
4	4.8	201172	257622	555322	197185
5	-10.7	177594	268243	562949	176001
6	6.6	188951	274594	570389	187565
7	14.4	209708	273345	571891	214567
8	-7.8	190899	276320	570621	197842

Example 25.2: Retrieving Business Formation Statistics (BFS) Data for Business Applications (BA_BA) Data Type

This example shows how to retrieve Business Formation Statistics (BFS) data from the US Census Bureau website. You can view these data at the following web page:

https://www.census.gov/econ/currentdata/dbsearch?program=BFS&startYear=2004&endYear=2022&categories=TOTAL&dataType=BA_BA&geoLevel=US&adjusted=1¬Adjusted=1&errorData=0

The following SASEUSCB LIBNAME statement specifies the libref (USCB), followed by the engine name (SASEUSCB) and the physical path where the retrieved USCB data are to be stored. The SETNAME= option specifies the name of the data set to retrieve data from. In this example, SETNAME=EITS_BFS requests data from the Business Formation Statistics data set. The USER= option specifies your unique USCB API key. Without a user API key (which you can obtain free of charge), you cannot retrieve data from the US Census Bureau. The FOR='US:*' option requests data for all the US states, and the GEO='*' option specifies all geographic levels. The DATATYPE=BA_BA option specifies the business applications data type. If you want to include all data types, omit this option. See Table 25.10 for available data types to use when you specify the DATATYPE= option for the BFS data set. See Table 25.9 for a complete list of available categories. The OUTJSON= BFSTX option names the file where the JSON data are stored. The output data set is created by using the name that is specified in the OUTJSON= option and placed in the folder that is specified in the SASEUSCB LIBNAME statement. The date range of the data starts with September 2007, as specified by the STARTDATE='2007-09' option. The ENDDATE='2008-04' option specifies the last month to include in the range of data to retrieve (April 2008). The FORMAT=JSON option specifies that the data are to be retrieved in the JSON format. Output 25.2.1 shows the BFS data that are retrieved. Time series whose names end in '_S' contain seasonal data, and time series whose names end in '_U' contain nonseasonal data.

```
options validvarname=any;

title 'Retrieve Business Formation Statistics for Business Applications,
      DATATYPE=BA_BA';
libname uscb saseuscb "%sysget(USCB)"
      user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
      setname=EITS_BFS
      for='US:*'
      geo='*'
      datatype=BA_BA
      startdate='2007-09'
      enddate='2008-04'
      outjson=uscbex02
      format=json;

data myBFS;
  set uscb.uscbex02 ;
run;

proc contents data=myBFS; run;
proc print data=myBFS; run;
```

Output 25.2.1 Retrieve Business Formation Statistics (BFS) Economic Indicator Time Series Data for Business Applications

**Retrieve Business Formation Statistics for Business Applications,
DATATYPE=BA_BA**

Obs	date	NAICS11_BA_BA_S.US	NAICS21_BA_BA_S.US	NAICS22_BA_BA_S.US	NAICS23_BA_BA_S.US
1	2007-09	2769	753	197	26830
2	2007-10	2820	817	220	28460

Obs	NAICS42_BA_BA_S.US	NAICS51_BA_BA_S.US	NAICS52_BA_BA_S.US	NAICS53_BA_BA_S.US
1	8165	4791	12769	17516
2	8869	5087	13535	14825

Obs	NAICS54_BA_BA_S.US	NAICS55_BA_BA_S.US	NAICS56_BA_BA_S.US	NAICS61_BA_BA_S.US
1	29773	1514	16351	3322
2	31011	1389	16965	3513

Obs	NAICS62_BA_BA_S.US	NAICS71_BA_BA_S.US	NAICS72_BA_BA_S.US	NAICS81_BA_BA_S.US
1	14322	7613	11872	19718
2	14569	8084	12517	20189

Obs	NAICSMNF_BA_BA_S.US	NAICSRET_BA_BA_S.US	NAICSTW_BA_BA_S.US	NONAICS_BA_BA_S.US
1	5182	27728	12281	2597
2	5427	30489	11674	2170

Obs	TOTAL_BA_BA_S.US	NAICS11_BA_BA_U.US	NAICS21_BA_BA_U.US	NAICS22_BA_BA_U.US
1	226061	1949	644	181
2	232630	1988	698	202

Obs	NAICS23_BA_BA_U.US	NAICS42_BA_BA_U.US	NAICS51_BA_BA_U.US	NAICS52_BA_BA_U.US
1	22398	7230	4208	10918
2	24018	8100	4428	12092

Obs	NAICS53_BA_BA_U.US	NAICS54_BA_BA_U.US	NAICS55_BA_BA_U.US	NAICS56_BA_BA_U.US
1	14392	25410	1255	13553
2	12601	26938	1142	14506

Obs	NAICS61_BA_BA_U.US	NAICS62_BA_BA_U.US	NAICS71_BA_BA_U.US	NAICS72_BA_BA_U.US
1	3027	12292	7003	10383
2	3079	12838	6974	11296

Obs	NAICS81_BA_BA_U.US	NAICSMNF_BA_BA_U.US	NAICSRET_BA_BA_U.US	NAICSTW_BA_BA_U.US
1	16882	4513	24880	11226
2	17365	4802	27780	10405

Obs	NONAICS_BA_BA_U.US	TOTAL_BA_BA_U.US
1	2046	194390
2	1371	202623

Output 25.2.1 *continued***Retrieve Business Formation Statistics for Business Applications,
DATATYPE=BA_BA**

Obs	date	NAICS11_BA_BA_S.US	NAICS21_BA_BA_S.US	NAICS22_BA_BA_S.US	NAICS23_BA_BA_S.US
3	2007-11	2657	731	154	25285
4	2007-12	3216	776	181	29121
Obs	NAICS42_BA_BA_S.US	NAICS51_BA_BA_S.US	NAICS52_BA_BA_S.US	NAICS53_BA_BA_S.US	
3	8418	4580	12492	12635	
4	9083	5369	14009	13776	
Obs	NAICS54_BA_BA_S.US	NAICS55_BA_BA_S.US	NAICS56_BA_BA_S.US	NAICS61_BA_BA_S.US	
3	28677	1177	16185	3391	
4	34342	1470	18154	3807	
Obs	NAICS62_BA_BA_S.US	NAICS71_BA_BA_S.US	NAICS72_BA_BA_S.US	NAICS81_BA_BA_S.US	
3	13775	7347	11764	18723	
4	16228	8131	13675	21606	
Obs	NAICSMNF_BA_BA_S.US	NAICSRET_BA_BA_S.US	NAICSTW_BA_BA_S.US	NONAICS_BA_BA_S.US	
3	4957	27638	10853	2047	
4	5594	31600	12784	2634	
Obs	TOTAL_BA_BA_S.US	NAICS11_BA_BA_U.US	NAICS21_BA_BA_U.US	NAICS22_BA_BA_U.US	
3	213484	2228	826	176	
4	245556	2605	722	170	
Obs	NAICS23_BA_BA_U.US	NAICS42_BA_BA_U.US	NAICS51_BA_BA_U.US	NAICS52_BA_BA_U.US	
3	25498	8865	4835	13795	
4	21092	7461	3989	13267	
Obs	NAICS53_BA_BA_U.US	NAICS54_BA_BA_U.US	NAICS55_BA_BA_U.US	NAICS56_BA_BA_U.US	
3	13308	30612	1268	16044	
4	12038	27118	1428	12849	
Obs	NAICS61_BA_BA_U.US	NAICS62_BA_BA_U.US	NAICS71_BA_BA_U.US	NAICS72_BA_BA_U.US	
3	3460	14558	7441	12771	
4	2841	12964	6096	10410	
Obs	NAICS81_BA_BA_U.US	NAICSMNF_BA_BA_U.US	NAICSRET_BA_BA_U.US	NAICSTW_BA_BA_U.US	
3	19169	5212	29976	11067	
4	15734	4601	23327	9558	
Obs	NONAICS_BA_BA_U.US	TOTAL_BA_BA_U.US			
3	1588	222697			
4	1673	189943			

Output 25.2.1 *continued*

**Retrieve Business Formation Statistics for Business Applications,
DATATYPE=BA_BA**

Obs	date	NAICS11_BA_BA_S.US	NAICS21_BA_BA_S.US	NAICS22_BA_BA_S.US	NAICS23_BA_BA_S.US
5	2008-01	2694	684	140	25584
6	2008-02	2628	752	193	24923
Obs	NAICS42_BA_BA_S.US	NAICS51_BA_BA_S.US	NAICS52_BA_BA_S.US	NAICS53_BA_BA_S.US	
5	8292	4686	12511	11551	
6	8411	4607	11968	11714	
Obs	NAICS54_BA_BA_S.US	NAICS55_BA_BA_S.US	NAICS56_BA_BA_S.US	NAICS61_BA_BA_S.US	
5	27669	1267	15066	3349	
6	28269	1205	15698	3213	
Obs	NAICS62_BA_BA_S.US	NAICS71_BA_BA_S.US	NAICS72_BA_BA_S.US	NAICS81_BA_BA_S.US	
5	13719	7082	11642	17564	
6	14170	7467	11766	18331	
Obs	NAICSMNF_BA_BA_S.US	NAICSRET_BA_BA_S.US	NAICSTW_BA_BA_S.US	NONAICS_BA_BA_S.US	
5	4927	27705	10641	2074	
6	5005	27442	10680	2021	
Obs	TOTAL_BA_BA_S.US	NAICS11_BA_BA_U.US	NAICS21_BA_BA_U.US	NAICS22_BA_BA_U.US	
5	208848	3223	651	129	
6	210463	4559	935	222	
Obs	NAICS23_BA_BA_U.US	NAICS42_BA_BA_U.US	NAICS51_BA_BA_U.US	NAICS52_BA_BA_U.US	
5	21626	7068	3900	11202	
6	34018	10808	6038	15196	
Obs	NAICS53_BA_BA_U.US	NAICS54_BA_BA_U.US	NAICS55_BA_BA_U.US	NAICS56_BA_BA_U.US	
5	10719	27794	1046	13193	
6	15059	38743	1465	21394	
Obs	NAICS61_BA_BA_U.US	NAICS62_BA_BA_U.US	NAICS71_BA_BA_U.US	NAICS72_BA_BA_U.US	
5	2726	13361	5935	9598	
6	4189	20065	9913	14622	
Obs	NAICS81_BA_BA_U.US	NAICSMNF_BA_BA_U.US	NAICSRET_BA_BA_U.US	NAICSTW_BA_BA_U.US	
5	15622	4413	22805	9984	
6	25125	6605	35528	13769	
Obs	NONAICS_BA_BA_U.US	TOTAL_BA_BA_U.US			
5	1068	186063			
6	1940	280193			

Output 25.2.1 *continued***Retrieve Business Formation Statistics for Business Applications,
DATATYPE=BA_BA**

Obs	date	NAICS11_BA_BA_S.US	NAICS21_BA_BA_S.US	NAICS22_BA_BA_S.US	NAICS23_BA_BA_S.US
7	2008-03	2713	797	248	25960
8	2008-04	2787	771	240	25393
Obs	NAICS42_BA_BA_S.US	NAICS51_BA_BA_S.US	NAICS52_BA_BA_S.US	NAICS53_BA_BA_S.US	
7	8779	4958	12794	12081	
8	9027	4934	12593	11707	
Obs	NAICS54_BA_BA_S.US	NAICS55_BA_BA_S.US	NAICS56_BA_BA_S.US	NAICS61_BA_BA_S.US	
7	29824	1232	16166	3474	
8	29931	1242	16114	3588	
Obs	NAICS62_BA_BA_S.US	NAICS71_BA_BA_S.US	NAICS72_BA_BA_S.US	NAICS81_BA_BA_S.US	
7	14696	7937	12434	19027	
8	14875	7845	12363	19446	
Obs	NAICSMNF_BA_BA_S.US	NAICSRET_BA_BA_S.US	NAICSTW_BA_BA_S.US	NONAICS_BA_BA_S.US	
7	5381	28716	11170	1942	
8	5029	28625	10698	1952	
Obs	TOTAL_BA_BA_S.US	NAICS11_BA_BA_U.US	NAICS21_BA_BA_U.US	NAICS22_BA_BA_U.US	
7	220328	3662	855	227	
8	219159	3183	775	220	
Obs	NAICS23_BA_BA_U.US	NAICS42_BA_BA_U.US	NAICS51_BA_BA_U.US	NAICS52_BA_BA_U.US	
7	29012	9296	5362	13148	
8	27411	8973	5120	12390	
Obs	NAICS53_BA_BA_U.US	NAICS54_BA_BA_U.US	NAICS55_BA_BA_U.US	NAICS56_BA_BA_U.US	
7	12715	31617	1370	18291	
8	11903	29764	1248	17814	
Obs	NAICS61_BA_BA_U.US	NAICS62_BA_BA_U.US	NAICS71_BA_BA_U.US	NAICS72_BA_BA_U.US	
7	3633	15109	8850	12839	
8	3693	14408	8198	12648	
Obs	NAICS81_BA_BA_U.US	NAICSMNF_BA_BA_U.US	NAICSRET_BA_BA_U.US	NAICSTW_BA_BA_U.US	
7	21021	5724	30933	11884	
8	20179	5133	29580	11179	
Obs	NONAICS_BA_BA_U.US	TOTAL_BA_BA_U.US			
7	1837	237385			
8	2346	226165			

You can subset your results by adding a KEEP= or DROP= option to your DATA statement as follows:


```
data myBFS( keep= date TOTAL:);
  set uscb.uscbex02 ;
run;
```

The results from the KEEP= option in the DATA statement return two time series, 'TOTAL_BA_BA_S.US'n and 'TOTAL_BA_BA_U.US'n, along with the date of each observation. Each time series is named by concatenating the category code, the data type code, and the seasonality code, using the underscore as the concatenation character, followed by the suffix '.US' (for United States).

Example 25.3: Retrieving International Trade in Goods and Services (FTD) Data for All Data Types

This example shows how to retrieve International Trade in Goods and Services (FTD) data from the US Census Bureau website. You can view these data at the following web page:

<https://www.census.gov/econ/currentdata/dbsearch?programCode=FTD>

The following SASEUSCB LIBNAME statement specifies the libref (USCB), followed by the engine name (SASEUSCB) and the physical path where the retrieved USCB data are to be stored. The SETNAME= option specifies the name of the data set to retrieve data from. In this example, SETNAME=EITS_FTD requests data from the International Trade in Goods and Services data set. The USER= option specifies your unique USCB API key. Without a user API key (which you can obtain free of charge), you cannot retrieve data from the US Census Bureau. The FOR='US:*' option requests data for all of the United States. The DATATYPE= option is omitted because the example includes all data types in the retrieved data. The CATEGORY= option is also omitted because the example includes all categories in the retrieved data. See [Table 25.12](#) for a complete list of available categories, and see [Table 25.13](#) for a complete list of available data types. The OUTJSON=USCBEX03 option names the file where the JSON data are stored. The output data set is created by using the name that is specified in the OUTJSON= option and placed in the folder that is specified in the SASEUSCB LIBNAME statement. The date range of the data starts with September 2021, as specified by the STARTDATE='2021-09' option. The ENDDATE='2022-04' option specifies the last month to include in the range of data to retrieve (April 2022). The FORMAT=JSON option specifies that the data are to be retrieved in the JSON format. [Output 25.3.1](#) shows the FTD data that are retrieved. Time series whose names end in '_S' contain seasonal data, and time series whose names end in '_U' contain nonseasonal data.

```
options validvarname=any;

title 'Retrieve International Trade in Goods and Services Data';
libname uscb saseuscb "%sysget(USCB) "
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  setname=EITS_FTD
  for='US:*'
  startdate='2021-09'
  enddate='2022-04'
  outjson=uscbex03
  format=json;

data myFT2D;
```

```

set uscb.uscbex03 ;
run;

proc contents data=myFT2D; run;
proc print data=myFT2D; run;

```

Output 25.3.1 Retrieve International Trade in Goods and Services (FTD) Data

Retrieve International Trade in Goods and Services Data

Obs	date	BOPGS_BAL_S	BOPGS_EXP_S	BOPGS_IMP_S	BOPG_BAL_U	BOPG_EXP_U	BOPG_IMP_U
1	2021-09	-77095	213039	290134	-103031	143002	246033
2	2021-10	-66942	227955	294897	-88071	164954	253025
3	2021-11	-80153	230642	310795	-103963	157499	261462
4	2021-12	-80792	235613	316405	-98430	160406	258836
5	2022-01	-85565	231087	316652	-99946	149224	249170
6	2022-02	-87070	235969	323039	-83863	152154	236017
7	2022-03	-101914	246646	348560	-116344	181849	298193
8	2022-04	-85376	254374	339750	-98762	176115	274876

References

- United States Census Bureau (2020). *Economic Indicator Time Series API User Guide*. https://www2.census.gov/data/api-documentation/EITS_API_User_Guide_Dec2020.pdf.
- United States Census Bureau (2021a). *Census Data API User Guide*. <https://www.census.gov/data/developers/guidance/api-user-guide.html>.
- United States Census Bureau (2021b). “Datasets in the Census Data API.” <https://api.census.gov/data.html>.

Chapter 26

SEVSELECT Procedure

Contents

Overview: SEVSELECT Procedure	1722
Using CAS Sessions and CAS Engine Librefs	1724
Getting Started: SEVSELECT Procedure	1725
A Simple Example of Fitting Predefined Distributions	1725
An Example with Left-Truncation and Right-Censoring	1730
An Example of Modeling Regression Effects	1735
Syntax: SEVSELECT Procedure	1739
Functional Summary	1740
PROC SEVSELECT Statement	1744
BY Statement	1756
CLASS Statement	1757
DISPLAY Statement	1757
DISPLAYOUT Statement	1758
DIST Statement	1759
EFFECT Statement	1761
LOSS Statement	1763
NLOPTIONS Statement	1766
OUTPUT Statement	1766
OUTSCORELIB Statement	1770
RESTRICT Statement	1772
SCALEMODEL Statement	1774
SELECTION Statement	1776
WEIGHT Statement	1778
Programming Statements	1778
Details: SEVSELECT Procedure	1779
Predefined Distributions	1779
Censoring and Truncation	1789
Parameter Estimation Method	1791
Parameter Initialization	1793
Estimating Regression Effects	1794
Empirical Distribution Function Estimation Methods	1799
Statistics of Fit	1806
Defining a Severity Distribution Model with the FCMP Procedure	1811
Predefined Utility Functions	1824
Scoring Functions	1829
Custom Objective Functions	1836

Input Data Tables	1839
Output Data Tables	1840
Displayed Output	1845
ODS Graphics	1847
Examples: SEVSELECT Procedure	1852
Example 26.1: Defining a Model for the Gaussian Distribution with a Scale Parameter	1852
Example 26.2: Defining a Model for Mixed-Tail Distributions	1856
Example 26.3: Fitting a Scaled Tweedie Model with Regressors	1863
Example 26.4: Fitting Distributions to Interval-Censored Data	1868
Example 26.5: Defining a Finite Mixture Model That Has a Scale Parameter	1870
Example 26.6: Scale Regression with Rich Regression Effects	1876
Example 26.7: Scale Regression Model Selection	1881
Example 26.8: Estimating Parameters Using the Cramér–von Mises Estimator	1892
Example 26.9: Predicting Mean and Value-at-Risk by Using Scoring Functions	1893
References	1898

Overview: SEVSELECT Procedure

The SEVSELECT procedure estimates parameters of any arbitrary continuous probability distribution that is used to model the magnitude (severity) of a continuous-valued event of interest. Examples of such events include loss amounts paid by an insurance company and demand of a product as depicted by its sales. PROC SEVSELECT is especially useful when the severity of an event does not follow typical distributions (such as the normal distribution) that are often assumed by standard statistical methods.

PROC SEVSELECT provides a default set of probability distribution models that includes the Burr, exponential, gamma, generalized Pareto, inverse Gaussian (Wald), lognormal, Pareto, Tweedie, and Weibull distributions. In the simplest form, you can estimate the parameters of any of these distributions by using a list of severity values that are recorded in a data table. You can optionally group the values by a set of BY variables. PROC SEVSELECT computes the estimates of the model parameters, their standard errors, and their covariance structure by using the maximum likelihood method for each of the BY groups.

PROC SEVSELECT can fit multiple distributions at the same time and choose the best distribution according to a selection criterion that you specify. You can use seven different statistics of fit as selection criteria. They are log likelihood, Akaike's information criterion (AIC), corrected Akaike's information criterion (AICC), Schwarz Bayesian information criterion (SBC), Kolmogorov-Smirnov statistic (KS), Anderson-Darling statistic (AD), and Cramér–von Mises statistic (CvM).

You can request that the procedure output different types of diagnostic and inferential results, including the summary statistics of analysis variables, progress and status of the nonlinear estimation process, parameter estimates and their standard errors, estimated covariance structure of the parameters, and statistics of fit.

The following key features make PROC SEVSELECT unique among SAS procedures that can estimate continuous probability distributions:

- It enables you to fit a distribution model when the severity values are truncated, censored, or both. You can specify any combination of the following types of censoring and truncation effects: left-censoring,

right-censoring, left-truncation, or right-truncation. This is especially useful in applications with an insurance-type model, where a severity (loss) is reported and recorded only if it is greater than the deductible amount (left-truncation) and where a severity value greater than or equal to the policy limit is recorded at the limit (right-censoring). Another useful application is that of interval-censored data, where you know both the lower limit (right-censoring) and upper limit (left-censoring) on the severity, but you do not know the exact value.

PROC SEVSELECT also enables you to specify a *probability of observability* for the left-truncated data, which is a probability of observing values greater than the left-truncation threshold. This additional information can be useful in certain applications to more correctly model the distribution of the severity of events.

- It uses an appropriate estimator of the empirical distribution function (EDF). EDF is required to compute the KS, AD, and CvM statistics of fit. The procedure also provides the EDF estimates to your custom parameter initialization method. When you specify truncation or censoring, the EDF is estimated by using either Kaplan-Meier's product-limit estimator or Turnbull's estimator. The former is used by default when you specify only one form of censoring effect (right-censoring or left-censoring), and the latter is used by default when you specify both left-censoring and right-censoring effects.
- It enables you to define any arbitrary continuous parametric distribution model and to estimate its parameters. You just need to define the key components of the distribution, such as its probability density function (PDF) and cumulative distribution function (CDF), as a set of functions and subroutines written with the FCMP procedure, which is part of Base SAS software. As long as the functions and subroutines follow certain rules, the SEVSELECT procedure can fit the distribution model defined by them.
- It can model the influence of exogenous or regressor variables on a probability distribution, as long as the distribution has a scale parameter. A linear combination of regression effects is assumed to affect the scale parameter via an exponential link function. This type of model is referred to as the scale regression model.

If a distribution does not have a scale parameter, then either it needs to have another parameter that can be derived from a scale parameter by using a supported transformation or it needs to be reparameterized to have a scale parameter. If neither of these is possible, then regression effects cannot be modeled.

You can easily specify many types of regression effects by using various operators on a set of classification and continuous variables. You can specify classification variables in the CLASS statement. You can also construct the following special effects by using the EFFECT statement: collection effects, multimember effects, polynomial effects, and spline effects.

If you specify a large number of regression effects, then you can use the SELECTION statement to tell PROC SEVSELECT to perform scale regression model selection.

- It enables you to specify your own objective function to be optimized for estimating the parameters of a model. You can write SAS programming statements to specify the contribution of each observation to the objective function. You can use keyword functions such as `_PDF_` and `_CDF_` to generalize the objective function to any distribution. If you do not specify your own objective function, then PROC SEVSELECT estimates the parameters of a model by maximizing the likelihood function of the data.
- It enables you to create scoring functions that offer a convenient way to evaluate any distribution function, such as PDF, CDF, QUANTILE, or your custom distribution function, for a fitted model on new observations.

PROC SEVSELECT is the next-generation version of PROC HPSEVERITY. It can run on SAS Cloud Analytic Services (CAS). Because PROC SEVSELECT is a next-generation high-performance analytical procedure, it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section [“Introduction to Shared Concepts”](#) on page 54 in Chapter 4, [“Shared Concepts.”](#)

Getting Started: SEVSELECT Procedure

This section outlines the use of the SEVSELECT procedure to fit continuous probability distribution models. Three examples illustrate different features of the procedure.

A Simple Example of Fitting Predefined Distributions

The simplest way to use PROC SEVSELECT is to fit all the predefined distributions to a set of values and let the procedure identify the best-fitting distribution.

Consider a lognormal distribution, whose probability density function (PDF) f and cumulative distribution function (CDF) F are as follows, respectively, where Φ denotes the CDF of the standard normal distribution:

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2} \quad \text{and} \quad F(x; \mu, \sigma) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right)$$

The following DATA step statements simulate a sample from a lognormal distribution with population parameters $\mu = 1.5$ and $\sigma = 0.25$, and store the sample in the variable Y of a data set Work.Test_sev1:

```
/*----- Simple Lognormal Example -----*/
data test_sev1(keep=y label='Simple Lognormal Sample');
  call streaminit(45678);
  label y='Response Variable';
  Mu = 1.5;
  Sigma = 0.25;
  do n = 1 to 100;
    y = exp(Mu) * rand('LOGNORMAL')**Sigma;
    output;
  end;
run;
```

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “[Using CAS Sessions and CAS Engine Librefs](#)” on page 54 and “[Loading a SAS Data Set onto a CAS Server](#)” on page 55 in Chapter 4, “[Shared Concepts](#).”

You can use a DATA step as follows to load the data set Work.Test_sev1 into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.test_sev1;
  set test_sev1;
run;
```

These statements assume that your libref is named mylib, as in the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1724, but you can substitute any appropriately defined libref.

The following statements fit all the predefined distribution models to the values of Y and identify the best distribution according to the corrected Akaike’s information criterion (AICC):

```
proc sevselect data=mylib.test_sev1 crit=aicc
              plots(histogram kernel)=all; /*(cdf pdf);*/
  loss y;
  dist _predefined_;
run;
```

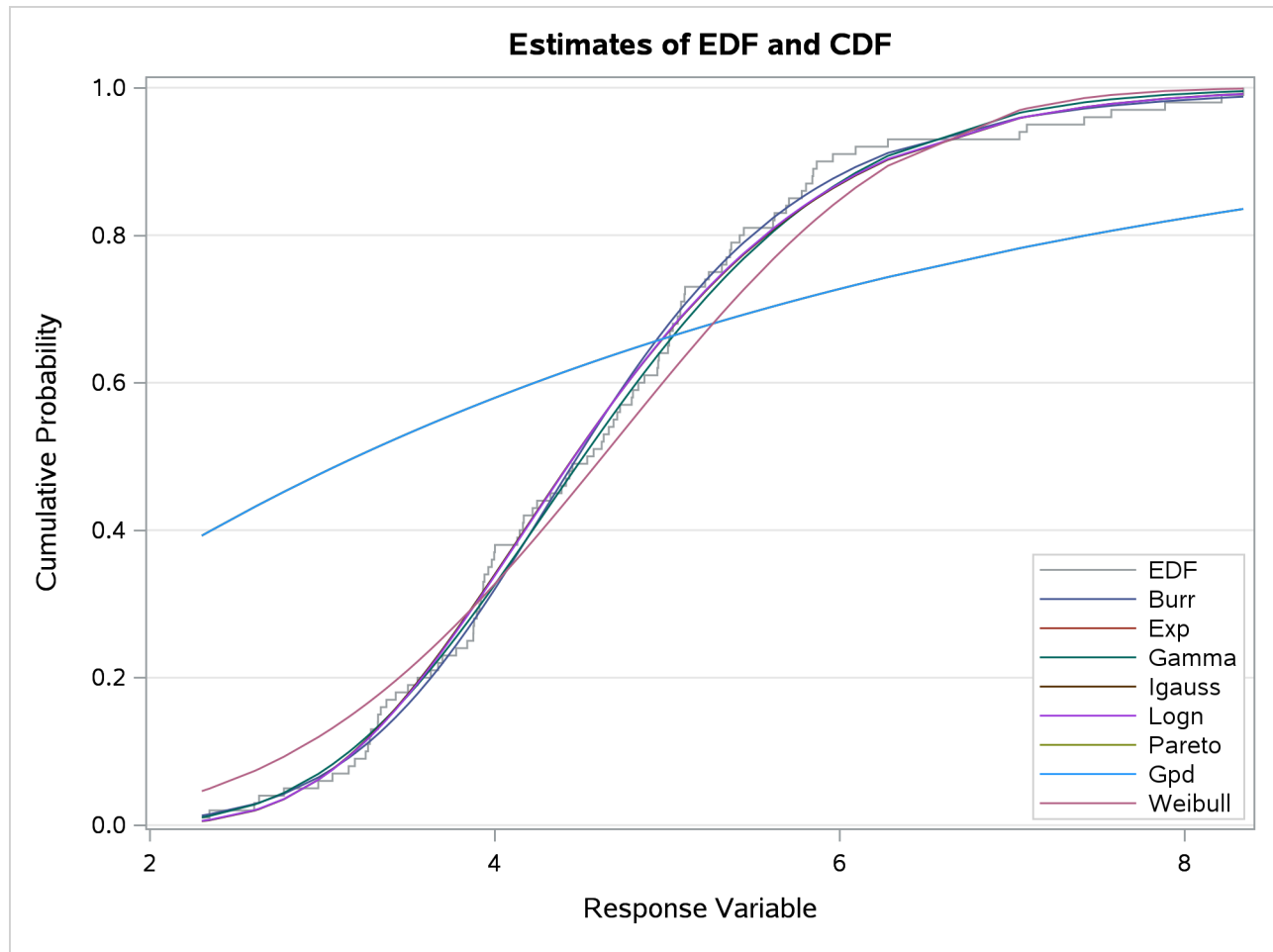
The PROC SEVSELECT statement specifies the input data table along with the model selection criterion. The PLOTS= option displays comparative plots of the probability density function (PDF) and the cumulative distribution function (CDF) for all candidate distributions. The HISTOGRAM and KERNEL options request that the PDF plot contain histogram and kernel density estimates, respectively, which are nonparametric estimates of the density function. The LOSS statement specifies the variable to be modeled, and the DIST statement with the _PREDEFINED_ keyword specifies that all the predefined distribution models be fitted.

Some of the default output that is displayed by this step is shown in Figure 26.1 through Figure 26.5. First the “Model Selection” table is displayed, as shown in Figure 26.1. The model selection table displays the convergence status, the value of the selection criterion, and the selection status for each of the candidate models. The Converged column indicates whether the estimation process for a particular distribution model has converged, might have converged, or failed. The Selected column indicates whether a particular distribution has the best fit for the data according to the selection criterion. For this example, the lognormal distribution model is selected, because it has the lowest value for the selection criterion.

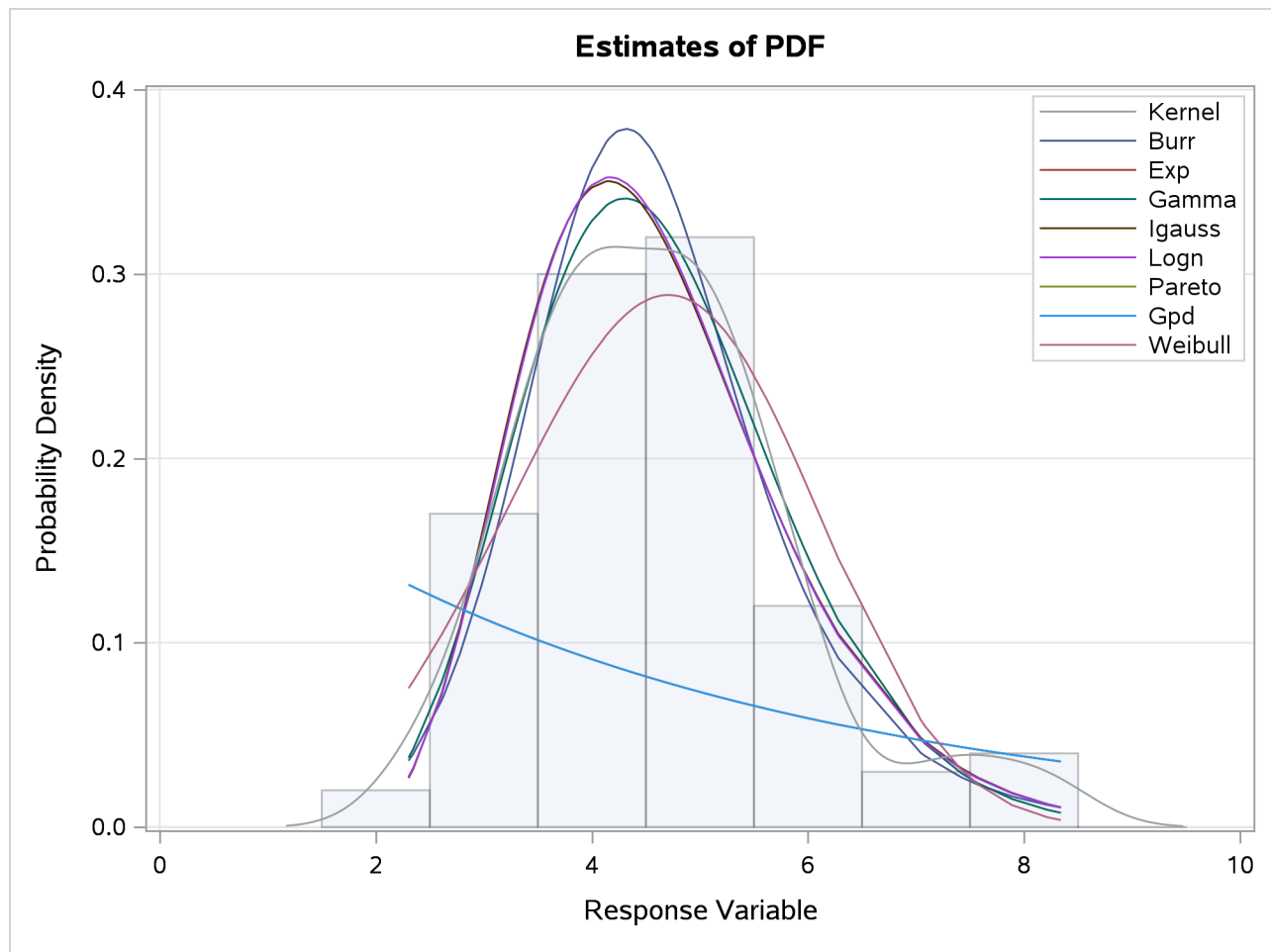
Figure 26.1 Model Selection Table

Model Selection			
Distribution	Converged	AICC	Selected
Burr	Yes	322.50845	No
Exp	Yes	508.12287	No
Gamma	Yes	320.50264	No
lgauss	Yes	319.61652	No
Logn	Yes	319.56579	Yes
Pareto	Yes	510.28172	No
Gpd	Yes	510.20576	No
Weibull	Yes	334.82373	No

Next, two comparative plots are prepared. These plots enable you to visually verify how the models differ from each other and from the nonparametric estimates. The plot in Figure 26.2 displays the CDF estimates of all the models and the estimates of the empirical distribution function (EDF). The CDF plot indicates that the Exp (exponential), Pareto, and Gpd (generalized Pareto) distributions are a poor fit as compared to the EDF estimate. The Weibull distribution is also a poor fit, although not as poor as exponential, Pareto, and generalized Pareto. The other four distributions seem to be quite close to each other and to the EDF estimate.

Figure 26.2 Comparison of EDF and CDF Estimates of the Fitted Models

The plot in [Figure 26.3](#) displays the PDF estimates of all the models and the nonparametric kernel and histogram estimates. The PDF plot enables better visual comparison between the Burr, gamma, Igauss (inverse Gaussian), and Logn (lognormal) models. The Burr and gamma models differ significantly from the Igauss and Logn models in the central portion of the range of Y values, whereas the latter two models fit the data almost identically. This provides a visual confirmation of the information in the “Model Selection” table in [Figure 26.1](#), which indicates that the AICC values of the Igauss and Logn models are very close.

Figure 26.3 Comparison of PDF Estimates of the Fitted Models

NOTE: When you request plots in a PROC SEVSELECT step that uses more than one worker node, the plots are approximate. PROC SEVSELECT creates these plots by using the sample that it uses to compute EDF-based fit statistics. It first computes the empirical quantiles on each worker node, then merges them to create the final plot. For more accurate visualization, it is recommended that you request plots only when you run PROC SEVSELECT in a CAS session that uses no more than one worker node.

The comparative plots are followed by the estimation information for each of the candidate models. The information for the lognormal model, which is the best-fitting model, is shown in Figure 26.4. The first table displays a summary of the distribution. The second table displays the convergence status. This is followed by a summary of the optimization process that indicates the technique used, the number of iterations, the number of times the objective function was evaluated, and the log likelihood that is obtained at the end of the optimization. Because the model with lognormal distribution has converged, PROC SEVSELECT displays its statistics of fit and parameter estimates. The estimates of $\mu=1.49605$ and $\sigma=0.26243$ are quite close to the population parameters of $\mu=1.5$ and $\sigma=0.25$ from which the sample was generated. The p -value for each estimate indicates the rejection of the null hypothesis that the estimate is 0, implying that both the estimates are significantly different from 0.

Figure 26.4 Estimation Details for the Lognormal Model**The SEVSELECT Procedure****Logn Distribution**

Model Information					
Distribution					Logn
Description					Lognormal Distribution
Distribution Parameters					2
Convergence Status					
Convergence criterion (GCONV=1E-8) satisfied.					
Optimization Summary					
Optimization Technique					Trust Region
Iterations					2
Function Calls					8
Log Likelihood					-157.72104
Fit Statistics					
-2 Log Likelihood					315.44208
Akaike's Information Criterion					319.44208
Corrected Akaike's Information Criterion					319.56579
Schwarz's Bayesian Information Criterion					324.65242
Kolmogorov-Smirnov Statistic					0.50641
Anderson-Darling Statistic					0.31240
Cramer-von Mises Statistic					0.04353
Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	1.49605	0.02651	56.43	<.0001
Sigma	1	0.26243	0.01874	14.00	<.0001

The parameter estimates of the Burr distribution are shown in [Figure 26.5](#). These estimates are used in the next example.

Figure 26.5 Parameter Estimates for the Burr Model

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	4.62348	0.46181	10.01	<.0001
Alpha	1	1.15706	0.47493	2.44	0.0167
Gamma	1	6.41227	0.99039	6.47	<.0001

An Example with Left-Truncation and Right-Censoring

PROC SEVSELECT enables you to specify that the response variable values are left-truncated or right-censored. The following DATA step expands the data table of the previous example to simulate a scenario that is typically encountered by an automobile insurance company. The values of the variable Y represent the loss values on claims that are reported to an auto insurance company. The variable THRESHOLD records the deductible on the insurance policy. If the actual value of Y is less than or equal to the deductible, then it is unobservable and does not get recorded. In other words, THRESHOLD specifies the left-truncation of Y. LIMIT records the policy limit. If the value of Y is equal to or greater than the recorded value, then the observation is right-censored.

```
/*----- Lognormal Model with left-truncation and censoring -----*/
data test_sev2(keep=y threshold limit
               label='A Lognormal Sample With Censoring and Truncation');
  set test_sev1;
  label y='Censored & Truncated Response';
  if _n_ = 1 then call streaminit(45679);

  /* make about 20% of the observations left-truncated */
  if (rand('UNIFORM') < 0.2) then
    threshold = y * (1 - rand('UNIFORM'));
  else
    threshold = .;
  /* make about 15% of the observations right-censored */
  iscens = (rand('UNIFORM') < 0.15);
  if (iscens) then
    limit = y;
  else
    limit = .;
run;
```

You can use a DATA step as follows to load the data set Work.Test_sev2 into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.test_sev2;
  set test_sev2;
run;
```

The following statements fit four predefined distributions (lognormal, Burr, gamma, and Weibull) and use the AICC to identify the distribution that best fits the data:

```
proc sevselect data=mylib.test_sev2 crit=aicc
               print=all plots=all;
  loss y / lt=threshold rc=limit;

  dist logn burr gamma weibull;
run;
```

The LOSS statement specifies the left-truncation and right-censoring variables. The DIST statement specifies the candidate distributions. The PRINT= option in the PROC SEVSELECT statement requests that all the

displayed output be prepared. The PLOTS= option in the PROC SEVSELECT statement requests that all available plots be prepared.

Some of the key results that PROC SEVSELECT prepares are shown in Figure 26.6 through Figure 26.13. In addition to the estimates of the range, mean, and standard deviation of Y, the “Descriptive Statistics for y” table shown in Figure 26.6 also indicates the number of observations that are left-truncated or right-censored. The “Model Selection” table in Figure 26.6 shows that models with all the candidate distributions have converged and that the Logn (lognormal) model has the best fit for the data according to the AICC criterion.

Figure 26.6 Summary Results for the Truncated and Censored Data

Descriptive Statistics for y			
Observations			100
Observations Used for Estimation			100
Minimum			2.30264
Maximum			8.34116
Mean			4.62007
Standard Deviation			1.23627
Left Truncated Observations			23
Right Censored Observations			14

Model Selection			
Distribution	Converged	AICC	Selected
Logn	Yes	298.92672	Yes
Burr	Yes	302.66229	No
Gamma	Yes	299.45293	No
Weibull	Yes	309.26779	No

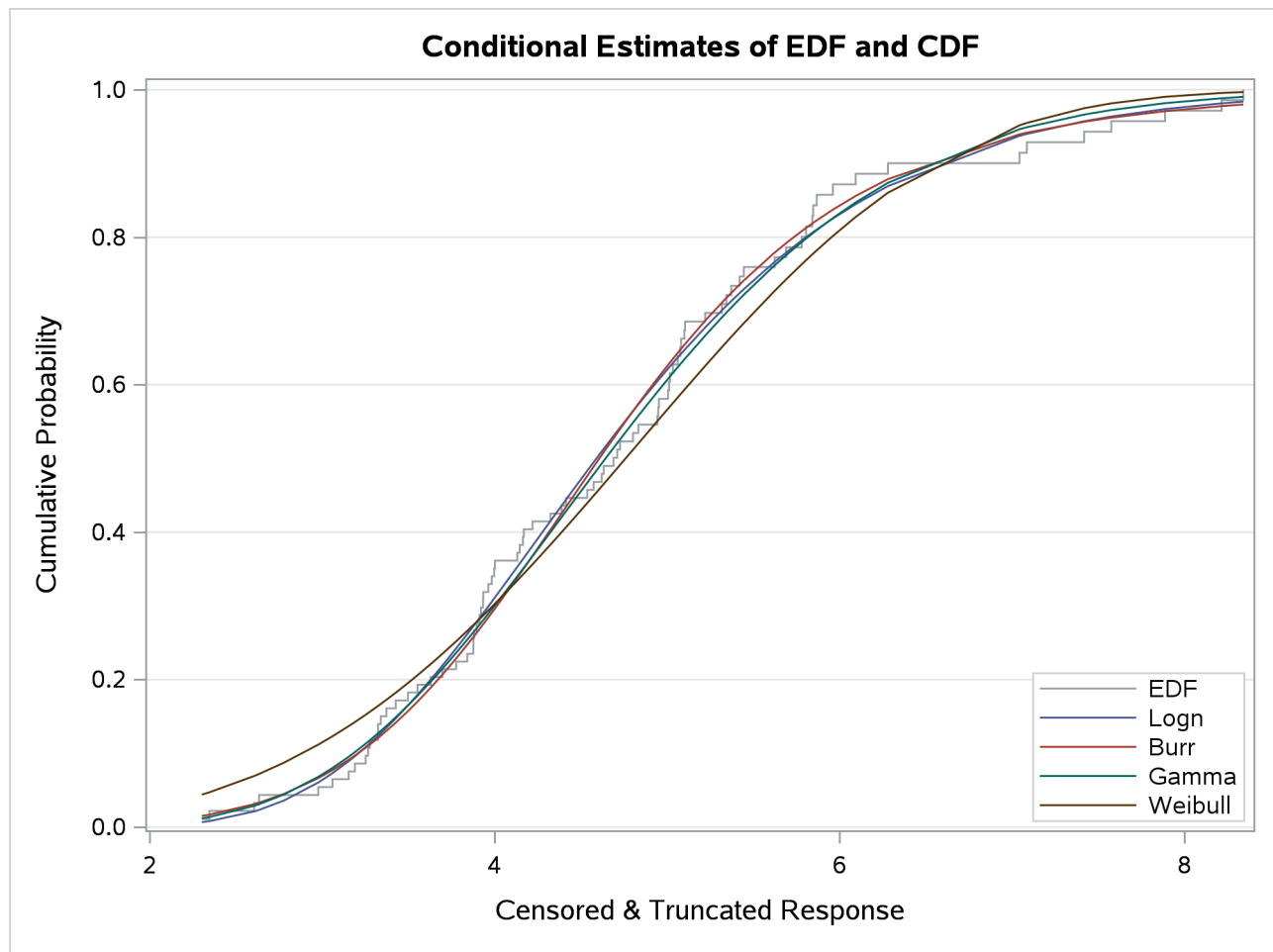
PROC SEVSELECT also prepares a table that shows all the fit statistics for all the candidate models. It is useful to see which model would be the best fit according to each of the criteria. The “All Fit Statistics” table prepared for this example is shown in Figure 26.7. It indicates that the lognormal model is chosen by all the criteria.

Figure 26.7 Comparing All Statistics of Fit for the Truncated and Censored Data

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	294.80301*	298.80301*	298.92672*	304.01335*	0.51824*	0.34736*	0.05159*
Burr	296.41229	302.41229	302.66229	310.22780	0.66984	0.36712	0.05726
Gamma	295.32921	299.32921	299.45293	304.53955	0.62511	0.42921	0.05526
Weibull	305.14408	309.14408	309.26779	314.35442	0.93307	1.40699	0.17465

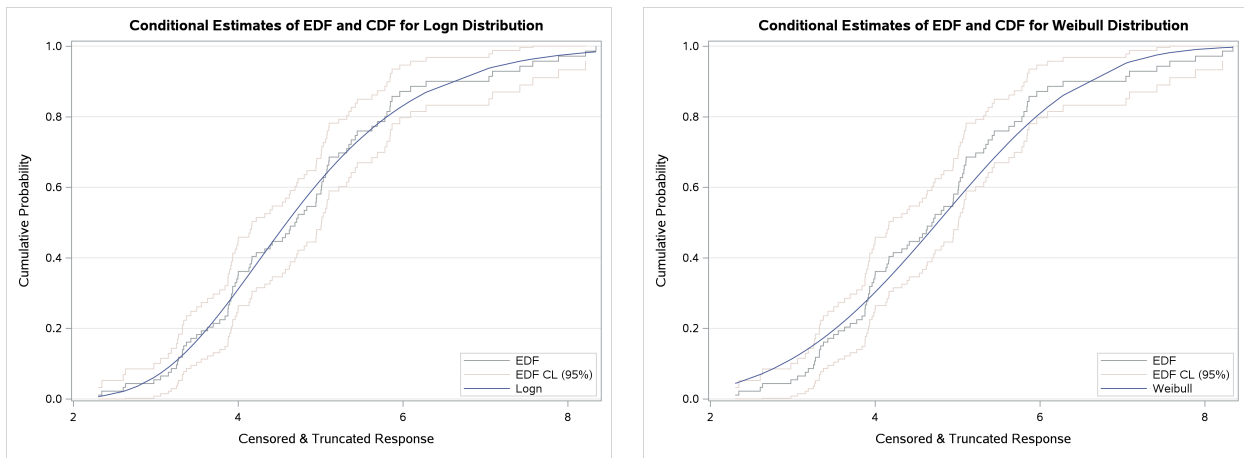
Asterisk (*) denotes the best model in the column.

The plot that compares EDF and CDF estimates is shown in Figure 26.8. When you specify left-truncation, both the EDF and CDF estimates are conditional on the response variable being greater than the smallest left-truncation threshold in the sample.

Figure 26.8 EDF and CDF Estimates for the Truncated and Censored Data

When you specify the `PLOTS=CDFPERDIST` option, PROC SEVSELECT prepares a plot that compares the nonparametric EDF estimates with the parametric CDF estimates for each distribution. These plots for lognormal and Weibull distributions are shown in Figure 26.9. These plots also contain the lower and upper confidence limits of EDF for the specified confidence level. Because no confidence level is specified in the `EDFALPHA=` option in the PROC SEVSELECT statement, a default confidence level of 95% is used; this is equivalent to specifying `EDFALPHA=0.05`. If the CDF estimates lie entirely within the EDF confidence interval, then you can be 95% confident that the parametric and nonparametric estimates are in agreement.

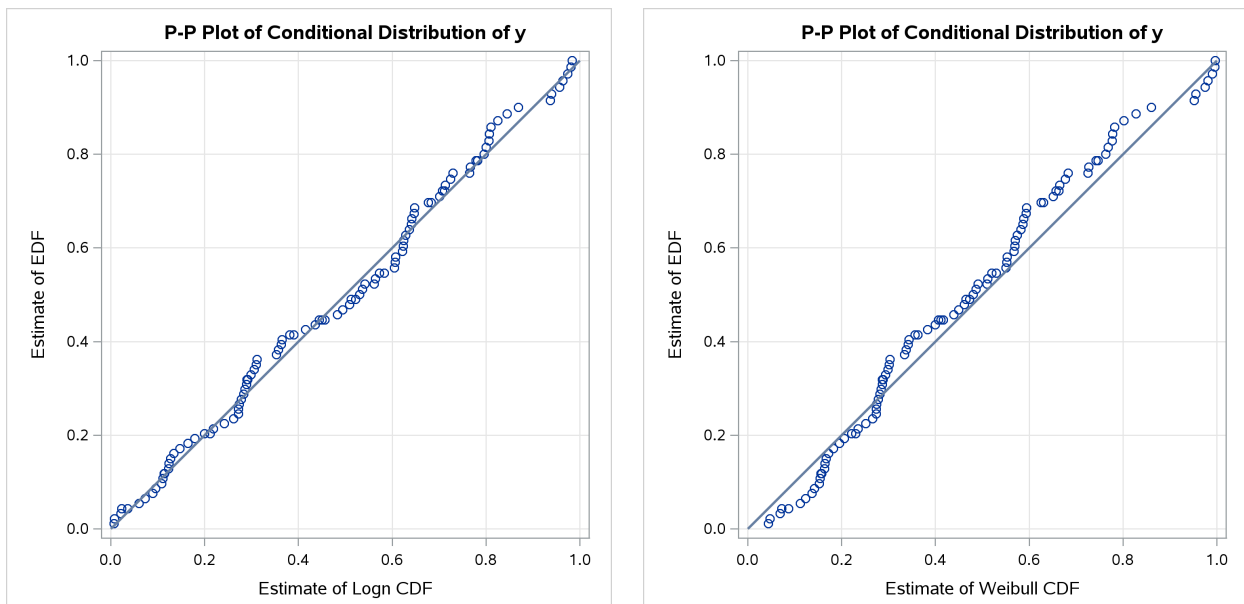
Figure 26.9 Comparing EDF and CDF Estimates for Lognormal and Weibull Models Fitted to Truncated and Censored Data



There are two additional ways to compare nonparametric (empirical) and parametric estimates for each model that has not failed to converge:

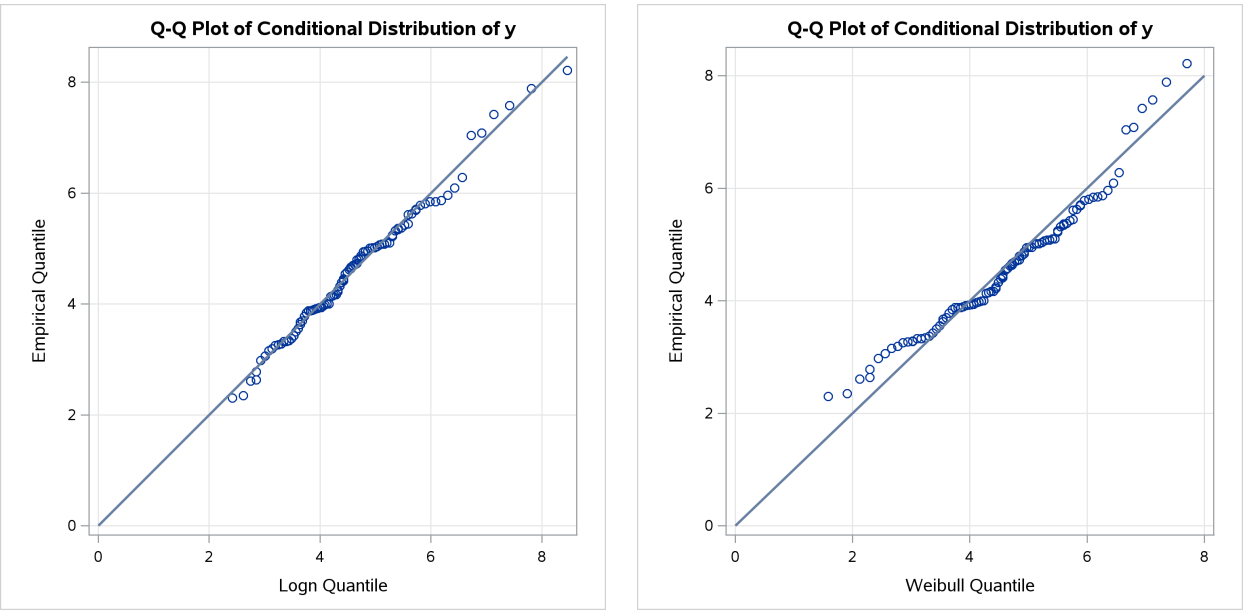
- A P-P plot is a scatter plot of the EDF and CDF estimates. The model for which the points are scattered closer to the unit-slope reference line is a better fit. The P-P plot for the lognormal distribution is shown in Figure 26.10. It indicates that the EDF matches the CDF very closely. In contrast, the P-P plot for the Weibull distribution, also shown in Figure 26.10, indicates a poor fit.

Figure 26.10 P-P Plots for Lognormal and Weibull Models Fitted to Truncated and Censored Data



- A Q-Q plot is a scatter plot of empirical quantiles and the quantiles of a parametric distribution. As in the P-P plot, points that are scattered closer to the unit-slope reference line indicate a better fit. The Q-Q plots of lognormal and Weibull distributions are shown in Figure 26.11; these plots confirm the conclusions that are arrived at by comparing the P-P plots.

Figure 26.11 Q-Q Plots for Lognormal and Weibull Models Fitted to Truncated and Censored Data



Specifying Initial Values for Parameters

All the predefined distributions have parameter initialization functions built into them. For the current example, Figure 26.12 shows the initial values that are obtained by the predefined method for the Burr distribution. It also shows the summary of the optimization process and the final parameter estimates.

Figure 26.12 Burr Model Summary for the Truncated and Censored Data

Initial Parameter Values and Bounds					
Parameter	Initial Value	Lower Bound	Upper Bound		
Theta	4.78102	1.05367E-8	Infy		
Alpha	2.00000	1.05367E-8	Infy		
Gamma	2.00000	1.05367E-8	Infy		
Optimization Summary					
Optimization Technique		Trust Region			
Iterations		8			
Function Calls		23			
Log Likelihood		-148.2061441			
Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	4.76980	0.62492	7.63	<.0001
Alpha	1	1.16363	0.58859	1.98	0.0509
Gamma	1	5.94081	1.05004	5.66	<.0001

You can specify a different set of initial values if estimates are available from fitting the distribution to similar

data. For this example, the parameters of the Burr distribution can be initialized with the final parameter estimates of the Burr distribution that were obtained in the first example (shown in Figure 26.5). One of the ways in which you can specify the initial values is as follows:

```
/*----- Specifying initial values using INIT= option -----*/
proc sevselect data=mylib.test_sev2 crit=aicc print=all;
  loss y / lt=threshold rc=limit;

  dist burr(init=(theta=4.62348 alpha=1.15706 gamma=6.41227));
run;
```

The names of the parameters that are specified in the INIT option must match the parameter names in the definition of the distribution. The results obtained with these initial values are shown in Figure 26.13. These results indicate that new set of initial values causes the optimizer to reach the same solution with fewer iterations and function evaluations as compared to the default initialization.

Figure 26.13 Burr Model Optimization Summary for the Truncated and Censored Data

The SEVSELECT Procedure

Burr Distribution

Optimization Summary					
Optimization Technique		Trust Region			
Iterations		5			
Function Calls		16			
Log Likelihood		-148.2061441			

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	4.76980	0.62492	7.63	<.0001
Alpha	1	1.16363	0.58859	1.98	0.0509
Gamma	1	5.94081	1.05004	5.66	<.0001

An Example of Modeling Regression Effects

Consider a scenario in which the magnitude of the response variable might be affected by some regressor (exogenous or independent) variables. The SEVSELECT procedure enables you to model the effect of such variables on the distribution of the response variable via an exponential link function. In particular, if you have k random regressor variables denoted by x_j ($j = 1, \dots, k$), then the distribution of the response variable Y is assumed to have the form

$$Y \sim \exp\left(\sum_{j=1}^k \beta_j x_j\right) \cdot \mathcal{F}(\Theta)$$

where $\mathcal{F}$ denotes the distribution of Y with parameters Θ and β_j ($j = 1, \dots, k$) denote the regression parameters (coefficients).

For the effective distribution of Y to be a valid distribution from the same parametric family as $\mathcal{F}$, it is necessary for $\mathcal{F}$ to have a scale parameter. The effective distribution of Y can be written as

$$Y \sim \mathcal{F}(\theta, \Omega)$$

where θ denotes the scale parameter and Ω denotes the set of nonscale parameters. The scale θ is affected by the regressors as

$$\theta = \theta_0 \cdot \exp\left(\sum_{j=1}^k \beta_j x_j\right)$$

where θ_0 denotes a *base* value of the scale parameter.

Given this form of the model, PROC SEVSELECT allows a distribution to be a candidate for modeling regression effects only if it has an untransformed or a log-transformed scale parameter.

All the predefined distributions, except the lognormal distribution, have a direct scale parameter (that is, a parameter that is a scale parameter without any transformation). For the lognormal distribution, the parameter μ is a log-transformed scale parameter. This can be verified by replacing μ with a parameter $\theta = e^\mu$, which results in the following expressions for the PDF f and the CDF F in terms of θ and σ , respectively, where Φ denotes the CDF of the standard normal distribution:

$$f(x; \theta, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x) - \log(\theta)}{\sigma}\right)^2} \quad \text{and} \quad F(x; \theta, \sigma) = \Phi\left(\frac{\log(x) - \log(\theta)}{\sigma}\right)$$

With this parameterization, the PDF satisfies the $f(x; \theta, \sigma) = \frac{1}{\theta} f\left(\frac{x}{\theta}; 1, \sigma\right)$ condition and the CDF satisfies the $F(x; \theta, \sigma) = F\left(\frac{x}{\theta}; 1, \sigma\right)$ condition. This makes θ a scale parameter. Hence, $\mu = \log(\theta)$ is a log-transformed scale parameter and the lognormal distribution is eligible for modeling regression effects.

The following DATA step simulates a lognormal sample whose scale is decided by the values of the three regressors X1, X2, and X3 as follows:

$$\mu = \log(\theta) = 1 + 0.75 X1 - X2 + 0.25 X3$$

```
/*----- Lognormal Model with Regressors -----*/
data test_sev3(keep=y x1-x3
                label='A Lognormal Sample Affected by Regressors');
    array x{*} x1-x3;
    array b{4} _TEMPORARY_ (1 0.75 -1 0.25);
    call streaminit(45678);
    label y='Response Influenced by Regressors';
    Sigma = 0.25;
    do n = 1 to 100;
        Mu = b(1); /* log of base value of scale */
        do i = 1 to dim(x);
            x(i) = rand('UNIFORM');
            Mu = Mu + b(i+1) * x(i);
        end;
        y = exp(Mu) * rand('LOGNORMAL')**Sigma;
        output;
    end;
run;
```

You can use a DATA step as follows to load the data set `Work.Test_sev3` into a data table in your session that is associated with the `mylib` libref. The DATA step assumes that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
data mylib.test_sev3;
    set test_sev3;
run;
```

The following PROC SEVSELECT step fits the lognormal, Burr, and gamma distribution models to these data. The regressors are specified in the SCALEMODEL statement.

```
proc sevselect data=mylib.test_sev3 crit=aicc print=all;
    loss y;
    scalemodel x1-x3;

    dist logn burr gamma;
run;
```

Some of the key results that PROC SEVSELECT prepares are shown in [Figure 26.14](#) through [Figure 26.18](#). The descriptive statistics of all the variables are shown in [Figure 26.14](#).

Figure 26.14 Summary Results for the Regression Example

The SEVSELECT Procedure

Descriptive Statistics for y					
Observations					100
Observations Used for Estimation					100
Minimum					1.17863
Maximum					6.65269
Mean					2.99859
Standard Deviation					1.12845

Descriptive Statistics for Regressors					
Variable	N	Minimum	Maximum	Mean	Standard Deviation
x1	100	0.0005115	0.97971	0.51689	0.28206
x2	100	0.01883	0.99937	0.47345	0.28885
x3	100	0.00255	0.97558	0.48301	0.29709

The comparison of the fit statistics of all the models is shown in [Figure 26.15](#). It indicates that the lognormal model is the best model according to each of the likelihood-based statistics, whereas the gamma model is the best model according to two of the three EDF-based statistics.

Figure 26.15 Comparison of Statistics of Fit for the Regression Example

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	187.49609*	197.49609*	198.13439*	210.52194*	1.97544	17.24618	1.21665
Burr	190.69154	202.69154	203.59476	218.32256	2.09334	13.93436*	1.28529
Gamma	188.91483	198.91483	199.55313	211.94069	1.94472*	15.84787	1.17617*

Asterisk (*) denotes the best model in the column.

The model information and the convergence results of the lognormal model are shown in Figure 26.16. The iteration history gives you a summary of how the optimizer is traversing the surface of the log-likelihood function in its attempt to reach the optimum. Both the change in the log likelihood and the maximum gradient of the objective function with respect to any of the parameters typically approach 0 if the optimizer converges.

Figure 26.16 Convergence Results for the Lognormal Model with Regressors

The SEVSELECT Procedure				
Logn Distribution				
Model Information				
Distribution	Logn			
Description	Lognormal Distribution			
Distribution Parameters	2			
Regression Parameters	3			
Convergence Status				
Convergence criterion (GCONV=1E-8) satisfied.				
Optimization Iteration History				
	Function Calls	Log Likelihood	Change	Maximum Gradient
0	2	93.75285		6.16002
1	4	93.74805	-0.00481	0.11031
2	6	93.74805	-1.5017E-6	0.0000338
3	10	93.74805	-1.705E-13	3.1353E-12
Optimization Summary				
Optimization Technique		Trust Region		
Iterations		3		
Function Calls		10		
Log Likelihood		-93.74804701		

The final parameter estimates of the lognormal model are shown in Figure 26.17. All the estimates are significantly different from 0. The estimate that is reported for the parameter MU is the base value for the log-transformed scale parameter μ . Let x_i ($1 \leq i \leq 3$) denote the observed value for regressor X_i . If the lognormal distribution is chosen to model Y , then the effective value of the parameter μ varies with the observed values of regressors as

$$\mu = 1.04047 + 0.65221 x_1 - 0.91116 x_2 + 0.16243 x_3$$

These estimated coefficients are reasonably close to the population parameters (that is, within one or two standard errors).

Figure 26.17 Parameter Estimates for the Lognormal Model with Regressors

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	1.04047	0.07614	13.66	<.0001
Sigma	1	0.22177	0.01609	13.78	<.0001
x1	1	0.65221	0.08167	7.99	<.0001
x2	1	-0.91116	0.07946	-11.47	<.0001
x3	1	0.16243	0.07782	2.09	0.0395

The estimates of the gamma distribution model, which is the best model according to a majority of the EDF-based statistics, are shown in [Figure 26.18](#). The estimate that is reported for the parameter *Theta* is the base value for the scale parameter θ . If the gamma distribution is chosen to model Y , then the effective value of the scale parameter is $\theta = 0.14293 \exp(0.64562 x_1 - 0.89831 x_2 + 0.14901 x_3)$.

Figure 26.18 Parameter Estimates for the Gamma Model with Regressors

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	0.14293	0.02329	6.14	<.0001
Alpha	1	20.37726	2.93277	6.95	<.0001
x1	1	0.64562	0.08224	7.85	<.0001
x2	1	-0.89831	0.07962	-11.28	<.0001
x3	1	0.14901	0.07870	1.89	0.0613

Syntax: SEVSELECT Procedure

The following statements are available in the SEVSELECT procedure:

```

PROC SEVSELECT options ;
  BY variable-list ;
  LOSS < response-variable > < / censoring-truncation-options > ;
  WEIGHT weight-variable ;
  DIST distribution-name-or-keyword < (distribution-option) < distribution-name-or-keyword
    < (distribution-option) > > ... > < / options > ;
  CLASS variable < (options) > ... < variable < (options) > > < / global-options > ;
  EFFECT name=effect-type(variables < / options >) ;
  SCALEMODEL regression-effect-list < / scalemodel-options > ;
  SELECTION < METHOD=method < (method-options) > >< options > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  RESTRICT restriction1 [, restriction2 ...] ;
  OUTPUT < OUT=libref.data-table > output-options ;
  OUTSCORELIB < OUTLIB=> fcmp-library-name options ;
  NLOPTIONS options ;
  Programming statements ;

```

Functional Summary

Table 26.1 summarizes the statements and options that control the SEVSELECT procedure.

Table 26.1 Functional Summary

Description	Statement	Option
Statements		
Specifies BY-group processing	BY	
Specifies the response variable to model along with censoring and truncation effects	LOSS	
Specifies the weight variable	WEIGHT	
Specifies distributions to fit	DIST	
Specifies the classification variables	CLASS	
Specifies the constructed regression effects	EFFECT	
Specifies the regression effects to model	SCALEMODEL	
Specifies the options for selecting the best subset of regression effects	SELECTION	
Specifies the ODS tables to display (an alternative to the PRINT= option)	DISPLAY	
Specifies the ODS tables to save as output data tables	DISPLAYOUT	
Specifies linear restrictions on parameters	RESTRICT	
Specifies the scoring functions and quantiles to write	OUTPUT	
Specifies the library to write scoring functions to	OUTSCORELIB	
Specifies the optimization options	NLOPTIONS	
Specifies programming statements that define an objective function	Programming statements	

Table 26.1 *continued*

Description	Statement	Option
Input and Output Options		
Specifies the input data table	PROC SEVSELECT	DATA=
Specifies the input data table for parameter estimates	PROC SEVSELECT	INEST=
Specifies the output data table for estimates of scoring functions and quantiles	OUTPUT	OUT=
Specifies the output data table for parameter estimates	PROC SEVSELECT	OUTEST=
Specifies the output data table for model information	PROC SEVSELECT	OUTMODELINFO=
Specifies the output data table for statistics of fit	PROC SEVSELECT	OUTSTAT=
Specifies the output item store for context and estimation results	PROC SEVSELECT	OUTSTORE=
Data Interpretation Options		
Specifies left-censoring	LOSS	LEFTCENSORED=
Specifies left-truncation	LOSS	LEFTTRUNCATED=
Specifies the probability of observability	LOSS	PROBOBSERVED=
Specifies right-censoring	LOSS	RIGHTCENSORED=
Specifies right-truncation	LOSS	RIGHTTRUNCATED=
Distribution Processing Options		
Specifies the input data table to read the distribution definitions from	DIST	INFUNCDEF=
Specifies that distributions be listed in the log without estimating any models that use them	DIST	LISTONLY
Specifies the input data table to write the distribution definitions to	DIST	OUTFUNCDEF=
Specifies that distributions be validated without estimating any models that use them	DIST	VALIDATEONLY
Model Estimation Options		
Specifies the method for computing mixture distribution	SCALEMODEL	DFMIXTURE=
Specifies that informative missingness be used to model missing values of regressors	SCALEMODEL	INFORMATIVE
Specifies initial values for model parameters	DIST	INIT=
Specifies the sample to be used for computing initial parameter estimates	PROC SEVSELECT	INITSAMPLE
Specifies that observations do not need to be balanced among threads of computation	PROC SEVSELECT	NOBALANCETHREADS
Specifies role of constant distribution parameters in fit statistic calculations	PROC SEVSELECT	NOCONSTFITSTATS

Table 26.1 *continued*

Description	Statement	Option
Specifies the objective function symbol	PROC SEVSELECT	OBJECTIVE=
Specifies the offset variable in the scale regression model	SCALEMODEL	OFFSET=
Specifies the denominator for computing covariance estimates	PROC SEVSELECT	VARDEF=
Regression Effect Selection Options		
Specifies whether and how the model hierarchy requirement is applied	SELECTION	HIERARCHY=
Specifies the set of regression effects to be forced in all models	SCALEMODEL	INCLUDE=
Specifies the selection method	SELECTION	METHOD=
Specifies the selection order for displaying effects in the selected model	SELECTION	ORDERSELECT
Specifies the optimization technique to use for the intermediate selection steps	PROC SEVSELECT	SELECTNLOTECH=
Specifies the set of regression effects to begin the selection process	SCALEMODEL	START=
Specifies how to apply the STOP= criterion	SELECTION	STOPHORIZON=
Regression Effect Selection Method Options		
Specifies a criterion for choosing the best model at each step	SELECTION	CHOOSE=
Specifies that the competitive form of the stepwise selection method be used	SELECTION	COMPETITIVE
Specifies the maximum number of effects in the model	SELECTION	MAXEFFECTS=
Specifies the maximum number of selection steps	SELECTION	MAXSTEPS=
Specifies the minimum number of effects in the model	SELECTION	MINEFFECTS=
Specifies a criterion to decide which effects enter or leave the model at each step	SELECTION	SELECT=
Specifies a criterion for stopping the selection process	SELECTION	STOP=
Empirical Distribution Function (EDF) Estimation Options		
Specifies the nonparametric method of CDF estimation	PROC SEVSELECT	EMPIRICALCDF=
EMPIRICALCDF=MODIFIEDKLM Options		
Specifies the α value for the lower bound on risk set size	PROC SEVSELECT	ALPHA=
Specifies the c value for the lower bound on risk set size	PROC SEVSELECT	C=

Table 26.1 *continued*

Description	Statement	Option
Specifies the absolute lower bound on risk set size	PROC SEVSELECT	RSLB=
EMPIRICALCDF=TURNBULL Options		
Specifies that the final EDF estimates be maximum likelihood estimates	PROC SEVSELECT	ENSUREMLE
Specifies the relative convergence criterion	PROC SEVSELECT	EPS=
Specifies the maximum number of iterations	PROC SEVSELECT	MAXITER=
Specifies the threshold below which an EDF estimate is deemed to be 0	PROC SEVSELECT	ZEROPROB=
OUT= Data Table Generation Options		
Specifies the variables to copy from the DATA= data table to the OUT= data table	OUTPUT	COPYVARS=
Specifies the scoring functions to estimate	OUTPUT	FUNCTIONS=
Specifies the quantiles to estimate	OUTPUT	QUANTILES=
OUTEST= Data Table Generation Options		
Specifies that the OUTEST= data table contain covariance estimates	PROC SEVSELECT	COVOUT
Specifies that only the selected regression parameters be written	PROC SEVSELECT	SELECTOUT
Specifies that estimates of parameters not in the final model be 0	PROC SEVSELECT	ZEROEST
Scoring Function Generation Options		
Writes scoring functions of all models to one package	OUTSCORELIB	COMMONPACKAGE
Specifies the output data table for BY-group identifiers	OUTSCORELIB	OUTBYID=
Specifies the output library for scoring functions	OUTSCORELIB	OUTLIB=
Displayed Output Options		
Specifies the criterion to report in the model selection table	PROC SEVSELECT	CRITERION=
Specifies the level of detail to be produced about the selection process	SELECTION	DETAILS=
Specifies the confidence level for reporting the confidence interval for EDF estimates	PROC SEVSELECT	EDFALPHA=
Limits or suppresses the display of class levels	PROC SEVSELECT	NOCLPRINT
Suppresses all displayed and graphical output	PROC SEVSELECT	NOPRINT
Specifies which graphical output to prepare	PROC SEVSELECT	PLOTS=
Specifies which graphical output to prepare about the selection process	SELECTION	PLOTS=
Specifies which output to display	PROC SEVSELECT	PRINT=

Table 26.1 *continued*

Description	Statement	Option
Prints internal names assigned to parameters	PROC SEVSELECT	PRINTINTERNALNAMES

PROC SEVSELECT Statement

PROC SEVSELECT *options* ;

The PROC SEVSELECT statement invokes the procedure. You can specify two types of *options* in the PROC SEVSELECT statement. One set of *options* controls input and output. The other set of *options* controls the model estimation and selection process.

The following *options* control the input data tables used by PROC SEVSELECT and various forms of output generated by PROC SEVSELECT. The *options* are listed in alphabetical order.

COVOUT

specifies that the OUTEST= data table contain the estimate of the covariance structure of the parameters. This option has no effect if you do not specify the OUTEST= option. For more information about how the covariance is reported in the OUTEST= data table, see the section “OUTEST= Data Table” on page 1840.

CRITERION=*criterion-option*

CRITERIA=*criterion-option*

CRIT=*criterion-option*

specifies the criterion to report in the model selection table.

If you specify two or more candidate models for estimation, then the one that has the best value for the specified criterion is marked as the best model in the model selection table that is displayed by default or when you specify the PRINT=SELECTION option. If you specify the OUTSTAT= data table, then the observation for the model that has the best value for the specified criterion has a value of 1 for the variable `_SELECTED_`.

You can specify one of the following *criterion-options*:

AD	specifies the Anderson-Darling (AD) statistic value, which is computed by using the empirical distribution function (EDF) estimate, as the selection criterion. A lower value is deemed better.
AIC	specifies Akaike's information criterion (AIC) as the selection criterion. A lower value is deemed better.
AICC	specifies the finite-sample corrected Akaike's information criterion (AICC) as the selection criterion. A lower value is deemed better.
CUSTOM	specifies the custom objective function as the selection criterion. You can specify this criterion only if you also specify the OBJECTIVE= option. A lower value is deemed better.

CVM	specifies the Cramér–von Mises (CvM) statistic value, which is computed by using the empirical distribution function (EDF) estimate, as the selection criterion. A lower value is deemed better.
KS	specifies the Kolmogorov-Smirnov (KS) statistic value, which is computed by using the empirical distribution function (EDF) estimate, as the selection criterion. A lower value is deemed better.
LOGLIKELIHOOD LL	specifies $-2 * \log(L)$ as the selection criterion, where L is the likelihood of the data. A lower value is deemed better.
SBC BIC	specifies the Schwarz Bayesian information criterion (SBC) as the selection criterion. A lower value is deemed better.

If you do not specify the **CRITERION=** option, then by default, the procedure uses **CRITERION=CUSTOM** if you specify the **OBJECTIVE=** option, or **CRITERION=LOGLIKELIHOOD** if you do not specify the **OBJECTIVE=** option.

For more information about these *criterion-options*, see the section “Statistics of Fit” on page 1806.

DATA=*libref.data-table*

names the input data table for PROC SEVSELECT to use. *libref.data-table* is a two-level name, where

<i>libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>libref</i> , see the section “Using CAS Sessions and CAS Engine Librefs” on page 1724.
<i>data-table</i>	specifies the name of the input data table.

EDFALPHA=*confidence-level*

specifies the confidence level in the range (0,1) to use for computing the confidence intervals for the EDF estimates. The lower and upper confidence limits that correspond to this level are displayed in the plot that is created when you specify the **PLOTS=CDFPERDIST** option.

If you do not specify the **EDFALPHA=** option, then PROC SEVSELECT uses a default value of 0.05.

INEST=*libref.data-table*

names the input data table that contains the initial values of the parameter estimates to start the optimization process.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1724. The *libref* must be identical to the *libref* that you specify in the **DATA=** option.

The initial values that you specify in the **INIT=** option in the **DIST** statement take precedence over any initial values that you specify in the **INEST=** data table. For more information about the variables in this data table, see the section “**INEST= Data Table**” on page 1839.

NOCLPRINT*<=number>*

suppresses the display of the “Class Level Information” table if you do not specify *number*. If you specify *number*, the values of the classification variables are displayed for only those variables whose number of levels is less than *number*. Specifying a *number* helps reduce the size of the “Class Level Information” table if some classification variables have a large number of levels. This option has no effect if you do not specify the CLASS statement.

NOPRINT

turns off all displayed and graphical output. If you specify this option, then any value that you specify for the PRINT= option is ignored.

OUTEST*=libref.data-table*

names the output data table to contain estimates of the parameter values and their standard errors for each model whose parameter estimation process converges.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1724. The *libref* must be identical to the *libref* that you specify in the DATA= option.

For more information about the variables in this data table, see the section “[OUTEST= Data Table](#)” on page 1840.

OUTMODELINFO*=libref.data-table*

names the output data table to contain the information about each candidate distribution.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1724. The *libref* must be identical to the *libref* that you specify in the DATA= option.

For more information about the variables in this data table, see the section “[OUTMODELINFO= Data Table](#)” on page 1843.

OUTSTAT*=libref.data-table*

names the output data table to contain the values of statistics of fit for each model whose parameter estimation process converges.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1724. The *libref* must be identical to the *libref* that you specify in the DATA= option.

For more information about the variables in this data table, see the section “[OUTSTAT= Data Table](#)” on page 1843.

OUTSTORE*=libref.data-table*

names the output data table to contain the context and results of the severity model estimation process in the form of an item store, which has a binary format that cannot be modified. You can specify this item store data table in a subsequent PROC CCDM step by using the SEVERITYSTORE= option.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option

and the section “Using CAS Sessions and CAS Engine Librefs” on page 1724. The *libref* must be identical to the *libref* that you specify in the DATA= option.

PLOTS <(global-plot-options)> <=plot-request-option>

PLOTS <(global-plot-options)> <=(plot-request-option ... plot-request-option)>

specifies which graphical output to create. If you specify more than one *global-plot-option*, then separate them with spaces and enclose them in parentheses. If you specify more than one *plot-request-option*, then separate them with spaces and enclose them in parentheses.

When the active CAS session contains more than one worker node, PROC SEVSELECT prepares the plots by using an aggregation algorithm that merges the plot data from individual worker nodes. Each worker node in turn prepares its plot data by using a sample of the local data to compute the empirical quantiles for a uniform grid of points along the empirical distribution function (EDF) axis; this data sample is the same sample that it uses to compute the EDF-based fit statistics. So the plots are approximate. You can still use them to compare the fits of different distributions visually. However, to get a more accurate visual comparison, it is recommended that you request plots only when you run the PROC SEVSELECT step in a CAS session that uses no more than one worker node.

You can specify the following *global-plot-options*:

HISTOGRAM

plots the histogram of the response variable on the probability density function (PDF) plots.

KERNEL

plots the kernel estimate of the probability density of the response variable on the PDF plots.

You can specify the following *plot-request-options*:

ALL

creates all the graphical output.

CDF

creates a plot that compares the cumulative distribution function (CDF) estimates of all the candidate distribution models to the empirical distribution function (EDF) estimate. The plot does not contain CDF estimates for models whose parameter estimation process does not converge.

CDFPERDIST

creates a plot of the CDF estimates of each candidate distribution model. A plot is not created for models whose parameter estimation process does not converge.

CONDITIONALPDF <(cpdf-options)>

CONDPDF <(cpdf-options)>

creates a plot that compares the conditional probability density function (PDF) estimates of all the candidate distribution models. The plot does not contain conditional PDF estimates for models whose parameter estimation process does not converge.

A conditional PDF of a loss random variable Y in an interval $(Y_l, Y_r]$ is the probability that a specific loss value is observed, given that the loss values belong to that interval. Formally, the conditional PDF of y , denoted by $f^c(y)$, for the $(Y_l, Y_r]$ interval is defined as $f^c(y) = \Pr[Y = y | Y_l < Y \leq Y_r]$. If $f(y)$ and $F(y)$ denote the PDF and CDF, respectively, at loss value y , then $f^c(y)$ for the $(Y_l, Y_r]$ interval is computed as $f^c(y) = f(y)/(F(Y_r) - F(Y_l))$. The scaling

factor of $1/(F(Y_r) - F(Y_l))$ ensures that the conditional PDF is a true PDF that integrates to 1 in the $(Y_l, Y_r]$ interval.

The conditional PDF comparison plot contains at most three regions (intervals) of mutually exclusive ranges of the loss variable's value,

- left-tail: $(y_{\min} - \epsilon, L]$
- center: $(L, R]$
- right-tail: $(R, y_{\max}]$

where $y_{\min}$ and $y_{\max}$ denote the smallest and largest values, respectively, of the loss variable in the DATA= data table and ϵ denotes a small machine-precision constant for a double-precision value.

You can specify the following *cpdf-options* to control how the values of L and R are computed and which regions are displayed:

LEFTQ=*number*

LEFT=*number*

L=*number*

specifies a *number*, which must be between 0 and 1, that sets the CDF value at the end of the left-tail region. The left-tail region always starts at the minimum loss variable value in the DATA= data table. If *number* is p_l , then PROC SEVSELECT sets L (the end of the left-tail region) equal to the $100p_l$ th percentile, which is an empirical estimate of $Q(p_l)$, where Q denotes the quantile function.

RIGHTQ=*number*

RIGHT=*number*

R=*number*

specifies a *number*, which must be between 0 and 1, that sets the CDF value at the start of the right-tail region. The right-tail region always ends at the maximum loss variable value in the DATA= data table. If *number* is p_r , then PROC SEVSELECT sets R (the start of the right-tail region) equal to the $100p_r$ th percentile, which is an empirical estimate of $Q(p_r)$, where Q denotes the quantile function.

SHOWREGION=(*region-options*)

SHOW=(*region-options*)

specifies one or more regions to display in the plot. You can specify any combination of the following *region-options*. If you specify more than one *region-option*, you must enclose the *region-options* in parentheses.

- | | |
|-------------------|---|
| CENTER C | displays the center region of the plot, which is the region between the end of the left-tail region and the beginning of the right-tail region. If you specify this option, you must also specify valid values for both the LEFTQ= and RIGHTQ= options. |
| LEFT L | displays the left-tail region of the plot. If you specify this option, you must also specify a valid value for the LEFTQ= option. |
| RIGHT R | displays the right-tail region of the plot. If you specify this option, you must also specify a valid value for the RIGHTQ= option. |

If you do not specify the `SHOWREGION` option, then PROC SEVSELECT determines the default displayed regions as follows:

- If you do not specify either the `LEFTQ=` or `RIGHTQ=` option, then this is equivalent to specifying (`LEFTQ=0.25 RIGHTQ=0.75`), and PROC SEVSELECT displays all three regions (left-tail, center, and right-tail).
- If you specify valid values for both the `LEFTQ=` and `RIGHTQ=` options, then PROC SEVSELECT displays all three regions (left-tail, center, and right-tail).
- If you specify a valid value for the `LEFTQ=` option but do not specify the `RIGHTQ=` option, then PROC SEVSELECT displays two regions: the left-tail region and the remaining region that combines the center and right-tail regions.
- If you specify a valid value for the `RIGHTQ=` option but do not specify the `LEFTQ=` option, then PROC SEVSELECT displays two regions: the right-tail region and the remaining region that combines the center and left-tail regions.

NOTE: PROC SEVSELECT computes the percentile estimates, which serve as region boundaries L and R , by using the EDF estimates that it computes by using the EDF estimation process as described in the section “[Empirical Distribution Function Estimation Methods](#)” on page 1799. For large input data tables or distributed mode of execution, the EDF estimates are approximate; therefore, the percentile estimates are also approximate—that is, they are not the same as the percentile estimates that would have been computed if the entire input data table were used.

CONDITIONALPDFPERDIST <(cpdf-options)>

CONDDPDFDIST <(cpdf-options)>

creates a plot of the conditional PDF estimates of each candidate distribution model. A plot is not created for models whose parameter estimation process does not converge.

The *cpdf-options* are identical to those listed for the [CONDITIONALPDF](#) plot option, except that they are interpreted in the context of each candidate distribution individually. You can specify a different set of values for the *cpdf-options* in the `CONDITIONALPDFPERDIST` option than you specify in the `CONDITIONALPDF` option.

NONE

creates none of the graphical output. If you specify this option, then it overrides all the other *plot-request-options*. The default graphical output is also suppressed.

PDF

creates a plot that compares the probability density function (PDF) estimates of all the candidate distribution models. The plot does not contain PDF estimates for models whose parameter estimation process does not converge.

PDFPERDIST

creates a plot of the PDF estimates of each candidate distribution model. A plot is not created for models whose parameter estimation process does not converge.

PP

creates the probability-probability plot (known as the P-P plot), which compares the CDF estimate of each candidate distribution model to the empirical distribution function (EDF). The data that are shown in this plot are used for computing the EDF-based statistics of fit.

QQ

creates the quantile-quantile plot (known as the Q-Q plot), which compares the empirical quantiles to the quantiles of each candidate distribution model.

PRINT *<(global-display-options) > <=display-option>*

PRINT *<(global-display-options) > <= (display-option ... display-option) >*

specifies which output to display. If you specify more than one *display-option*, then separate them with spaces and enclose them in parentheses.

You can specify the following *global-display-options*:

ONLY

turns off the default displayed output and displays only the requested output.

SYMMETRIC=*< symmetric-matrix-print-options >*

SYMM=*< symmetric-matrix-print-options >*

controls the contents of tables that display a symmetric matrix. You can specify the following *symmetric-matrix-print-options*:

SHAPE=*shape*

specifies the shape of the symmetric matrix. You can specify one of the following *shape* values:

FULL | F displays the full matrix.

LOWER | L displays the lower triangular matrix.

UPPER | U displays the upper triangular matrix.

By default, SHAPE=FULL.

NOUNITDIAG | ND

turns off the display of the diagonal for symmetric matrices that have a unit diagonal. By default, PROC SEVSELECT displays the unit diagonal.

If you do not specify the SYMMETRIC= option, then the procedure uses SYMMETRIC=(SHAPE=FULL) by default.

You can specify the following *display-options*:

ALL

displays all the output.

ALLFITSTATS

displays the comparison of all the statistics of fit for all the models in one table. The table does not include the models whose parameter estimation process does not converge.

CONVSTATUS

displays the convergence status of the parameter estimation process.

CORRB**CORR**

displays the estimated correlation matrix of parameters. You can control the display of this matrix by specifying the SHAPE= and NUNITDIAG suboptions of the SYMMETRIC= *global-display-option*.

COVB**COV**

displays the estimated variance-covariance matrix of parameters. You can control the display of this matrix by specifying the SHAPE= suboption of the SYMMETRIC= *global-display-option*.

DESCSTATS

displays the descriptive statistics for the response variable. If you specify the SCALEMODEL statement, then this option also displays the descriptive statistics for the regression effects that do not contain a CLASS variable.

DISTINFO

displays the information about each specified distribution. For each distribution, the information includes the name, description, validity status, and number of distribution parameters.

ESTIMATES**PARMEST**

displays the final estimates of parameters. The estimates are not displayed for models whose parameter estimation process does not converge.

ESTIMATIONDETAILS

displays the details of the estimation process for all the models in one table.

INITIALVALUES

displays the initial values and bounds used for estimating each model.

NLOHISTORY

displays the iteration history of the nonlinear optimization process used for estimating the parameters.

NLOSUMMARY

displays the summary of the nonlinear optimization process used for estimating the parameters.

NONE

displays none of the output. If you specify this option, then it overrides all other display options. The default displayed output is also suppressed.

SELECTION**SELECT**

displays the model selection table.

STATISTICS**FITSTATS**

displays the statistics of fit for each model. The statistics of fit are not displayed for models whose parameter estimation process does not converge.

If you do not specify the PRINT= option or if you do not specify the ONLY *global-display-option*, then the default displayed output is equivalent to specifying PRINT=(SELECTION CONVSTATUS NLOSUMMARY STATISTICS ESTIMATES).

PRINTINTERNALNAMES

prints internal names that are assigned to parameters.

SELECTOUT

specifies that only the regression parameters that correspond to the selected effects be written to the OUTEST= data table. If you omit this option, then the special missing value .N is written for such regression parameters. For more information, see the section “[OUTEST= Data Table](#)” on page 1840.

This option has no effect if you do not specify the SCALEMODEL and SELECTION statements.

VARDEF=DF | N

specifies the denominator to use for computing the covariance estimates. You can specify one of the following values:

DF specifies that the number of nonmissing observations minus the model degrees of freedom (number of parameters) be used.

N specifies that the number of nonmissing observations be used.

For more information about the covariance estimation, see the section “[Estimating Covariance and Standard Errors](#)” on page 1792.

ZEROEST

specifies that zero be written to the OUTEST= data table as an estimate of the regression parameter that is not in the final model (because it is either collinear or not selected). If you omit this option, then the special missing value .R or .N is written for such regression parameters. For more information, see the section “[OUTEST= Data Table](#)” on page 1840.

This option has no effect if you do not specify the SCALEMODEL statement.

The following *options* control the model estimation and selection process:

EMPIRICALCDF=method**EDF=method**

specifies the method to use for computing the nonparametric or empirical estimate of the cumulative distribution function of the data. You can specify one of the following values for *method*:

AUTOMATIC | AUTO

specifies that the method be chosen automatically based on the data specification.

If you do not specify any censoring or truncation, then the standard empirical estimation method (STANDARD) is chosen. If you specify both right-censoring and left-censoring, then Turnbull's estimation method (TURNBULL) is chosen. For all other combinations of censoring and truncation, the Kaplan-Meier method (KAPLANMEIER) is chosen.

KAPLANMEIER | KM

specifies that the product limit estimator proposed by Kaplan and Meier (1958) be used. Specification of this method has no effect when you specify both right-censoring and left-censoring.

MODIFIEDKM | MKM <(options)>

specifies that the modified product limit estimator be used. Specification of this method has no effect when you specify both right-censoring and left-censoring.

This method allows Kaplan-Meier's product limit estimates to be more robust by ignoring the contributions to the estimate due to small risk-set sizes. The risk set is the set of observations at the risk of failing, where an observation is said to fail if it has not been processed yet and might experience censoring or truncation. You can specify the minimum risk-set size that makes it eligible to be included in the estimation either as an absolute lower bound on the size (RSLB= option) or a relative lower bound determined by the formula cn^α proposed by Lai and Ying (1991). You can specify the values of c and α by using the C= and ALPHA= options, respectively. By default, the relative lower bound is used with values of $c = 1$ and $\alpha = 0.5$. However, you can modify the default by using the following *options*:

ALPHA | A=number

specifies the value to use for α when the lower bound on the risk set size is defined as cn^α . This value must satisfy $0 < \alpha < 1$.

C=number

specifies the value to use for c when the lower bound on the risk set size is defined as cn^α . This value must satisfy $c > 0$.

RSLB=number

specifies the absolute lower bound on the risk set size to be included in the estimate.

NOTURNBULL

specifies that the method be chosen automatically based on the data specification and that Turnbull's method not be used. This option is the default.

This method first replaces each left-censored or interval-censored observation with an uncensored observation. If the resulting set of observations has any truncated or right-censored observations, then the Kaplan-Meier method (KAPLANMEIER) is chosen. Otherwise, the standard empirical estimation method (STANDARD) is chosen. The observations are modified only for the purpose of computing the EDF estimates; the modification does not affect the parameter estimation process.

STANDARD | STD

specifies that the standard empirical estimation method be used. If you specify both right-censoring and left-censoring, then the specification of this method has no effect. If you specify any other combination of censoring or truncation effects, then this method ignores such effects, and can thus result in estimates that are more biased than those obtained with other methods that are more suitable for censored or truncated data.

TURNBULL | EM <(options)>

specifies that the Turnbull's method be used. This method is used when you specify both right-censoring and left-censoring. An iterative expectation-maximization (EM) algorithm proposed by Turnbull (1976) is used to compute the empirical estimates. If you also specify truncation, then the modification suggested by Frydman (1994) is used.

This method is used if you specify both right-censoring and left-censoring and if you explicitly specify the `EMPIRICALCDF=TURNBULL` option.

You can modify the default behavior of the EM algorithm by using the following *options*:

ENSUREMLE

specifies that the final EDF estimates be maximum likelihood estimates. The Kuhn-Tucker conditions are computed for the likelihood maximization problem and checked to ensure that EM algorithm converges to maximum likelihood estimates. The method generalizes the method proposed by Gentleman and Geyer (1994) by taking into account any truncation information that you might specify.

EPS=number

specifies the maximum relative error to be allowed between estimates of two consecutive iterations. This criterion is used to check the convergence of the algorithm. If you do not specify this option, then PROC SEVSELECT uses a default value of 1.0E-8.

MAXITER=number

specifies the maximum number of iterations to attempt to find the empirical estimates. If you do not specify this option, then PROC SEVSELECT uses a default value of 500.

ZEROPROB=number

specifies the threshold below which an empirical estimate of the probability is considered zero. This option is used to decide if the final estimate is a maximum likelihood estimate. This option does not have an effect if you do not specify the ENSUREMLE option. If you specify the ENSUREMLE option, but do not specify this option, then PROC SEVSELECT uses a default value of 1.0E-8.

For more information about each of the methods, see the section “[Empirical Distribution Function Estimation Methods](#)” on page 1799.

INITSAMPLE (initsample-option)**INITSAMPLE (initsample-option ... initsample-option)**

specifies that a sample of the input data be used for initializing the distribution parameters. If you specify more than one *initsample-option*, then separate them with spaces.

When you do not specify initial values for the distribution parameters, PROC SEVSELECT needs to compute the empirical distribution function (EDF) estimates as part of the default method for parameter

initialization. The EDF estimation process can be expensive, especially when you specify censoring or truncation effects for the loss variable. Furthermore, it is not amenable to parallelism due to the sequential nature of the algorithm for truncation effects. You can use the INITSAMPLE option to specify that only a fraction of the input data be used in order to reduce the time taken to compute the EDF estimates. PROC SEVSELECT uses the uniform random sampling method to select the sample, the size and randomness of which are controlled by the following *initsample-options*:

FRACTION=number

specifies the fraction, between 0 and 1, of the input data to be used for sampling.

SEED=number

specifies the seed to be used for the uniform random number generator. This option enables you to select the same sample from the same input data across different runs of PROC SEVSELECT, which can be useful for replicating the results across different runs. If you do not specify the seed value, PROC SEVSELECT generates a seed that is based on the system clock.

SIZE=number

specifies the size of the sample. If the data are distributed across different nodes, then this size applies to the sample that is prepared at each node. For example, let the input data table of size 100,000 observations be distributed across 10 nodes such that each node has 10,000 observations. If you specify SIZE=1000, then each node computes a local EDF estimate by using a sample of size 1,000 selected randomly from its 10,000 observations.

If you specify both the SIZE= and FRACTION= options, then the value that you specify in the SIZE= option is used and the FRACTION= option is ignored.

If you do not specify the INITSAMPLE option, then a uniform random sample of at most 10,000 observations is used for EDF estimation on each node that has observations.

NOBALANCETHREADS

NOBALTHR

specifies that the observations do not need to be balanced among threads of computation.

Starting with SAS Econometrics 8.5, the parameter estimation phase balances the observations among all threads of computation by default; this helps achieve significantly faster estimation times. However, this balanced processing can cause small numerical differences compared to the prior releases that did not balance observations among threads of computation. You should specify this option only if you want to restore the numerical behavior of releases prior to 8.5. However, use of this option can lead to a significantly slower estimation phase in most cases, and therefore it is strongly discouraged.

NOCONSTFITSTATS

NOCONSTSOF

excludes the constant distribution parameters, if any are specified, from the calculations of likelihood-based fit statistics (AIC, AICC, and SBC) that depend on the number of model parameters.

By default, constant distribution parameters are assumed to be estimable (even if PROC SEVSELECT does not estimate them) and are included in the calculations of the likelihood-based fit statistics.

OBJECTIVE=*symbol-name*

names the symbol that represents the objective function in the SAS programming statements that you specify. For each model to be estimated, PROC SEVSELECT executes the programming statements to compute the value of this symbol for each observation. The values are added across all observations to obtain the value of the objective function. The optimization algorithm estimates the model parameters such that the objective function value is *minimized*. A separate optimization problem is solved for each candidate distribution. If you specify a BY statement, then a separate optimization problem is solved for each candidate distribution within each BY group.

For more information about writing SAS programming statements to define your own objective function, see the section “[Custom Objective Functions](#)” on page 1836.

SELECTNLOTECH=CONGRA | DBLDOG | NEWRAP | NMSIMP | NRRIDG | QUANEW | TRUREG

specifies the nonlinear optimization technique to use for the intermediate steps of the regression effect selection process. The can specify one of the following techniques:

CONGRA	performs a conjugate-gradient optimization.
DBLDOG	performs a version of double-dogleg optimization.
NEWRAP	performs a Newton-Raphson optimization that combines a line-search algorithm with ridging.
NMSIMP	performs a Nelder-Mead simplex optimization.
NRRIDG	performs a Newton-Raphson optimization with ridging.
QUANEW	performs a dual quasi-Newton optimization.
TRUREG	performs a trust region optimization.

This option has no effect if you do not specify the SELECTION statement.

If you omit this option, then the default optimization technique for the intermediate steps of the selection process is the same technique that PROC SEVSELECT uses to estimate the parameters of the final selected model. You can control that technique by specifying the [TECHNIQUE=](#) option in the [NLOPTIONS](#) statement.

BY Statement

BY *variable-list* ;

A BY statement can be used in the SEVSELECT procedure to process the input data table in groups of observations defined by the BY variables.

If you specify the BY statement, PROC SEVSELECT does not need the input data table to be sorted in the order of the BY variables. As a consequence, the NOTSORTED and DESCENDING options have no effect and PROC SEVSELECT ignores them.

CLASS Statement

CLASS *variable* <(*options*)> . . . <*variable* <(*options*)> > </ *global-options* > ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. You can list the response variable for binary models in the CLASS statement, but this is not required. Table 26.2 summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “CLASS Statement” on page 56 in Chapter 4, “Shared Concepts.”

Table 26.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable
SPLIT	Splits levels of CLASS variables into independent effects

DISPLAY Statement

DISPLAY <*table-list*> </ *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “Displayed Output” on page 1845. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *Selection-Summary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the output table name.

The ODS table names that you can specify are listed in the section “[Displayed Output](#)” on page 1845. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output tables for all display tables. The name of the created output table is the same as the name of the corresponding display table. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing output table of the same name.

REPEATED

replicates all output tables on all nodes.

DIST Statement

DIST *distribution-name-or-keyword* < (*distribution-option*) < *distribution-name-or-keyword* < (*distribution-option*) > > ... > < / *options* > ;

The DIST statement specifies candidate distributions to be estimated by the SEVSELECT procedure. You can specify multiple DIST statements, and each statement can contain one or more distribution specifications.

For your convenience, PROC SEVSELECT provides the following 10 different predefined distributions (the name in parentheses is the name to use in the DIST statement): Burr (BURR), exponential (EXP), gamma (GAMMA), generalized Pareto (GPD), inverse Gaussian or Wald (IGAUSS), lognormal (LOGN), Pareto (PARETO), Tweedie (TWEEDIE), scaled Tweedie (STWEEDIE), and Weibull (WEIBULL). These are described in detail in the section “[Predefined Distributions](#)” on page 1779.

You can specify any of the predefined distributions or any distribution that you have defined. If a distribution that you specify is not a predefined distribution, then you must submit the CMPLIB= system option with appropriate libraries before you submit the PROC SEVSELECT step to enable the procedure to find the functions associated with your distribution. The predefined distributions are defined in the Sashelp.Svrtldist library. However, you are not required to specify this library in the CMPLIB= system option. For more information about defining your own distributions, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811.

As a convenience, you can also use a shortcut keyword to indicate a list of distributions. You can specify one or more of the following keywords:

ALL

specifies all the predefined distributions and the distributions that you have defined in the libraries that you specify in the CMPLIB= system option. In addition to the eight predefined distributions included by the _PREDEFINED_ keyword, this list also includes the Tweedie and scaled Tweedie distributions that are defined in the Sashelp.Svrtldist library.

PREDEFINED

specifies the list of eight predefined distributions: BURR, EXP, GAMMA, GPD, IGAUSS, LOGN, PARETO, and WEIBULL. Although the TWEEDIE and STWEEDIE distributions are available in the Sashelp.Svrtldist library along with these eight distributions, they are not included by this keyword. If you want to fit the TWEEDIE and STWEEDIE distributions, then you must specify them explicitly or use the _ALL_ keyword.

USER

specifies the list of all the distributions that you have defined in the libraries that you specify in the CMPLIB= system option. This list does not include the distributions defined in the Sashelp.Svrtldist library, even if you specify the Sashelp.Svrtldist library in the CMPLIB= option.

The use of these keywords, especially `_ALL_`, can result in a large list of distributions, which might take a longer time to estimate. A warning is printed to the SAS log if the number of total distribution models to estimate exceeds 10.

The following *distribution-option* values can be used in the DIST statement for a distribution name that is not a shortcut keyword:

INIT=(name=value ... name=value)

specifies the initial values to be used for the distribution parameters to start the parameter estimation process. You must specify the values by parameter names, and the parameter names must match the names used in the model definition. For example, let a model M's definition contain an `M_PDF` function with the following signature:

```
function M_PDF(x, alpha, beta);
```

For this model, the names `alpha` and `beta` must be used for the INIT option. The names are case-insensitive. If you do not specify initial values for some parameters in the INIT statement, then a default value of 0.001 is assumed for those parameters. If you specify an incorrect parameter, PROC SEVSELECT prints a warning to the SAS log and does not fit the model. All specified values must be nonmissing.

If you are modeling regression effects, then the initial value of the first distribution parameter (`alpha` in the preceding example) should be the initial *base* value of the scale parameter or log-transformed scale parameter. For more information, see the section “[Estimating Regression Effects](#)” on page 1794.

The use of INIT= option is one of the three methods available for initializing the parameters. For more information, see the section “[Parameter Initialization](#)” on page 1793. If none of the initialization methods is used, then PROC SEVSELECT initializes all parameters to 0.001.

You can specify the following *options* in the DIST statement:

INFUNCDEF="table-name"

INTAB="table-name"

INFUNCDEF=table-name<(CASLIB="library")>

INTAB=table-name<(CASLIB="library")>

specifies a data table that contains the distribution function definitions. If you use the first form to specify the data table, then PROC SEVSELECT assumes that the table is in the library that is active in the current CAS session. To specify a different library, use the second form.

LISTONLY

prints the list of all candidate distributions to the SAS log without doing any further processing on them. This option is especially useful when you use a shortcut keyword to include a list of distributions. It enables you to find out which distributions are included by the keyword.

OUTFUNCDEF="table-name"<option>

OUTTAB="table-name" <option>

specifies the output data table to write the distribution function definitions to. The table is created in the library that is active for the current CAS session.

This option is useful for creating a data table that you can use to invoke the actions in the **severity** action set from the PROC CAS, Lua, or Python client.

You can specify the following *option*:

GLOBAL

PROMOTED

promotes the data table to make it available globally.

If you do not specify this option, the table is session-local, which means it is accessible only to the current CAS session that is executing the PROC SEVSELECT step. When the current CAS session is terminated, the UTFUNCDEF= table is deleted.

When you specify this option, the table persists in the memory of the CAS server and is available to other CAS sessions that you might start after the CAS session that is executing the current PROC SEVSELECT step is terminated.

If you do not specify the UTFUNCDEF= or INFUNCDEF= option, then PROC SEVSELECT creates a session-local data table named svrtdist in the library that is active for the current CAS session.

VALIDATEONLY

VALIDATEONLY (NOSCALEMODEL)

checks all candidate distributions for validity without doing any further processing on them. The first form checks whether each distribution's first parameter is a scale parameter, which is a requirement for fitting a scale regression model for that distribution. You can disable that check by using the second form, which specifies the NOSCALEMODEL option; this is useful if you do not want to fit a scale regression model for a distribution.

PROC SEVSELECT writes the distribution information to the DistributionInfo ODS table.

This option is especially useful when you specify your own distributions or when you specify the `_USER_` or `_ALL_` keyword in the DIST statement. It enables you to check whether your custom distribution definitions satisfy PROC SEVSELECT's requirements for the specified modeling task. It is recommended that you specify the SCALEMODEL statement if you intend to fit a model with regression effects, because the SCALEMODEL statement instructs PROC SEVSELECT to perform additional checks to validate whether regression effects can be modeled on each candidate distribution.

This option is also useful in conjunction with the `UTFUNCDEF=` option when you define or update a custom distribution and want to write its definition to the CAS table that you can use to invoke the actions in the **severity** action set from the PROC CAS, Lua, or Python client.

EFFECT Statement

EFFECT *name=effect-type* (*variables* < / *options* >) ;

The EFFECT statement enables you to construct special collections of columns for design matrices. These collections are referred to as *constructed effects* to distinguish them from the usual model effects that are formed from continuous or classification variables, as discussed in the section “GLM Parameterization of Classification Variables and Effects” on page 94 in Chapter 4, “Shared Concepts.”

You can specify the following *effect-types*:

COLLECTION	specifies a collection effect that defines one or more variables as a single effect that has multiple degrees of freedom. The variables in a collection are considered as a unit for purposes of estimation and inference.
MULTIMEMBER MM	specifies a multimember classification effect whose levels are determined by one or more variables that appear in a CLASS statement.
POLYNOMIAL POLY	specifies a multivariate polynomial effect in the specified numeric variables.
SPLINE	specifies a regression spline effect whose columns are univariate spline expansions of one or more variables. A spline expansion replaces the original variable with an expanded or larger set of new variables.

Table 26.3 summarizes the *options* available in the EFFECT statement.

Table 26.3 EFFECT Statement Options

Option	Description
Collection Effects Options	
DETAILS	Displays the constituents of the collection effect
Multimember Effects Options	
DETAILS	Displays the levels of the multimember effect
NOEFFECT	Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns
STDIZE	Standardizes the design matrix entries so that each observation has a sum of 1
WEIGHT=	Specifies the weight variable for the contributions of each classification effect
Polynomial Effects Options	
DEGREE=	Specifies the degree of the polynomial
DETAILS	Displays details of the specified polynomial
MDEGREE=	Specifies the maximum degree of any variable in a term of the polynomial
NOSEPARATE	Treats the polynomial as a single effect with multiple degrees of freedom
STANDARDIZE=	Specifies centering and scaling suboptions for the variables that define the polynomial

Table 26.3 *continued*

Option	Description
Spline Effects Options	
BASIS=	Specifies the type of basis (B-spline basis or truncated power function basis) for the spline effect
DATABOUNDARY	Uses the extremes of the data as boundary knots for a B-spline basis
DEGREE=	Specifies the degree of the spline effect
DETAILS	Displays the knots and locations for each spline basis function
KNOTMAX=	Requests equally spaced right-side boundary knots starting at the variables' maximum and ending at the KNOTMAX= value
KNOTMETHOD=	Specifies how to construct the knots for the spline effect
KNOTMIN=	Requests equally spaced left-side boundary knots starting at the KNOTMIN= value and ending at the variables' minimum value
NATURALCUBIC	Specifies a natural cubic spline basis for the spline effect
SEPARATE	Treats the spline basis for each variable as a separate effect when multiple variables are specified
SPLIT	Treats each design matrix column as a separate effect for selection methods

For more information about the syntax of these *effect-types* and how columns of constructed effects are computed, see the section “[EFFECT Statement](#)” on page 62 in Chapter 4, “[Shared Concepts](#).” Each *effect-type* has a **DETAILS** option that you can use to print tables that contain the details of the effect. However, these tables are produced only if you also specify **PRINT=ALL** option in the PROC SEVSELECT statement.

LOSS Statement

LOSS < *response-variable-name* > < / *censoring-truncation-options* > ;

The LOSS statement specifies the name of the response or loss variable whose distribution needs to be modeled. You can also specify additional options to indicate any truncation or censoring of the response. The specification of response variable is optional if you specify at least one type of censoring. You must specify a response variable if you do not specify any censoring. If you specify more than one LOSS statement, then the first statement is used.

All the analysis variables that you specify in this statement must be present in the input data table that you specify by using the **DATA=** option in the PROC SEVSELECT statement. The response variable is expected to have nonmissing values. If the variable has a missing value in an observation, then a warning is written to the SAS log and that observation is ignored.

The following *censoring-truncation-options* can be used in the LOSS statement:

LEFTCENSORED | LC=variable-name**LEFTCENSORED | LC=number**

specifies the left-censoring variable or a global left-censoring limit.

You can use the *variable-name* argument to specify a data table variable that contains the left-censoring limit. If the value of this variable is missing, then PROC SEVSELECT assumes that such observations are not left-censored.

Alternatively, you can use the *number* argument to specify a left-censoring limit value that applies to all the observations in the data table. This limit must be a nonzero positive number.

By the definition of left-censoring, an exact value of the response is not known when it is less than or equal to the left-censoring limit. If you specify the response variable and the value of that variable is less than or equal to the value of the left-censoring limit for some observations, then PROC SEVSELECT treats such observations as left-censored and the value of the response variable is ignored. If you specify the response variable and the value of that variable is greater than the value of the left-censoring limit for some observations, then PROC SEVSELECT assumes that such observations are not left-censored and the value of the left-censoring limit is ignored.

If you specify both right-censoring and left-censoring limits, then the left-censoring limit must be greater than or equal to the right-censoring limit. If both limits are identical, then the observation is assumed to be uncensored.

For more information about left-censoring, see the section “[Censoring and Truncation](#)” on page 1789.

LEFTTRUNCATED | LT=variable-name < (left-truncation-option) >**LEFTTRUNCATED | LT=number < (left-truncation-option) >**

specifies the left-truncation variable or a global left-truncation threshold.

You can use the *variable-name* argument to specify a data table variable that contains the left-truncation threshold. If the value of this variable is missing or 0 for some observations, then PROC SEVSELECT assumes that such observations are not left-truncated.

Alternatively, you can use the *number* argument to specify a left-truncation threshold that applies to all the observations in the data table. This threshold must be a nonzero positive number.

It is assumed that the response variable contains the observed values. By the definition of left-truncation, you can observe only a value that is greater than the left-truncation threshold. If a response variable value is less than or equal to the left-truncation threshold, a warning is printed to the SAS log, and the observation is ignored. For more information about left-truncation, see the section “[Censoring and Truncation](#)” on page 1789.

You can specify the following *left-truncation-option* for an alternative interpretation of the left-truncation threshold:

PROBOBSERVED | POBS=number

specifies the probability of observability, which is defined as the probability that the underlying severity event is observed (and recorded) for the specified left-threshold value.

The specified *number* must lie in the (0.0, 1.0] interval. A value of 1.0 is equivalent to specifying that there is no left-truncation, because it means that no severity events can occur with a value less than or equal to the threshold. If you specify value of 1.0, PROC SEVSELECT prints a warning to the SAS log and proceeds by assuming that LEFTTRUNCATED= option is not specified.

For more information, see the section “[Probability of Observability](#)” on page 1790.

RIGHTCENSORED | **RC=***variable-name*

RIGHTCENSORED | **RC=***number*

specifies the right-censoring variable or a global right-censoring limit.

You can use the *variable-name* argument to specify a data table variable that contains the right-censoring limit. If the value of this variable is missing, then PROC SEVSELECT assumes that such observations are not right-censored.

Alternatively, you can use the *number* argument to specify a right-censoring limit value that applies to all the observations in the data table. This limit must be a nonzero positive number.

By the definition of right-censoring, an exact value of the response is not known when it is greater than or equal to the right-censoring limit. If you specify the response variable and the value of that variable is greater than or equal to the value of the right-censoring limit for some observations, then PROC SEVSELECT treats such observations as right-censored and the value of the response variable is ignored. If you specify the response variable and the value of that variable is less than the value of the right-censoring limit for some observations, then PROC SEVSELECT assumes that such observations are not right-censored and the value of the right-censoring limit is ignored.

If you specify both right-censoring and left-censoring limits, then the left-censoring limit must be greater than or equal to the right-censoring limit. If both limits are identical, then the observation is assumed to be uncensored.

For more information about right-censoring, see the section “[Censoring and Truncation](#)” on page 1789.

RIGHTTRUNCATED | **RT=***variable-name*

RIGHTTRUNCATED | **RT=***number*

specifies the right-truncation variable or a global right-truncation threshold.

You can use the *variable-name* argument to specify a data table variable that contains the right-truncation threshold. If the value of this variable is missing for some observations, then PROC SEVSELECT assumes that such observations are not right-truncated.

Alternatively, you can use the *number* argument to specify a right-truncation threshold that applies to all the observations in the data table. This threshold must be a nonzero positive number.

It is assumed that the response variable contains the observed values. By the definition of right-truncation, you can observe only a value that is less than or equal to the right-truncation threshold. If a response variable value is greater than the right-truncation threshold, a warning is printed to the SAS log, and the observation is ignored. For more information about right-truncation, see the section “[Censoring and Truncation](#)” on page 1789.

NLOPTIONS Statement

NLOPTIONS *options* ;

The SEVSELECT procedure uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization of the likelihood function to obtain the estimates of distribution and regression parameters. You can use the NLOPTIONS statement to control different aspects of this optimization process. If you specify more than one NLOPTIONS statement, then the first statement is used.

For most problems, the default settings of the optimization process are adequate. However, in some cases it might be useful to change the optimization technique or to change the maximum number of iterations. The following statement uses the MAXITER= option to set the maximum number of iterations to 200 and uses the TECH= option to change the optimization technique to the double-dogleg optimization (DBLDOG) rather than the default technique, the trust region optimization (TRUREG), used in the SEVSELECT procedure:

```
nloptions tech=dbldog maxiter=200;
```

For more information about the options you can specify in the NLOPTIONS statement, see the section “[Optimization Options](#)” on page 84. For more information about the optimization methods, see the section “[Choosing an Optimization Algorithm](#)” on page 108.

OUTPUT Statement

OUTPUT < **OUT**=*libref.data-table* > *output-options* ;

The OUTPUT statement creates the output data table that contains the estimates of scoring functions and quantiles.

You must specify the following option:

OUT=*libref.data-table*

names the output data table for PROC SEVSELECT to use. You must specify this option before any other options. *libref.data-table* is a two-level name, where

<i>libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>libref</i> , see the section “ Using CAS Sessions and CAS Engine Librefs ” on page 1724.
---------------	---

<i>data-table</i>	specifies the name of the output data table.
-------------------	--

The *libref* must be identical to the *libref* that you specify in the **DATA**= option.

To control the contents of the OUT= data table, specify the following *output-options*:

COPYVARS=*variable-list*

specifies the names of the variables that you want to copy from the input DATA= data table to the OUT= data table. If you want to specify more than one name, then separate them by spaces.

If you specify the BY statement, then the BY variables are not automatically copied to the OUT= data table, so you must specify the BY variables in the COPYVARS= option.

FUNCTIONS=(*function*<(arg)><=*variable*> <*function*<(arg)><=*variable*>> ...)

specifies the scoring functions that you want to estimate. For each scoring function that you want to estimate, specify the suffix of the scoring function as the *function*. For each *function* that you specify in this option and for each distribution *D* that you specify in the DIST statement, the FCMP function *D_function* must be available in the search path that you specify by using the CMPLIB= system option. The signature of *D_function* must be identical to the signature of the required distribution function such as *D_CDF* or *D_LOGCDF*. For example, for the function 'FOO_BAR' to be a scoring function, you must specify the distribution 'FOO' in the DIST statement, and you must define 'FOO_BAR' in the following manner if the distribution 'FOO' has parameters named 'P1' and 'P2':

```
function FOO_BAR(y, P1, P2);
  /* Code to compute BAR by using y, P1, and P2 */
  R = <computed BAR>;
  return (R);
endsub;
```

You can then specify BAR as the *function* in the FUNCTIONS= option. For more information about the signature that defines a distribution function, see the description of the [D_CDF](#) function in the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811.

If you want to evaluate the scoring function at a specific value of the response variable, then specify a number *arg*, which is enclosed in parentheses immediately after the *function*. If you do not specify *arg* or if you specify a missing value as *arg*, then for each observation in the DATA= data table, PROC SEVSELECT computes the value *v* by using the following table and evaluates the scoring function at *v*, where *y*, *r*, and *l* denote the values of the response variable, right-censoring limit, and left-censoring limit, respectively:

Right-Censored	Left-Censored	<i>v</i>
No	No	<i>y</i>
No	Yes	<i>l</i>
Yes	No	<i>r</i>
Yes	Yes	$(l + r)/2$

You can specify the suffix of the variable that contains the estimate of the scoring function by specifying a valid SAS name as a *variable*. If you do not specify a *variable*, then PROC SEVSELECT uses *function* as the suffix of the variable name.

To illustrate the FUNCTIONS= option with an example, assume that you specify the following DIST and OUTPUT statements:

```
dist exp logn;
output out=mylib.score functions=(cdf pdf(1000)=f1000 mean);
```

Let both exponential (EXP) and lognormal (LOGN) distributions converge. If $\hat{\theta}$ is the final estimate of the scale parameter of the exponential distribution, then PROC SEVSELECT creates the following three scoring function variables for the exponential (EXP) distribution in the mylib.Score data table:

EXP_CDF	contains the CDF estimate $F_{\text{exp}}(v, \hat{\theta})$, where F_{exp} denotes the CDF of the exponential distribution and v is the value that is determined by the preceding table.
EXP_F1000	contains the PDF estimate $f_{\text{exp}}(1000, \hat{\theta})$, where f_{exp} denotes the PDF of the exponential distribution.
EXP_MEAN	contains the mean of the exponential distribution for the scale parameter $\hat{\theta}$.

Similarly, if $\hat{\mu}$ and $\hat{\sigma}$ are the final estimates of the log-scale and shape parameters of the lognormal distribution, respectively, then PROC SEVSELECT creates the following three scoring function variables for the lognormal (LOGN) distribution in the mylib.Score data table:

LOGN_CDF	contains the CDF estimate $F_{\text{logn}}(v, \hat{\mu}, \hat{\sigma})$, where F_{logn} denotes the CDF of the lognormal distribution and v is the value that is determined by the preceding table.
LOGN_F1000	contains the probability density function (PDF) estimate $f_{\text{logn}}(1000, \hat{\mu}, \hat{\sigma})$, where f_{logn} denotes the PDF of the lognormal distribution.
LOGN_MEAN	contains the mean of the lognormal distribution for the parameters $\hat{\mu}$ and $\hat{\sigma}$.

If you specify the SCALEMODEL statement, then the value of the scale parameter of a distribution depends on the values of the regression parameters. So it might be different for different observations. In this example, the values of $\hat{\theta}$ and $\hat{\mu}$ might vary by observation, which might cause the values of the EXP_F1000, EXP_MEAN, LOGN_F1000, and LOGN_MEAN variables to vary by observation. The values of the EXP_CDF and LOGN_CDF variables might vary not only because of the varying values of v but also because of the varying values of $\hat{\theta}$ and $\hat{\mu}$.

If you do not specify the SCALEMODEL statement, then the values of scoring functions for which you specify a nonmissing argument *arg* and scoring functions that do not depend on the response variable value do not vary by observation. In this example, the values of the EXP_F1000, EXP_MEAN, LOGN_F1000, and LOGN_MEAN variables do not vary by observation.

If a distribution does not converge, then the scoring function variables for that distribution contain missing values in all observations.

QUANTILES=*quantile-options*

specifies the quantiles that you want to estimate. To use this option, for each distribution that you specify in the DIST statement, the FCMP function *D_QUANTILE* must be defined in the search path that you specify by using the CMPLIB= system option.

You can specify the following *quantile-options*:

CDF=*CDF-values*

POINTS=*CDF-values*

specifies the CDF values at which you want to estimate the quantiles. *CDF-values* can be one or more numbers, separated by spaces. Each number must be in the interval (0,1).

NAMES=(*variable-names*)

specifies the suffixes of the names of the variables for each of the quantile estimates. If you specify n ($n \geq 0$) names in the *variable-names* option and k values in the CDF= option, and if $n < k$, then PROC SEVSELECT uses the n names to name the variables that correspond to the first n CDF values. For each of the remaining $k - n$ CDF values, p_i ($n < i \leq k$), PROC SEVSELECT creates a variable name P_t , where t is the text representation of $100p_i$ that is formed by retaining at most NDECIMAL= digits after the decimal point and replacing the decimal point with an underscore ('_').

NDECIMAL=*number*

specifies the number of digits to keep after the decimal point when PROC SEVSELECT creates the name of the quantile estimate variable. If you do not specify this option, then the default value is 3.

For example, assume that you specify the following DIST and OUTPUT statements:

```
dist burr;
output out=score quantiles=(cdf=0.9 0.975 0.995 names=(ninety var));
```

PROC SEVSELECT creates three quantile estimate variables, BURR_NINETY, BURR_VAR, and BURR_P99_5, in the mylib.Score data table for the Burr distribution. These variables contain the estimates of $Q_{\text{Burr}}(p, \hat{\theta}, \hat{\alpha}, \hat{\gamma})$, for $p = 0.9, 0.975$, and 0.995 , respectively, where Q_{Burr} denotes the quantile function and $\hat{\theta}$, $\hat{\alpha}$, and $\hat{\gamma}$ denote the parameter estimates of the Burr distribution.

If you specify the SCALEMODEL statement, then the quantile estimate might vary by observation, because the scale parameter of a distribution depends on the values of the regression parameters.

If you do not specify the SCALEMODEL statement, then the quantile estimates do not vary by observation, and if you do not specify any scoring functions in the FUNCTIONS= option whose estimates vary by observation, then the OUT= data table contains only one observation per BY group.

If a distribution does not converge, then the quantile estimate variables for that distribution contain missing values for all observations.

For more information about the variables and observations in the OUT= data table, see the section “[OUT= Data Table](#)” on page 1840.

OUTSCORELIB Statement

OUTSCORELIB <**OUTLIB=**> *fcmp-library-name* options ;

The OUTSCORELIB statement specifies the library to write scoring functions to. Scoring functions enable you to easily compute a distribution function on the fitted parameters of the distribution without going through a potentially complex process of extracting the fitted parameter estimates from other output, such as the OUTEST= data table that PROC SEVSELECT creates.

If you specify the SCALEMODEL statement and specify interaction or classification effects, then PROC SEVSELECT ignores the OUTSCORELIB statement and does not generate scoring functions. In other words, if you specify the SCALEMODEL statement, then PROC SEVSELECT generates scoring functions if you specify only singleton continuous effects in the SCALEMODEL statement.

You must specify the following option as the first option in the statement:

OUTLIB=*fcmp-library-name*

names the FCMP library to contain the scoring functions. PROC SEVSELECT writes the scoring functions to the FCMP library named *fcmp-library-name*. If a library or data set named *fcmp-library-name* already exists, PROC SEVSELECT deletes it before proceeding.

This option is similar to the OUTLIB= option that you would specify in a PROC FCMP statement, except that *fcmp-library-name* must be a two-level name, whereas the OUTLIB= option in the PROC FCMP statement requires a three-level name. The third level of a three-level name specifies the package to which the functions belong. You do not need to specify the package name in the *fcmp-library-name*, because PROC SEVSELECT automatically creates the package for you. By default, it creates a separate package for each distribution that has not failed to converge. Each package is named for a distribution. For example, if you define and fit a distribution named *mydist*, and if *mydist* does not fail to converge, then PROC SEVSELECT creates a package named *mydist* in the OUTLIB= library that you specify. Further, suppose that the definition of the *mydist* distribution contains three distribution functions, *mydist_PDF*(*x*,*Parm1*,*Parm2*), *mydist_LOGCDF*(*x*,*Parm1*,*Parm2*), and *mydist_XYZ*(*x*,*Parm1*,*Parm2*). If you specify the OUTSCORELIB statement

```
outscorelib outlib=sasuser.scorefunc;
```

then the library Sasuser.Scorefunc contains the following three functions in a package named *mydist*: *SEV_PDF*(*x*), *SEV_LOGCDF*(*x*), and *SEV_XYZ*(*x*).

The key feature of scoring functions is that they do not require the parameter arguments (*Parm1* and *Parm2* in this example). The fitted parameter estimates are encoded inside the scoring function so that you can compute or score the value of each function for a given value of the loss variable without having to know or extract the parameter estimates through some other means.

For convenience, you can omit the OUTLIB= portion of the specification and specify only the name, as in the following example:

```
outscorelib sasuser.scorefunc;
```

When the SEVSELECT procedure runs successfully, the *fcmp-library-name* is appended to the CMPLIB system option, so you can immediately start using the scoring functions in a DATA step or PROC FCMP step. Note that the generated scoring FCMP functions can be used only in a DATA step that runs on the client machine; you cannot use them in a DATA step that runs on the CAS server.

You can specify the following *options* in the OUTSCORELIB statement:

COMMONPACKAGE

ONEPACKAGE

creates one common package to contain all the scoring functions.

If you specify this option, then all the scoring functions are created in a package called *sevfit*. For each distribution function that has the name *distribution_suffix*, the name of the corresponding scoring function is formed as *SEV_suffix_distribution*. For example, the scoring function of the distribution function 'MYDIST_BAR' is named 'SEV_BAR_MYDIST'.

If you do not specify this option, then all scoring functions for a distribution are created in a package that has the same name as the distribution, and for each distribution function that has the name *distribution_suffix*, the name of the corresponding scoring function is formed as *SEV_suffix*. For example, the scoring function of the distribution function 'MYDIST_BAR' is named 'SEV_BAR'.

OUTBYID=*libref.data-table*

names the output data table to contain the unique identifier for each BY group. This unique identifier is used as part of the name of the package or scoring function for each distribution. This is a required option when you specify a BY statement in PROC SEVSELECT.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1724. The *libref* must be identical to the *libref* that you specify in the DATA= option.

The OUTBYID= data table contains one observation per BY group and a variable named *_ID_* in addition to the BY variables that you specify in the BY statement. The *_ID_* variable contains the unique identifier for each BY group. The identifier of the BY group is the decimal representation of the sequence number of the BY group. The first BY group has an identifier of 1, the second BY group has an identifier of 2, the tenth BY group has an identifier of 10, and so on.

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution, PROC SEVSELECT creates as many packages as the number of BY groups. The unique BY-group identifier is used as a suffix for the package name. For example, if your DATA= data table has three BY groups and you specify the OUTSCORELIB statement

```
outscorelib outlib=sasuser.byscorefunc outbyid=sasuser.byid;
```

then for the distribution 'MYDIST', the library Sasuser.Byscorefunc contains the three packages 'MYDIST1', 'MYDIST2', and 'MYDIST3', and each package contains one scoring function named 'SEV_BAR' for each distribution function named 'MYDIST_BAR'.

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, PROC SEVSELECT creates as many versions of the distribution function as the number of BY groups. The unique BY-group identifier is used as a suffix for the function name. Extending the previous example, if you specify the OUTSCORELIB statement with the COMMONPACKAGE option,

```
outscorelib outlib=sasuser.byscorefunc outbyid=sasuser.byid commonpackage;
```

then for the distribution function ‘MYDIST_BAR’ of the distribution ‘MYDIST’, the Sasuser.Byscorefunc library contains the following three scoring functions: ‘SEV_BAR_MYDIST1’, ‘SEV_BAR_MYDIST2’, and ‘SEV_BAR_MYDIST3’. All the scoring functions are created in one common package named *sevfit*.

For both the preceding examples, the data table Sasuser.Byid contains three observations, one for each BY group. The value of the *_ID_* variable is 1 for the first BY group, 2 for the second BY group, and 3 for the third BY group.

For more information about scoring functions, see the section “[Scoring Functions](#)” on page 1829.

RESTRICT Statement

```
RESTRICT restriction1 [, restriction2 ...] ;
```

The RESTRICT statement imposes linear restrictions on the regression effect parameter estimates of the scale model that is specified in the [SCALEMODEL](#) statement. The restrictions are applied to all distributions that are specified in the DIST statement, including user-supplied distributions. The RESTRICT statement allows you to restrict only the parameters that are associated with the regression effects in the SCALEMODEL statement. If you want to restrict the distribution parameters, then you can define your own distribution function in the FCMP procedure and specify the desired restrictions. For more information about defining your own distributions, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811.

You can specify any number of RESTRICT statements. Each *restriction* is written either as a single linear equation or as a comma-separated list of two or more linear equations. A restriction equation consists of an *expression*, followed by an equality *operator* (=) or an inequality *operator* (<, >, <=, >=), followed by a second *expression*:

expression operator expression

The *operator* can be =, <, >, <=, or >=.

RESTRICT Statement Expressions

A restriction expression in a RESTRICT statement is composed of parameter names, constants, and the operators times (*), plus (+), and minus (–). Each restriction expression must be a linear function of the parameters in the model. In addition, no grouping symbols (such as parentheses) are allowed, and the constant factor in any product can appear only on the left-hand side of the times (*) operator.

In the following example, the assumption is that you have a data table in which *y* is the loss variable and *x1*–*x3* are continuous variables. The PROC SEVSELECT program uses a RESTRICT statement to impose a restriction on the estimate for the parameter that is associated with the variable *x2*. Thus, in any solution that the optimizer finds, the solution must satisfy the condition that the parameter that is associated with the variable *x2* is equal to 1.5:

```
proc sevselect data=mylib.exrestrict;
  loss y;
  scalemodel x1-x3;
  dist stweedie;
  restrict x2=1.5;
run;
```

It is important to keep in mind that the parameters that are associated with the variables, not the variables themselves, are restricted. Thus, in the preceding RESTRICT statement, the variable name “x2” refers to the parameter that is associated with the variable x2, not to the variable itself.

Parameter names are shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as “*”, “-”, “(”, or “)”), then you must use the internal name of the parameter in order to refer to that parameter in the RESTRICT statement. You can specify the [PRINTINTERNALNAMES](#) option in the [PROC SEVSELECT](#) statement if you want to see the internal names of the parameters. When you specify the [PRINTINTERNALNAMES](#) option, an extra column is added to the “Parameter Estimates” table to show the internal name of each parameter.

Restrictions should be consistent and not redundant. All restriction equations in all RESTRICT statements are applied jointly.

RESTRICT Statement Examples

Examples of valid RESTRICT statements include the following:

```
restrict x1=0.1;
restrict a+b=1;
restrict a-b=0, b+c=1.5;
restrict 2*f=g+h, f+g=0;
```

Examples of invalid RESTRICT statements include the following:

```
restrict x1^2=4;
restrict x1*x3=4;
restrict x1/x3=2;
restrict sin(a)=0;
restrict a*0.5=1;
restrict 2*(f+h)=1;
```

In the first four examples of invalid RESTRICT statements, the equation is nonlinear. The fifth example is invalid because the constant factor (0.5) cannot appear on the right-hand side of the times (*) operator. The last example is invalid because grouping symbols are not allowed.

The set of restrictions must be consistent. For example, you cannot specify the following statement because the three restrictions are not consistent:

```
restrict f-g=0,
       f-h=0,
       g-h=1;
```

Lagrange multipliers are reported in the “Parameter Estimates” table for all the active linear constraints along with a test of the null hypothesis that the restriction is satisfied. They are identified by the names Restrict1, Restrict2, and so on. The test statistic is the estimate divided by its standard error, and it is asymptotically

normally distributed. This distribution is used to compute the p -value for the test. Nonactive (nonbinding) restrictions have no effect on the estimation results and are not included in the output.

SCALEMODEL Statement

SCALEMODEL *regression-effect-list* < / *scalemodel-options* > ;

The SCALEMODEL statement specifies regression effects. A regression effect is formed from one or more regressor variables according to effect construction rules. Each regression effect forms one element of $\mathbf{X}$ in the linear model structure $\mathbf{X}\boldsymbol{\beta}$ that affects the scale parameter of the distribution. The SCALEMODEL statement in conjunction with the CLASS statement supports a rich set of effects. Effects are specified by a special notation that uses regressor variable names and operators. There are two types of regressor variables: classification (or CLASS) variables and continuous variables. Classification variables can be either numeric or character and are specified in a CLASS statement. To include CLASS variables in regression effects, you must specify the CLASS statement so that it appears before the SCALEMODEL statement. A regressor variable that is not declared in the CLASS statement is assumed to be continuous. For information about effect construction rules, see the section “[Specification and Parameterization of Model Effects](#)” on page 91 in Chapter 4, “[Shared Concepts](#).”

All the regressor variables must be present in the input data table that you specify by using the DATA= option in the PROC SEVSELECT statement. The scale parameter of each candidate distribution is linked to the linear predictor $\mathbf{X}\boldsymbol{\beta}$ that includes an intercept. If a distribution does not have a scale parameter, then a model based on that distribution is not estimated. If you specify more than one SCALEMODEL statement, then the first statement is used.

The regressor variables are expected to have nonmissing values. If any of the variables has a missing value in an observation, then a warning is written to the SAS log and that observation is ignored.

For more information about modeling regression effects, see the section “[Estimating Regression Effects](#)” on page 1794.

You can specify the following *scalemodel-options* in the SCALEMODEL statement:

DFMIXTURE=*method-name* < (*method-options*) >

specifies the method for computing representative estimates of the cumulative distribution function (CDF) and the probability density function (PDF).

When you specify regression effects, the scale of the distribution depends on the values of the regressors. For a particular distribution family, each observation in the input data table implies a different scaled version of the distribution. To compute estimates of CDF and PDF that are comparable across different distribution families, PROC SEVSELECT needs to construct a single representative distribution from all such distributions. You can specify one of the following *method-name* values to specify the method that is used to construct the representative distribution. For more information about each of the methods, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

FULL

specifies that the representative distribution be the mixture of N distributions such that each distribution has a scale value that is implied by each of the N observations that are used for estimation. This method is the slowest.

MEAN

specifies that the representative distribution be the one-point mixture of the distribution whose scale value is computed by using the mean of the N values of the linear predictor that are implied by the N observations that are used for estimation. If you do not specify the `DFMIXTURE=` option, then this method is used by default. This is also the fastest method.

QUANTILE <(K=q)>

specifies that the representative distribution be the mixture of a fixed number of distributions whose scale values are computed by using the quantiles from the sample of N values of the linear predictor that are implied by the N observations that are used for estimation.

You can use the `K=` option to specify the number of distributions in the mixture. If you specify `K=q`, then the mixture contains $(q - 1)$ distributions such that each distribution has as its scale one of the $(q - 1)$ -quantiles.

If you do not specify the `K=` option, then PROC SEVSELECT uses the default of 2, which implies the use of a one-point mixture with a distribution whose scale value is the median of all scale values.

RANDOM <(random-method-options)>

specifies that the representative distribution be the mixture of a fixed number of distributions whose scale values are computed by using the values of the linear predictor that are implied by a randomly chosen subset of the set of all observations that are used for estimation. The same subset of observations is used for each distribution family.

You can specify the following *random-method-options* to specify how the subset is chosen:

K=r

specifies the number of distributions to include in the mixture. If you do not specify this option, then PROC SEVSELECT uses the default of 15.

SEED=number

specifies the seed that is used to generate the uniform random sample of observation indices. If you do not specify this option, then PROC SEVSELECT generates a seed internally that is based on the current value of the system clock.

INCLUDE=n**INCLUDE=single-effect****INCLUDE=(effects)**

forces effects to be included in all models. If you specify `INCLUDE=n`, then the first n effects that are listed in the `SCALEMODEL` statement are included in all models. If you specify `INCLUDE=single-effect` or `INCLUDE=(single-effect)`, then the specified effects are forced into all models. The `INCLUDE=` option has no effect if you do not specify the `SELECTION` statement.

INFORMATIVE

models missing values by using extra model effects. These effects consist of dummy variables that take the value 1 when the value of a continuous model variable involved in the effect is missing, and take the value 0 otherwise. The missing value in the original model effect is replaced by the average value of the effect for the nonmissing values. For continuous-by-class effects, such as $A*x$, where A is a classification variable and x is a continuous variable, informative missingness creates multiple dummy columns and substitutes the effect mean of x that corresponds to the respective level of A . Missing

values for classification variables are treated as valid levels. For more information about informative missingness, see the section “[Informative Missingness](#)” on page 106 in Chapter 4, “[Shared Concepts](#).”

OFFSET=*offset-variable-name*

specifies the name of the offset variable in the scale regression model. An offset variable is a regressor variable whose regression coefficient is known to be 1. For more information, see the section “[Offset Variable](#)” on page 1795.

START=*n*

START=*single-effect*

START=(*effects*)

specifies how to begin the selection process in the FORWARD, FORWARDSWAP, and STEPWISE selection methods from the initial model that you designate. If you specify **START=***n*, then the starting model consists of the first *n* effects that are listed in the [SCALEMODEL](#) statement. If you specify **START=***single-effect* or **START=**(*single-effect*), then the starting model consists of these specified effects.

The **START=** option has no effect if you do not specify the [SELECTION](#) statement. This option is not supported when you specify **METHOD=BACKWARD** in the [SELECTION](#) statement.

SELECTION Statement

SELECTION < **METHOD=***method* < (*method-options*) > > < *options* > ;

The **SELECTION** statement performs model selection by examining whether effects should be added to or removed from the model according to rules that are defined by the selection methods. The statement is fully documented in the section “[SELECTION Statement](#)” on page 78 in Chapter 4, “[Shared Concepts](#).” The SEVSELECT procedure supports only a subset of the *method* values and *method-options* that are listed in that section.

NOTE: In the context of the **SELECTION** statement, the term *model* refers to the set of regression effects in the scale regression model. The final severity model consists of the set of distribution parameters and the set of selected regression effects.

You can use the following option to specify the selection method:

METHOD=BACKWARD | FORWARD | FORWARDSWAP | STEPWISE

specifies the selection method to use. You can specify one of the following methods:

BACKWARD	specifies the backward elimination method.
FORWARD	specifies the forward selection method.
FORWARDSWAP	specifies forward-swap selection, which is an extension of the forward selection method.
STEPWISE	specifies the stepwise regression method.

By default, **METHOD=STEPWISE**. For more information about each of the selection methods, see the section “[Model Selection Methods](#)” on page 102

The SEVSELECT procedure supports a specific set of values for the following *method-options*:

CHOOSE=criterion

chooses from the list of models (at each step of the selection process) the model that yields the best value of the specified criterion. The SEVSELECT procedure supports only the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- AICC** specifies the corrected Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, the CHOOSE= criterion is the same as the SELECT= criterion. For more information, see the detailed description of the [CHOOSE=](#) option.

SELECT=criterion

specifies the criterion that the procedure uses to determine the order in which effects enter or leave at each step of the selection method. The SEVSELECT procedure supports the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- AICC** specifies the corrected Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, SELECT=SBC. For more information, see the detailed description of the [SELECT=](#) option.

STOP=criterion

specifies a criterion that is used to stop the selection process. The SEVSELECT procedure supports only the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- AICC** specifies the corrected Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, the STOP= criterion is the same as the SELECT= criterion. For more information, see the detailed description of the [STOP=](#) option.

The SEVSELECT procedure supports the following *method-options* as they are described in the section “[SELECTION Statement](#)” on page 78 in Chapter 4, “[Shared Concepts](#)”:

- [COMPETITIVE](#)
- [MAXEFFECTS=](#)
- [MAXSTEPS=](#)
- [MINEFFECTS=](#)

The SEVSELECT procedure supports the following statement *options* as they are described in the section “[SELECTION Statement](#)” on page 78 in Chapter 4, “[Shared Concepts](#)”:

- DETAILS=
- HIERARCHY=
- ORDERSELECT
- STOPHORIZON=

WEIGHT Statement

WEIGHT *variable-name* ;

The WEIGHT statement specifies the name of a variable whose values represent the weight of each observation. PROC SEVSELECT associates a weight of w to each observation, where w is the value of the WEIGHT variable for the observation. If the weight value is missing or less than or equal to 0, then the observation is ignored and a warning is written to the SAS log. When you do not specify the WEIGHT statement, each observation is assigned a weight of 1. If you specify more than one WEIGHT statement, then the last statement is used.

The weights are normalized so that they add up to the actual sample size. In particular, the weight of each observation is multiplied by $\frac{N}{\sum_{i=1}^N w_i}$, where N is the sample size. All computations, including the computations of the EDF-based statistics of fit, use normalized weights.

Programming Statements

You can use a series of programming statements that use variables in the input data table that you specify in the DATA= option in the PROC SEVSELECT statement to assign a value to an objective function symbol. You must specify the objective function symbol by using the **OBJECTIVE=** option in the PROC SEVSELECT statement. If you do not specify the **OBJECTIVE=** option in the PROC SEVSELECT statement, then the programming statements are ignored and models are estimated using the maximum likelihood method.

You can use most DATA step statements and functions in your program. Any additional functions, restrictions, and differences are listed in the section “[Custom Objective Functions](#)” on page 1836.

Details: SEVSELECT Procedure

Predefined Distributions

For the response variable Y , PROC SEVSELECT assumes the model

$$Y \sim \mathcal{F}(\Theta)$$

where $\mathcal{F}$ is a continuous probability distribution with parameters Θ . The model hypothesizes that the observed response is generated from a stochastic process that is governed by the distribution $\mathcal{F}$. This model is usually referred to as the error model. Given a representative input sample of response variable values, PROC SEVSELECT estimates the model parameters for any distribution $\mathcal{F}$ and computes the statistics of fit for each model. This enables you to find the distribution that is most likely to generate the observed sample.

A set of predefined distributions is provided with the SEVSELECT procedure. A summary of the distributions is provided in [Table 26.4](#). For each distribution, the table lists the name of the distribution that should be used in the DIST statement, the parameters of the distribution along with their bounds, and the mathematical expressions for the probability density function (PDF) and cumulative distribution function (CDF) of the distribution.

All the predefined distributions, except LOGN and TWEEDIE, are parameterized such that their first parameter is the scale parameter. For LOGN, the first parameter μ is a log-transformed scale parameter. TWEEDIE does not have a scale parameter. The presence of scale parameter or a log-transformed scale parameter enables you to use all of the predefined distributions, except TWEEDIE, as a candidate for estimating regression effects.

A distribution model is associated with each predefined distribution. You can also define your own distribution model, which is a set of functions and subroutines that you define by using the FCMP procedure. For more information, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811.

Table 26.4 Predefined PROC SEVSELECT Distributions

Name	Distribution	Parameters	PDF (f) and CDF (F)
BURR	Burr	$\theta > 0, \alpha > 0,$ $\gamma > 0$	$f(x) = \frac{\alpha \gamma z^\gamma}{x(1+z^\gamma)^{(\alpha+1)}}$ $F(x) = 1 - \left(\frac{1}{1+z^\gamma}\right)^\alpha$
EXP	Exponential	$\theta > 0$	$f(x) = \frac{1}{\theta} e^{-z}$ $F(x) = 1 - e^{-z}$
GAMMA	Gamma	$\theta > 0, \alpha > 0$	$f(x) = \frac{z^\alpha e^{-z}}{x \Gamma(\alpha)}$ $F(x) = \frac{\gamma(\alpha, z)}{\Gamma(\alpha)}$
GPD	Generalized Pareto	$\theta > 0, \xi > 0$	$f(x) = \frac{1}{\theta} (1 + \xi z)^{-1-1/\xi}$ $F(x) = 1 - (1 + \xi z)^{-1/\xi}$
IGAUSS	Inverse Gaussian (Wald)	$\theta > 0, \alpha > 0$	$f(x) = \frac{1}{\theta} \sqrt{\frac{\alpha}{2\pi z^3}} e^{\frac{-\alpha(z-1)^2}{2z}}$ $F(x) = \Phi\left((z-1)\sqrt{\frac{\alpha}{z}}\right) + \Phi\left(-(z+1)\sqrt{\frac{\alpha}{z}}\right) e^{2\alpha}$
LOGN	Lognormal	μ (no bounds), $\sigma > 0$	$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$ $F(x) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right)$
PARETO	Pareto	$\theta > 0, \alpha > 0$	$f(x) = \frac{\alpha \theta^\alpha}{(x+\theta)^{\alpha+1}}$ $F(x) = 1 - \left(\frac{\theta}{x+\theta}\right)^\alpha$
TWEEDIE	Tweedie**	$p > 1, \mu > 0,$ $\phi > 0$	$f(x) = a(x, \phi) \exp\left[\frac{1}{\phi} \left(\frac{x\mu^{1-p}}{1-p} - \kappa(\mu, p)\right)\right]$ $F(x) = \int_0^x f(t) dt$
STWEEDIE	Scaled Tweedie**	$\theta > 0, \lambda > 0,$ $1 < p < 2$	$f(x) = a(x, \theta, \lambda, p) \exp\left(-\frac{x}{\theta} - \lambda\right)$ $F(x) = \int_0^x f(t) dt$
WEIBULL	Weibull	$\theta > 0, \tau > 0$	$f(x) = \frac{\tau}{\theta} z^{\tau-1} e^{-z^\tau}$ $F(x) = 1 - e^{-z^\tau}$

**For more information, see the section “[Tweedie Distributions](#)” on page 1781.

Notes:

1. $z = x/\theta$, wherever z is used.
2. θ denotes the scale parameter for all the distributions. For LOGN, $\log(\theta) = \mu$.
3. Parameters are listed in the order in which they are defined in the distribution model.
4. $\gamma(a, b) = \int_0^b t^{a-1} e^{-t} dt$ is the lower incomplete gamma function.
5. $\Phi(y) = \frac{1}{2} \left(1 + \operatorname{erf}\left(\frac{y}{\sqrt{2}}\right)\right)$ is the standard normal CDF.

Tweedie Distributions

Tweedie distributions are a special case of the exponential dispersion family (Jørgensen 1987) with a property that the variance of the distribution is equal to $\phi\mu^p$, where μ is the mean of the distribution, ϕ is a dispersion parameter, and p is an index parameter as discovered by Tweedie (1984). The distribution is defined for all values of p except for values of p in the open interval $(0, 1)$. Many important known distributions are a special case of Tweedie distributions including normal ($p=0$), Poisson ($p=1$), gamma ($p=2$), and the inverse Gaussian ($p=3$). Apart from these special cases, the probability density function (PDF) of the Tweedie distribution does not have an analytic expression. For $p > 1$, it has the form (Dunn and Smyth 2005),

$$f(x; \mu, \phi, p) = a(x, \phi) \exp \left[\frac{1}{\phi} \left(\frac{x\mu^{1-p}}{1-p} - \kappa(\mu, p) \right) \right]$$

where $\kappa(\mu, p) = \mu^{2-p}/(2-p)$ for $p \neq 2$ and $\kappa(\mu, p) = \log(\mu)$ for $p = 2$. The function $a(x, \phi)$ does not have an analytical expression. It is typically evaluated using series expansion methods described in Dunn and Smyth (2005).

For $1 < p < 2$, the Tweedie distribution is a compound Poisson-gamma mixture distribution, which is the distribution of S defined as

$$S = \sum_{i=1}^N X_i$$

where $N \sim \text{Poisson}(\lambda)$ and $X_i \sim \text{gamma}(\alpha, \theta)$ are independent and identically distributed gamma random variables with shape parameter α and scale parameter θ . At $X = 0$, the density is a probability mass that is governed by the Poisson distribution, and for values of $X > 0$, it is a mixture of gamma variates with Poisson mixing probability. The parameters λ , α , and θ are related to the natural parameters μ , ϕ , and p of the Tweedie distribution as

$$\begin{aligned}\lambda &= \frac{\mu^{2-p}}{\phi(2-p)} \\ \alpha &= \frac{2-p}{p-1} \\ \theta &= \phi(p-1)\mu^{p-1}\end{aligned}$$

The mean of a Tweedie distribution is positive for $p > 1$.

Two predefined versions of the Tweedie distribution are provided with the SEVSELECT procedure. The first version, named TWEEDIE and defined for $p > 1$, has the natural parameterization with parameters μ , ϕ , and p . The second version, named STWEEDIE and defined for $1 < p < 2$, is the version with a scale parameter. It corresponds to the compound Poisson-gamma distribution with gamma scale parameter θ , Poisson mean parameter λ , and the index parameter p . The index parameter decides the shape parameter α of the gamma distribution as

$$\alpha = \frac{2-p}{p-1}$$

The parameters θ and λ of the STWEEDIE distribution are related to the parameters μ and ϕ of the TWEEDIE distribution as

$$\begin{aligned}\mu &= \lambda\theta\alpha \\ \phi &= \frac{(\lambda\theta\alpha)^{2-p}}{\lambda(2-p)} = \frac{\theta}{(p-1)(\lambda\theta\alpha)^{p-1}}\end{aligned}$$

You can fit either version when there are no regression variables. Each version has its own merits. If you fit the TWEEDIE version, you have the direct estimate of the overall mean of the distribution. If you are interested in the most practical range of the index parameter $1 < p < 2$, then you can fit the STWEEDIE version, which provides you direct estimates of the Poisson and gamma components that comprise the distribution (an estimate of the gamma shape parameter α is easily obtained from the estimate of p).

If you want to estimate the effect of exogenous (regression) variables on the distribution, then you must use the STWEEDIE version, because PROC SEVSELECT requires a distribution to have a scale parameter in order to estimate regression effects. For more information, see the section “[Estimating Regression Effects](#)” on page 1794. The gamma scale parameter θ is the scale parameter of the STWEEDIE distribution. If you are interested in determining the effect of regression variables on the mean of the distribution, you can do so by first fitting the STWEEDIE distribution to determine the effect of the regression variables on the scale parameter θ . Then, you can easily estimate how the mean of the distribution μ is affected by the regression variables using the relationship $\mu = c\theta$, where $c = \lambda\alpha = \lambda(2 - p)/(p - 1)$. The estimates of the regression parameters remain the same, whereas the estimate of the intercept parameter is adjusted by the estimates of the λ and p parameters.

Parameter Initialization for Predefined Distributions

The parameters are initialized by using the method of moments for all the distributions, except for the gamma and the Weibull distributions. For the gamma distribution, approximate maximum likelihood estimates are used. For the Weibull distribution, the method of percentile matching is used.

Given n observations of the severity value y_i ($1 \leq i \leq n$), the estimate of k th raw moment is denoted by m_k and computed as

$$m_k = \frac{1}{n} \sum_{i=1}^n y_i^k$$

The 100 p th percentile is denoted by π_p ($0 \leq p \leq 1$). By definition, π_p satisfies

$$F(\pi_p-) \leq p \leq F(\pi_p)$$

where $F(\pi_p-) = \lim_{h \downarrow 0} F(\pi_p - h)$. PROC SEVSELECT uses the following practical method of computing π_p . Let $\hat{F}_n(y)$ denote the empirical distribution function (EDF) estimate at a severity value y . Let y_p^- and y_p^+ denote two consecutive values in the ascending sequence of y values such that $\hat{F}_n(y_p^-) < p$ and $\hat{F}_n(y_p^+) \geq p$. Then, the estimate $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = y_p^- + \frac{p - \hat{F}_n(y_p^-)}{\hat{F}_n(y_p^+) - \hat{F}_n(y_p^-)}(y_p^+ - y_p^-)$$

Let ϵ denote the smallest double-precision floating-point number such that $1 + \epsilon > 1$. This machine precision constant can be obtained by using the CONSTANT function in Base SAS software.

The details of how parameters are initialized for each predefined distribution are as follows:

BURR Burr proposed 12 types of families of continuous distributions (Burr 1942; Rodriguez and Balan 2006). The predefined BURR distribution in PROC SEVSELECT implements Burr's

type XII distribution. The parameters are initialized by using the method of moments. The k th raw moment of the Burr distribution of type XII is

$$E[X^k] = \frac{\theta^k \Gamma(1 + k/\gamma) \Gamma(\alpha - k/\gamma)}{\Gamma(\alpha)}, \quad -\gamma < k < \alpha\gamma$$

Three moment equations $E[X^k] = m_k$ ($k = 1, 2, 3$) need to be solved for initializing the three parameters of the distribution. In order to get an approximate closed form solution, the second shape parameter $\hat{\gamma}$ is initialized to a value of 2. If $2m_3 - 3m_1m_2 > 0$, then simplifying and solving the moment equations yields the following feasible set of initial values:

$$\hat{\theta} = \sqrt{\frac{m_2m_3}{2m_3 - 3m_1m_2}}, \quad \hat{\alpha} = 1 + \frac{m_3}{2m_3 - 3m_1m_2}, \quad \hat{\gamma} = 2$$

If $2m_3 - 3m_1m_2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = \sqrt{m_2}, \quad \hat{\alpha} = 2, \quad \hat{\gamma} = 2$$

EXP The parameters are initialized by using the method of moments. The k th raw moment of the exponential distribution is

$$E[X^k] = \theta^k \Gamma(k + 1), \quad k > -1$$

Solving $E[X] = m_1$ yields the initial value of $\hat{\theta} = m_1$.

GAMMA The parameter α is initialized by using its *approximate* maximum likelihood (ML) estimate. For a set of n independent and identically distributed observations y_i ($1 \leq i \leq n$) drawn from a gamma distribution, the log likelihood l is defined as follows:

$$\begin{aligned} l &= \sum_{i=1}^n \log \left(y_i^{\alpha-1} \frac{e^{-y_i/\theta}}{\theta^\alpha \Gamma(\alpha)} \right) \\ &= (\alpha - 1) \sum_{i=1}^n \log(y_i) - \frac{1}{\theta} \sum_{i=1}^n y_i - n\alpha \log(\theta) - n \log(\Gamma(\alpha)) \end{aligned}$$

Using a shorter notation of $\sum$ to denote $\sum_{i=1}^n$ and solving the equation $\partial l / \partial \theta = 0$ yields the following ML estimate of θ :

$$\hat{\theta} = \frac{\sum y_i}{n\alpha} = \frac{m_1}{\alpha}$$

Substituting this estimate in the expression of l and simplifying gives

$$l = (\alpha - 1) \sum \log(y_i) - n\alpha - n\alpha \log(m_1) + n\alpha \log(\alpha) - n \log(\Gamma(\alpha))$$

Let d be defined as follows:

$$d = \log(m_1) - \frac{1}{n} \sum \log(y_i)$$

Solving the equation $\partial l / \partial \alpha = 0$ yields the following expression in terms of the digamma function, $\psi(\alpha)$:

$$\log(\alpha) - \psi(\alpha) = d$$

The digamma function can be approximated as follows:

$$\hat{\psi}(\alpha) \approx \log(\alpha) - \frac{1}{\alpha} \left(0.5 + \frac{1}{12\alpha + 2} \right)$$

This approximation is within 1.4% of the true value for all the values of $\alpha > 0$ except when α is arbitrarily close to the positive root of the digamma function (which is approximately 1.461632). Even for the values of α that are close to the positive root, the absolute error between true and approximate values is still acceptable ($|\hat{\psi}(\alpha) - \psi(\alpha)| < 0.005$ for $\alpha > 1.07$). Solving the equation that arises from this approximation yields the following estimate of α :

$$\hat{\alpha} = \frac{3 - d + \sqrt{(d - 3)^2 + 24d}}{12d}$$

If this approximate ML estimate is infeasible, then the method of moments is used. The k th raw moment of the gamma distribution is

$$E[X^k] = \theta^k \frac{\Gamma(\alpha + k)}{\Gamma(\alpha)}, \quad k > -\alpha$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial value for α :

$$\hat{\alpha} = \frac{m_1^2}{m_2 - m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance), then α is initialized as follows:

$$\hat{\alpha} = 1$$

After computing the estimate of α , the estimate of θ is computed as follows:

$$\hat{\theta} = \frac{m_1}{\hat{\alpha}}$$

Both the maximum likelihood method and the method of moments arrive at the same relationship between $\hat{\alpha}$ and $\hat{\theta}$.

GPD

The parameters are initialized by using the method of moments. Notice that for $\xi > 0$, the CDF of the generalized Pareto distribution (GPD) is:

$$\begin{aligned} F(x) &= 1 - \left(1 + \frac{\xi x}{\theta} \right)^{-1/\xi} \\ &= 1 - \left(\frac{\theta/\xi}{x + \theta/\xi} \right)^{1/\xi} \end{aligned}$$

This is equivalent to a Pareto distribution with scale parameter $\theta_1 = \theta/\xi$ and shape parameter $\alpha = 1/\xi$. Using this relationship, the parameter initialization method used for the PARETO distribution is used to get the following initial values for the parameters of the GPD distribution:

$$\hat{\theta} = \frac{m_1 m_2}{2(m_2 - m_1^2)}, \quad \hat{\xi} = \frac{m_2 - 2m_1^2}{2(m_2 - m_1^2)}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance) or $m_2 - 2m_1^2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = \frac{m_1}{2}, \quad \hat{\xi} = \frac{1}{2}$$

IGAUSS

The parameters are initialized by using the method of moments. The standard parameterization of the inverse Gaussian distribution (also known as the Wald distribution), in terms of the location parameter μ and shape parameter λ , is as follows (Klugman, Panjer, and Willmot 1998, p. 583):

$$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x - \mu)^2}{2\mu^2 x}\right)$$

$$F(x) = \Phi\left(\left(\frac{x}{\mu} - 1\right) \sqrt{\frac{\lambda}{x}}\right) + \Phi\left(-\left(\frac{x}{\mu} + 1\right) \sqrt{\frac{\lambda}{x}}\right) \exp\left(\frac{2\lambda}{\mu}\right)$$

For this parameterization, it is known that the mean is $E[X] = \mu$ and the variance is $\text{Var}[X] = \mu^3/\lambda$, which yields the second raw moment as $E[X^2] = \mu^2(1 + \mu/\lambda)$ (computed by using $E[X^2] = \text{Var}[X] + (E[X])^2$).

The predefined IGAUSS distribution in PROC SEVSELECT uses the following alternate parameterization to allow the distribution to have a scale parameter, θ :

$$f(x) = \sqrt{\frac{\alpha\theta}{2\pi x^3}} \exp\left(\frac{-\alpha(x - \theta)^2}{2x\theta}\right)$$

$$F(x) = \Phi\left(\left(\frac{x}{\theta} - 1\right) \sqrt{\frac{\alpha\theta}{x}}\right) + \Phi\left(-\left(\frac{x}{\theta} + 1\right) \sqrt{\frac{\alpha\theta}{x}}\right) \exp(2\alpha)$$

The parameters θ (scale) and α (shape) of this alternate form are related to the parameters μ and λ of the preceding form such that $\theta = \mu$ and $\alpha = \lambda/\mu$. Using this relationship, the first and second raw moments of the IGAUSS distribution are

$$E[X] = \theta$$

$$E[X^2] = \theta^2 \left(1 + \frac{1}{\alpha}\right)$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = \frac{m_1^2}{m_2 - m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance), then the parameters are initialized as follows:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = 1$$

LOGN

The parameters are initialized by using the method of moments. The k th raw moment of the lognormal distribution is

$$E[X^k] = \exp\left(k\mu + \frac{k^2\sigma^2}{2}\right)$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\mu} = 2 \log(m_1) - \frac{\log(m_2)}{2}, \quad \hat{\sigma} = \sqrt{\log(m_2) - 2 \log(m_1)}$$

PARETO The parameters are initialized by using the method of moments. The k th raw moment of the Pareto distribution is

$$E[X^k] = \frac{\theta^k \Gamma(k+1) \Gamma(\alpha-k)}{\Gamma(\alpha)}, -1 < k < \alpha$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\theta} = \frac{m_1 m_2}{m_2 - 2m_1^2}, \quad \hat{\alpha} = \frac{2(m_2 - m_1^2)}{m_2 - 2m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance) or $m_2 - 2m_1^2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = 2$$

TWEEDIE The parameter p is initialized by assuming that the sample is generated from a gamma distribution with shape parameter α and by computing $\hat{p} = \frac{\hat{\alpha}+2}{\hat{\alpha}+1}$. The initial value $\hat{\alpha}$ is obtained from using the method previously described for the GAMMA distribution. The parameter μ is the mean of the distribution. Hence, it is initialized to the sample mean as

$$\hat{\mu} = m_1$$

Variance of a Tweedie distribution is equal to $\phi \mu^p$. Thus, the sample variance is used to initialize the value of ϕ as

$$\hat{\phi} = \frac{m_2 - m_1^2}{\hat{\mu}^{\hat{p}}}$$

STWEEDIE STWEEDIE is a compound Poisson-gamma mixture distribution with mean $\mu = \lambda \theta \alpha$, where α is the shape parameter of the gamma random variables in the mixture and the parameter p is determined solely by α . First, the parameter p is initialized by assuming that the sample is generated from a gamma distribution with shape parameter α and by computing $\hat{p} = \frac{\hat{\alpha}+2}{\hat{\alpha}+1}$. The initial value $\hat{\alpha}$ is obtained from using the method previously described for the GAMMA distribution. As done for initializing the parameters of the TWEEDIE distribution, the sample mean and variance are used to compute the values $\hat{\mu}$ and $\hat{\phi}$ as

$$\hat{\mu} = m_1$$

$$\hat{\phi} = \frac{m_2 - m_1^2}{\hat{\mu}^{\hat{p}}}$$

Based on the relationship between the parameters of TWEEDIE and STWEEDIE distributions described in the section “[Tweedie Distributions](#)” on page 1781, values of θ and λ are initialized as

$$\hat{\theta} = \hat{\phi}(\hat{p} - 1)\hat{\mu}^{p-1}$$

$$\hat{\lambda} = \frac{\hat{\mu}}{\hat{\theta}\hat{\alpha}}$$

WEIBULL The parameters are initialized by using the percentile matching method. Let $q1$ and $q3$ denote the estimates of the 25th and 75th percentiles, respectively. Using the formula for the CDF of Weibull distribution, they can be written as

$$1 - \exp(-(q1/\theta)^\tau) = 0.25$$

$$1 - \exp(-(q3/\theta)^\tau) = 0.75$$

Simplifying and solving these two equations yields the following initial values,

$$\hat{\theta} = \exp\left(\frac{r \log(q1) - \log(q3)}{r - 1}\right), \quad \hat{\tau} = \frac{\log(\log(4))}{\log(q3) - \log(\hat{\theta})}$$

where $r = \log(\log(4)) / \log(\log(4/3))$. These initial values agree with those suggested in Klugman, Panjer, and Willmot (1998).

A summary of the initial values of all the parameters for all the predefined distributions is given in [Table 26.5](#). The table also provides the names of the parameters to use in the **INIT=** option in the DIST statement if you want to provide a different initial value.

Table 26.5 Parameter Initialization for Predefined Distributions

Distribution	Parameter	Name for INIT Option	Default Initial Value
BURR	θ	theta	$\sqrt{\frac{m_2 m_3}{2m_3 - 3m_1 m_2}}$
	α	alpha	$1 + \frac{m_3}{2m_3 - 3m_1 m_2}$
	γ	gamma	2
EXP	θ	theta	m_1
GAMMA	θ	theta	m_1/α
	α	alpha	$\frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
GPD	θ	theta	$m_1 m_2 / (2(m_2 - m_1^2))$
	ξ	xi	$(m_2 - 2m_1^2) / (2(m_2 - m_1^2))$
IGAUSS	θ	theta	m_1
	α	alpha	$m_1^2 / (m_2 - m_1^2)$
LOGN	μ	mu	$2 \log(m_1) - \log(m_2)/2$
	σ	sigma	$\sqrt{\log(m_2) - 2 \log(m_1)}$
PARETO	θ	theta	$m_1 m_2 / (m_2 - 2m_1^2)$
	α	alpha	$2(m_2 - m_1^2) / (m_2 - 2m_1^2)$
TWEEDIE	μ	mu	m_1
	ϕ	phi	$(m_2 - m_1^2) / m_1^p$
	p	p	$(\alpha + 2) / (\alpha + 1)$
			where $\alpha = \frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
STWEEDIE	θ	theta	$(m_2 - m_1^2)(p - 1) / m_1$
	λ	lambda	$m_1^2 / (\alpha(m_2 - m_1^2)(p - 1))$
	p	p	$(\alpha + 2) / (\alpha + 1)$
			where $\alpha = \frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
WEIBULL	θ	theta	$\exp\left(\frac{r \log(q1) - \log(q3)}{r-1}\right)$
	τ	tau	$\log(\log(4)) / (\log(q3) - \log(\hat{\theta}))$

Notes:

1. m_k denotes the k th raw moment.
2. $d = \log(m_1) - (\sum \log(y_i))/n$
3. $q1$ and $q3$ denote the 25th and 75th percentiles, respectively.
4. $r = \log(\log(4)) / \log(\log(4/3))$

Censoring and Truncation

One of the key features of PROC SEVSELECT is that it enables you to specify whether the severity event's magnitude is observable and if it is observable, then whether the exact value of the magnitude is known. If an event is unobservable when the magnitude is in certain intervals, then it is referred to as a truncation effect. If the exact magnitude of the event is not known, but it is known to have a value in a certain interval, then it is referred to as a censoring effect.

PROC SEVSELECT allows a severity event to be subject to any combination of the following four censoring and truncation effects:

- **Left-truncation:** An event is said to be left-truncated if it is observed only when $Y > T^l$, where Y denotes the random variable for the magnitude and T^l denotes a random variable for the truncation threshold. You can specify left-truncation using the `LEFTTRUNCATED=` option in the LOSS statement.
- **Right-truncation:** An event is said to be right-truncated if it is observed only when $Y \leq T^r$, where Y denotes the random variable for the magnitude and T^r denotes a random variable for the truncation threshold. You can specify right-truncation using the `RIGHTTRUNCATED=` option in the LOSS statement.
- **Left-censoring:** An event is said to be left-censored if it is known that the magnitude is $Y \leq C^l$, but the exact value of Y is not known. C^l is a random variable for the censoring limit. You can specify left-censoring using the `LEFTCENSORED=` option in the LOSS statement.
- **Right-censoring:** An event is said to be right-censored if it is known that the magnitude is $Y > C^r$, but the exact value of Y is not known. C^r is a random variable for the censoring limit. You can specify right-censoring using the `RIGHTCENSORED=` option in the LOSS statement.

For each effect, you can specify a different threshold or limit for each observation or specify a single threshold or limit that applies to all the observations.

If all four types of effects are present on an event, then the following relationship holds: $T^l < C^r \leq C^l \leq T^r$. PROC SEVSELECT checks these relationships and writes a warning to the SAS log if any relationship is violated.

If you specify the response variable in the LOSS statement, then PROC SEVSELECT also checks whether each observation satisfies the definitions of the specified censoring and truncation effects. If you specify left-truncation, then PROC SEVSELECT ignores observations where $Y \leq T^l$, because such observations are not observable by definition. Similarly, if you specify right-truncation, then PROC SEVSELECT ignores observations where $Y > T^r$. If you specify left-censoring, then PROC SEVSELECT treats an observation with $Y > C^l$ as uncensored and ignores the value of C^l . The observations with $Y \leq C^l$ are considered as left-censored, and the value of Y is ignored. If you specify right-censoring, then PROC SEVSELECT treats an observation with $Y \leq C^r$ as uncensored and ignores the value of C^r . The observations with $Y > C^r$ are considered as right-censored, and the value of Y is ignored. If you specify both left-censoring and right-censoring, it is referred to as interval-censoring. If $C^r < C^l$ is satisfied for an observation, then it is considered as interval-censored and the value of the response variable is ignored. If $C^r = C^l$ for an observation, then PROC SEVSELECT assumes that observation to be uncensored. If all the observations in a data table are censored in some form, then the specification of the response variable in the LOSS statement is

optional, because the actual value of the response variable is not required for the purposes of estimating a model.

Specification of censoring and truncation affects the likelihood of the data (see the section “[Likelihood Function](#)” on page 1791) and how the empirical distribution function (EDF) is estimated (see the section “[Empirical Distribution Function Estimation Methods](#)” on page 1799).

Probability of Observability

For left-truncated data, PROC SEVSELECT also enables you to provide additional information in the form of *probability of observability* by using the `PROBOBSERVED=` option. It is defined as the probability that the underlying severity event gets observed (and recorded) for the specified left-truncation threshold value. For example, if you specify a value of 0.75, then for every 75 observations recorded above a specified threshold, 25 more events have happened with a severity value less than or equal to the specified threshold. Although the exact severity value of those 25 events is not known, PROC SEVSELECT can use the information about the number of those events.

In particular, for each left-truncated observation, PROC SEVSELECT assumes a presence of $(1 - p)/p$ additional observations with $y_i = t_i$. These additional observations are then used for computing the likelihood (see the section “[Probability of Observability and Likelihood](#)” on page 1792) and an unconditional estimate of the empirical distribution function (see the section “[EDF Estimates and Truncation](#)” on page 1804).

Truncation and Conditional CDF Estimates

If you specify left-truncation without the probability of observability or if you specify right-truncation, then the EDF estimates that are computed by all methods except the STANDARD method are conditional on the truncation information. For more information, see the section “[EDF Estimates and Truncation](#)” on page 1804. In such cases, PROC SEVSELECT uses conditional estimates of the CDF for computational or visual comparison to the EDF estimates.

Let $t_{\min}^l = \min_i \{t_i^l\}$ be the smallest value of the left-truncation threshold (t_i^l is the left-truncation threshold for observation i) and $t_{\max}^r = \max_i \{t_i^r\}$ be the largest value of the right-truncation threshold (t_i^r is the right-truncation threshold for observation i). If $\hat{F}(y)$ denotes the unconditional estimate of the CDF at y , then the conditional estimate $\hat{F}^c(y)$ is computed as follows:

- If you do not specify the probability of observability, then the EDF estimates are conditional on the left-truncation information. If an observation is both left-truncated and right-truncated, then

$$\hat{F}^c(y) = \frac{\hat{F}(y) - \hat{F}(t_{\min}^l)}{\hat{F}(t_{\max}^r) - \hat{F}(t_{\min}^l)}$$

If an observation is left-truncated but not right-truncated, then

$$\hat{F}^c(y) = \frac{\hat{F}(y) - \hat{F}(t_{\min}^l)}{1 - \hat{F}(t_{\min}^l)}$$

If an observation is right-truncated but not left-truncated, then

$$\hat{F}^c(y) = \frac{\hat{F}(y)}{\hat{F}(t_{\max}^r)}$$

- If you specify the probability of observability, then EDF estimates are not conditional on the left-truncation information. If an observation is not right-truncated, then the conditional estimate is the same as the unconditional estimate. If an observation is right-truncated, then the conditional estimate is computed as

$$\hat{F}^c(y) = \frac{\hat{F}(y)}{\hat{F}(t_{\max}^r)}$$

If you specify regression effects, then $\hat{F}(y)$, $\hat{F}(t_{\min}^l)$, and $\hat{F}(t_{\max}^r)$ are all computed from a mixture distribution, as described in the section “CDF and PDF Estimates with Regression Effects” on page 1797.

Parameter Estimation Method

PROC SEVSELECT uses the maximum likelihood (ML) method to estimate the parameters of each model. A nonlinear optimization process is used to maximize the log of the likelihood function.

Likelihood Function

Let $f_{\Theta}(x)$ and $F_{\Theta}(x)$ denote the PDF and CDF, respectively, evaluated at x for a set of parameter values Θ . Let Y denote the random response variable, and let y denote its value recorded in an observation in the input data table. Let T^l and T^r denote the random variables for the left-truncation and right-truncation threshold, respectively, and let t^l and t^r denote their values for an observation, respectively. If there is no left-truncation, then $t^l = \tau^l$, where τ^l is the smallest value in the support of the distribution; so $F(t^l) = 0$. If there is no right-truncation, then $t^r = \tau_h$, where τ_h is the largest value in the support of the distribution; so $F(t^r) = 1$. Let C^l and C^r denote the random variables for the left-censoring and right-censoring limit, respectively, and let c^l and c^r denote their values for an observation, respectively. If there is no left-censoring, then $c^l = \tau_h$; so $F(c^l) = 1$. If there is no right-censoring, then $c^r = \tau^l$; so $F(c^r) = 0$.

The set of input observations can be categorized into the following four subsets within each BY group:

- E is the set of uncensored and untruncated observations. The likelihood of an observation in E is

$$l_E = \Pr(Y = y) = f_{\Theta}(y)$$

- E_t is the set of uncensored observations that are truncated. The likelihood of an observation in E_t is

$$l_{E_t} = \Pr(Y = y | t^l < Y \leq t^r) = \frac{f_{\Theta}(y)}{F_{\Theta}(t^r) - F_{\Theta}(t^l)}$$

- C is the set of censored observations that are not truncated. The likelihood of an observation C is

$$l_C = \Pr(c^r < Y \leq c^l) = F_{\Theta}(c^l) - F_{\Theta}(c^r)$$

- C_t is the set of censored observations that are truncated. The likelihood of an observation C_t is

$$l_{C_t} = \Pr(c^r < Y \leq c^l | t^l < Y \leq t^r) = \frac{F_{\Theta}(c^l) - F_{\Theta}(c^r)}{F_{\Theta}(t^r) - F_{\Theta}(t^l)}$$

Note that $(E \cup E_t) \cap (C \cup C_t) = \emptyset$. Also, the sets E_t and C_t are empty when you do not specify truncation, and the sets C and C_t are empty when you do not specify censoring.

Given this, the likelihood of the data L is as follows:

$$L = \prod_E f_{\Theta}(y) \prod_{E_t} \frac{f_{\Theta}(y)}{F_{\Theta}(t^r) - F_{\Theta}(t^l)} \prod_C F_{\Theta}(c^l) - F_{\Theta}(c^r) \prod_{C_t} \frac{F_{\Theta}(c^l) - F_{\Theta}(c^r)}{F_{\Theta}(t^r) - F_{\Theta}(t^l)}$$

The maximum likelihood procedure used by PROC SEVSELECT finds an optimal set of parameter values $\hat{\Theta}$ that maximizes $\log(L)$ subject to the boundary constraints on parameter values. For a distribution *dist*, you can specify such boundary constraints by using the *dist_LOWERBOUNDS* and *dist_UPPERBOUNDS* subroutines. For more information, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811. Some aspects of the optimization process can be controlled by using the *NLOPTIONS* statement.

Probability of Observability and Likelihood

If you specify the probability of observability for the left-truncation, then PROC SEVSELECT uses a modified likelihood function for each truncated observation. If the probability of observability is $p \in (0.0, 1.0]$, then for each left-truncated observation with truncation threshold t^l , there exist $(1-p)/p$ observations with a response variable value less than or equal to t^l . Each such observation has a probability of $\Pr(Y \leq t^l) = F_{\Theta}(t^l)$. The right-truncation and censoring information does not apply to these added observations. Thus, following the notation of the section “[Likelihood Function](#)” on page 1791, the likelihood of the data is as follows:

$$L = \prod_E f_{\Theta}(y) \prod_{E_t, t^l = \tau^l} \frac{f_{\Theta}(y)}{F_{\Theta}(t^r)} \prod_{E_t, t^l > \tau^l} \frac{f_{\Theta}(y)}{F_{\Theta}(t^r)} F_{\Theta}(t^l)^{\frac{1-p}{p}} \\ \prod_C F_{\Theta}(c^l) - F_{\Theta}(c^r) \prod_{C_t, t^l = \tau^l} \frac{F_{\Theta}(c^l) - F_{\Theta}(c^r)}{F_{\Theta}(t^r)} \prod_{C_t, t^l > \tau^l} \frac{F_{\Theta}(c^l) - F_{\Theta}(c^r)}{F_{\Theta}(t^r)} F_{\Theta}(t^l)^{\frac{1-p}{p}}$$

Note that the likelihood of the observations that are not left-truncated (observations in sets E and C , and observations in sets E_t and C_t for which $t^l = \tau^l$) is not affected.

If you specify a custom objective function, then PROC SEVSELECT accounts for the probability of observability only while computing the empirical distribution function estimate. The parameter estimates are affected only by your custom objective function.

Estimating Covariance and Standard Errors

PROC SEVSELECT computes an estimate of the covariance matrix of the parameters by using the asymptotic theory of the maximum likelihood estimators (MLE). If N denotes the number of observations used for estimating a parameter vector θ , then the theory states that as $N \rightarrow \infty$, the distribution of $\hat{\theta}$, the estimate of θ , converges to a normal distribution with mean θ and covariance $\hat{C}$ such that $\mathbf{I}(\theta) \cdot \hat{C} \rightarrow 1$, where $\mathbf{I}(\theta) = -E[\nabla^2 \log(L(\theta))]$ is the information matrix for the likelihood of the data, $L(\theta)$. The covariance estimate is obtained by using the inverse of the information matrix.

In particular, if $\mathbf{G} = \nabla^2(-\log(L(\boldsymbol{\theta})))$ denotes the Hessian matrix of the negative of log likelihood, then the covariance estimate is computed as

$$\hat{\mathbf{C}} = \frac{N}{d} \mathbf{G}^{-1}$$

where d is a denominator that is determined by the **VARDEF=** option. If **VARDEF=N**, then $d = N$, which yields the asymptotic covariance estimate. If **VARDEF=DF**, then $d = N - k$, where k is number of parameters (the model's degrees of freedom). The **VARDEF=DF** option is the default, because it attempts to correct the potential bias introduced by the finite sample.

The standard error s_i of the parameter θ_i is computed as the square root of the i th diagonal element of the estimated covariance matrix; that is, $s_i = \sqrt{\hat{C}_{ii}}$.

If you specify a custom objective function, then the covariance matrix of the parameters is still computed by inverting the information matrix, except that the Hessian matrix $\mathbf{G}$ is computed as $\mathbf{G} = \nabla^2 \log(U(\boldsymbol{\theta}))$, where U denotes your custom objective function that is minimized by the optimizer.

Covariance and standard error estimates might not be available if the Hessian matrix is found to be singular at the end of the optimization process. This can especially happen if the optimization process stops without converging.

Parameter Initialization

PROC SEVSELECT enables you to initialize parameters of a model in different ways. A model can have two kinds of parameters: distribution parameters and regression parameters.

The distribution parameters can be initialized by using one of the following three methods:

INIT= option	You can use the INIT= option in the DIST statement.
INEST= option	You can use the INEST= data table in the PROC SEVSELECT statement.
PARMINIT subroutine	You can define a <i>dist_PARMINIT</i> subroutine in the distribution model. For more information, see the section “ Defining a Severity Distribution Model with the FCMP Procedure ” on page 1811.

Note that only one of the initialization methods is used. You cannot combine them. They are used in the following order:

- The method that uses the **INIT=** option takes the highest precedence. If you use the **INIT=** option to provide an initial value for at least one parameter, then other initialization methods (**INEST=** or **PARMINIT**) are not used. If you specify initial values for some but not all the parameters by using the **INIT=** option, then the uninitialized parameters are initialized to the default value of 0.001.

If you use this option and if you specify the regression effects, then the value of the first distribution parameter must be related to the initial value for the *base* value of the scale or log-transformed scale parameter. For more information, see the section “[Estimating Regression Effects](#)” on page 1794.

- The method that uses the **INEST=** data table takes second precedence. If the **INEST=** data table contains a nonmissing value for even one distribution parameter, then the **PARMINIT** method is not used and any uninitialized parameters are initialized to the default value of 0.001.

- If none of the distribution parameters are initialized by using the INIT= option or the INEST= data table, but the distribution model defines a PARMINIT subroutine, then PROC SEVSELECT invokes that subroutine with appropriate inputs to initialize the parameters. If the PARMINIT subroutine returns missing values for some parameters, then those parameters are initialized to the default value of 0.001.
- If none of the initialization methods are used, each distribution parameter is initialized to the default value of 0.001.

For more information about regression models and initialization of regression parameters, see the section “[Estimating Regression Effects](#)” on page 1794.

PARMINIT-Based Parameter Initialization Method and Distributed Data

If the input data are distributed across the computational nodes and if you use the PARMINIT subroutine to initialize the distribution parameters, then PROC SEVSELECT invokes that subroutine on each computational node with the data that are local to that node. The EDF estimates that are supplied to the PARMINIT subroutine are also computed using the local data. The initial values of the parameters that are supplied to the optimizer are the average of the local estimates that are computed on each node. This approach works well if the data are distributed randomly across nodes.

Estimating Regression Effects

The SEVSELECT procedure enables you to estimate the influence of regression (exogenous) effects while fitting a distribution if the distribution has a scale parameter or a log-transformed scale parameter.

Let x_j , $j = 1, \dots, k$, denote the k regression effects. Let β_j denote the regression parameter that corresponds to the effect x_j . If you do not specify regression effects, then the model for the response variable Y is of the form

$$Y \sim \mathcal{F}(\Theta)$$

where $\mathcal{F}$ is the distribution of Y with parameters Θ . This model is usually referred to as the error model. The regression effects are modeled by extending the error model to the following form:

$$Y \sim \exp\left(\sum_{j=1}^k \beta_j x_j\right) \cdot \mathcal{F}(\Theta)$$

Under this model, the distribution of Y is valid and belongs to the same parametric family as $\mathcal{F}$ if and only if $\mathcal{F}$ has a scale parameter. Let θ denote the scale parameter and Ω denote the set of nonscale distribution parameters of $\mathcal{F}$. Then the model can be rewritten as

$$Y \sim \mathcal{F}(\theta, \Omega)$$

such that θ is modeled by the regression effects as

$$\theta = \theta_0 \cdot \exp\left(\sum_{j=1}^k \beta_j x_j\right)$$

where θ_0 is the *base* value of the scale parameter. Thus, the scale regression model consists of the following parameters: θ_0 , Ω , and β_j ($j = 1, \dots, k$).

Given this form of the model, distributions without a scale parameter cannot be considered when regression effects are to be modeled. If a distribution does not have a direct scale parameter, then PROC SEVSELECT accepts it only if it has a log-transformed scale parameter—that is, if it has a parameter $p = \log(\theta)$.

Offset Variable

You can specify that an offset variable be included in the scale regression model by specifying it in the **OFFSET=** option of the SCALEMODEL statement. The offset variable is a regressor whose regression coefficient is known to be 1. If x_o denotes the offset variable, then the scale regression model becomes

$$\theta = \theta_0 \cdot \exp(x_o + \sum_{j=1}^k \beta_j x_j)$$

The regression coefficient of the offset variable is fixed at 1 and not estimated, so it is not reported in the ParameterEstimates ODS table. However, if you specify the OUTEST= data table, then the regression coefficient is added as a variable to that data table. The value of the offset variable in OUTEST= data table is equal to 1 for the estimates row (_TYPE_='EST') and is equal to the special missing value .F for the standard error (_TYPE_='STDERR') and covariance (_TYPE_='COV') rows.

An offset variable is useful to model the scale parameter per unit of some measure of exposure. For example, in the automobile insurance context, measure of exposure can be the number of car-years insured or the total number of miles driven by a fleet of cars at a rental car company. For worker's compensation insurance, if you want to model the expected loss per enterprise, then you can use the number of employees or total employee salary as the measure of exposure. For epidemiological data, measure of exposure can be the number of people who are exposed to a certain pathogen when you are modeling the loss associated with an epidemic. In general, if e denotes the value of the exposure measure and if you specify $x_o = \log(e)$ as the offset variable, then you are modeling the influence of other regression effects (x_j) on the size of the scale of the distribution *per unit of exposure*.

Another use for an offset variable is when you have a priori knowledge of the influence of some exogenous variables that cannot be included in the SCALEMODEL statement. You can model the combined influence of such variables as an offset variable in order to correct for the omitted variable bias.

Parameter Initialization for Regression Models

The regression parameters are initialized either by using the values that you specify or by the default method.

- If you provide initial values for the regression parameters, then you must provide valid, nonmissing initial values for θ_0 and β_j parameters for all j .

You can specify the initial value for θ_0 by using either the INEST= data table or the INIT= option in the DIST statement. If the distribution has a direct scale parameter (no transformation), then the initial value for the first parameter of the distribution is used as an initial value for θ_0 . If the distribution has a log-transformed scale parameter, then the initial value for the first parameter of the distribution is used as an initial value for $\log(\theta_0)$.

If you use the INEST= data table to specify the initial values for β_j , then the INEST= data table must be an unmodified version of a OUTEST= data table that you have created in a previous PROC SEVSELECT step with the same set of regression effects.

- If you do not specify valid initial values for θ_0 or β_j parameters for all j , then PROC SEVSELECT initializes those parameters by using the following method:

Let a random variable Y be distributed as $\mathcal{F}(\theta, \Omega)$, where θ is the scale parameter. By the definition of the scale parameter, a random variable $W = Y/\theta$ is distributed as $\mathcal{G}(\Omega)$ such that $\mathcal{G}(\Omega) = \mathcal{F}(1, \Omega)$. Given a random error term e that is generated from a distribution $\mathcal{G}(\Omega)$, a value y from the distribution of Y can be generated as

$$y = \theta \cdot e$$

Taking the logarithm of both sides and using the relationship of θ with the regression effects yields

$$\log(y) = \log(\theta_0) + \sum_{j=1}^k \beta_j x_j + \log(e)$$

PROC SEVSELECT makes use of the preceding relationship to initialize parameters of a regression model with distribution *dist* as follows:

1. The following linear regression problem is solved to obtain initial estimates of β_0 and β_j :

$$\log(y) = \beta_0 + \sum_{j=1}^k \beta_j x_j$$

The estimates of $\beta_j (j = 1, \dots, k)$ in the solution of this regression problem are used to initialize the respective regression parameters of the model. The estimate of β_0 is later used to initialize the value of θ_0 .

The results of this regression are also used to detect whether any regression parameters are linearly dependent on the other regression parameters. If any such parameters are found, then a warning is written to the SAS log and the corresponding parameter is eliminated from further analysis. The estimates for linearly dependent regression parameters are denoted by the special missing value .R in the OUTEST= data table and in any displayed output.

2. Let s_0 denote the initial value of the scale parameter.

If the distribution model of *dist* does not contain the *dist_PARMINIT* subroutine, then s_0 and all the nonscale distribution parameters are initialized to the default value of 0.001.

However, it is strongly recommended that each distribution's model contain the *dist_PARMINIT* subroutine. For more information, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811. If that subroutine is defined, then s_0 is initialized as follows: Each input value y_i of the response variable is transformed to its scale-normalized version w_i as

$$w_i = \frac{y_i}{\exp(\beta_0 + \sum_{j=1}^k \beta_j x_{ij})}$$

where x_{ij} denotes the value of j th regression effect in the i th input observation. These w_i values are used to compute the input arguments for the *dist_PARMINIT* subroutine. The values that are computed by the subroutine for nonscale parameters are used as their respective initial values. If the distribution has an untransformed scale parameter, then s_0 is set to the value of the scale parameter that is computed by the subroutine. If the distribution has a log-transformed scale parameter P , then s_0 is computed as $s_0 = \exp(l_0)$, where l_0 is the value of P computed by the subroutine.

3. The value of θ_0 is initialized as

$$\theta_0 = s_0 \cdot \exp(\beta_0)$$

Reporting Estimates of Regression Parameters

When you request estimates to be written to the output (either ODS displayed output or in the OUTEST= data table), the estimate of the base value of the first distribution parameter is reported. If the first parameter is the log-transformed scale parameter, then the estimate of $\log(\theta_0)$ is reported; otherwise, the estimate of θ_0 is reported. The transform of the first parameter of a distribution *dist* is controlled by the *dist_SCALETRANSFORM* function that is defined for it.

CDF and PDF Estimates with Regression Effects

When regression effects are estimated, the estimate of the scale parameter depends on the values of the regressors and the estimates of the regression parameters. This dependency results in a potentially different distribution for each observation. To make estimates of the cumulative distribution function (CDF) and probability density function (PDF) comparable across distributions and comparable to the empirical distribution function (EDF), PROC SEVSELECT computes and reports the CDF and PDF estimates from a representative distribution. The *representative distribution* is a mixture of a certain number of distributions, where each distribution differs only in the value of the scale parameter. You can specify the number of distributions in the mixture and how their scale values are chosen by using the *DFMIXTURE*= option in the SCALEMODEL statement.

Let N denote the number of observations that are used for estimation, K denote the number of components in the mixture distribution, s_k denote the scale parameter of the k th mixture component, and d_k denote the weight associated with k th mixture component.

Let $f(y; s_k, \hat{\Omega})$ and $F(y; s_k, \hat{\Omega})$ denote the PDF and CDF, respectively, of the k th component distribution, where $\hat{\Omega}$ denotes the set of estimates of all parameters of the distribution other than the scale parameter. Then, the PDF and CDF estimates, $f^*(y)$ and $F^*(y)$, respectively, of the mixture distribution at y are computed as

$$f^*(y) = \frac{1}{D} \sum_{k=1}^K d_k f(y; s_k, \hat{\Omega})$$

$$F^*(y) = \frac{1}{D} \sum_{k=1}^K d_k F(y; s_k, \hat{\Omega})$$

where D is the normalization factor ($D = \sum_{k=1}^K d_k$).

PROC SEVSELECT uses the $F^*(y)$ values to compute the EDF-based statistics of fit.

The scale values s_k for the K mixture components are derived from the set $\{\hat{\lambda}_i\}$ ($i = 1, \dots, N$) of N linear predictor values, where $\hat{\lambda}_i$ denotes the estimate of the linear predictor due to observation i . It is computed as

$$\hat{\lambda}_i = \log(\hat{\theta}_0) + \sum_{j=1}^k \hat{\beta}_j x_{ij}$$

where $\hat{\theta}_0$ is an estimate of the base value of the scale parameter, $\hat{\beta}_j$ are the estimates of regression coefficients, and x_{ij} is the value of j th regression effect in observation i .

Let w_i denote the weight of observation i . If you specify the WEIGHT statement, then the weight is equal to the value of the specified weight variable for the corresponding observation in the DATA= data table; otherwise, the weight is set to 1.

You can specify one of the following *method-names* in the DFMIXTURE= option in the SCALEMODEL statement to specify the method of choosing K and the corresponding s_k and d_k values:

FULL In this method, there are as many mixture components as the number of observations that are used for estimation. In other words, $K = N$, $s_k = \hat{\theta}_k$, and $d_k = w_k$ ($k = 1, \dots, N$). This is the slowest method, because it requires $O(N)$ computations to compute the mixture CDF $F^*(y_i)$ or the mixture PDF $f^*(y_i)$ of one observation. For N observations, the computational complexity in terms of number of CDF or PDF evaluations is $O(N^2)$. Even for moderately large values of N , the time that is taken to compute the mixture CDF and PDF can significantly exceed the time that is taken to estimate the model parameters. So it is recommended that you use the FULL method only for small data tables.

MEAN In this method, the mixture contains only one distribution, whose scale value is determined by the mean of the linear predictor values that are implied by all the observations. In other words, s_1 is computed as

$$s_1 = \exp \left(\frac{1}{N} \sum_{i=1}^N \hat{\lambda}_i \right)$$

The component's weight d_1 is set to 1.

This method is the fastest because it requires only one CDF or PDF evaluation per observation. The computational complexity is $O(N)$ for N observations.

If you do not specify the DFMIXTURE= option in the SCALEMODEL statement, then this is the default method.

QUANTILE In this method, a certain number of quantiles are chosen from the set of all linear predictor values. If you specify a value of q for the K= option when specifying this method, then $K = q - 1$ and s_k ($k = 1, \dots, K$) is computed as $s_k = \exp(\hat{\lambda}_k)$, where $\hat{\lambda}_k$ is the k th q -quantile from the set $\{\hat{\lambda}_i\}$ ($i = 1, \dots, N$). The weight of each of the components (d_k) is assumed to be 1 for this method.

The default value of q is 2, which implies a one-point mixture that has a distribution whose scale value is equal to the median scale value.

For this method, PROC SEVSELECT needs to sort the N linear predictor values in the set $\{\hat{\lambda}_i\}$; the sorting requires $O(N \log(N))$ computations. Then, computing the mixture estimate of one observation requires $(q - 1)$ CDF or PDF evaluations. Hence, the computational complexity of this method is $O(qN) + O(N \log(N))$ for computing a mixture CDF or PDF of N observations. For $q \ll N$, the QUANTILE method is significantly faster than the FULL method.

RANDOM In this method, a uniform random sample of observations is chosen, and the mixture contains the distributions that are implied by those observations. If you specify a value of r for the K= option when specifying this method, then the size of the sample is r . Hence, $K = r$. If l_j denotes the index of j th observation in the sample ($j = 1, \dots, r$), such that $1 \leq l_j \leq N$, then the scale of k th component distribution in the mixture is $s_k = \exp(\hat{\lambda}_{l_k})$. The weight of each of the components (d_k) is assumed to be 1 for this method.

You can also specify the seed to be used for generating the random sample by using the SEED= option for this method. The same sample of observations is used for all models.

Computing a mixture estimate of one observation requires r CDF or PDF evaluations. Hence, the computational complexity of this method is $O(rN)$ for computing a mixture CDF or PDF of N observations. For $r \ll N$, the RANDOM method is significantly faster than the FULL method.

Empirical Distribution Function Estimation Methods

The empirical distribution function (EDF) is a nonparametric estimate of the cumulative distribution function (CDF) of the distribution. PROC SEVSELECT computes EDF estimates for two purposes: to send the estimates to a distribution's [PARMINIT](#) subroutine in order to initialize the distribution parameters, and to compute the EDF-based statistics of fit.

To reduce the time that it takes to compute the EDF estimates, you can use the [INITSAMPLE](#) option to specify that only a fraction of the input data be used. If you do not specify the INITSAMPLE option and the data table has more than 10,000 valid observations, then a uniform random sample of at most 10,000 observations is used for EDF estimation.

In the distributed mode of execution, in which data are distributed across the grid nodes, the EDF estimates are computed on each node by using the portion of the input data that is located on that node. These local EDF estimates are an approximation of the global EDF estimates, which would have been computed by using the entire input data table. PROC SEVSELECT does not compute global EDF estimates. Let X denote a quantity that depends on the EDF estimates. X can be either an EDF-based initial value of a distribution parameter or an EDF-based statistic of fit. PROC SEVSELECT estimates X as follows: First, each grid node k computes an estimate X_k by using the local EDF estimates that are computed on that node. Then, the estimate $\hat{X}$ of X is computed as an average of all the X_k values; that is, $\hat{X} = \sum_{i=1}^K X_k$, where K denotes the total number of nodes where the data reside.

This section describes the methods that are used for computing EDF estimates.

Notation

Let there be a set of N observations, each containing a quintuplet of values $(y_i, t_i^l, t_i^r, c_i^r, c_i^l)$, $i = 1, \dots, N$, where y_i is the value of the response variable, t_i^l is the value of the left-truncation threshold, t_i^r is the value of the right-truncation threshold, c_i^r is the value of the right-censoring limit, and c_i^l is the value of the left-censoring limit.

If an observation is not left-truncated, then $t_i^l = \tau^l$, where τ^l is the smallest value in the support of the distribution; so $F(t_i^l) = 0$. If an observation is not right-truncated, then $t_i^r = \tau_h$, where τ_h is the largest value in the support of the distribution; so $F(t_i^r) = 1$. If an observation is not right-censored, then $c_i^r = \tau^l$; so $F(c_i^r) = 0$. If an observation is not left-censored, then $c_i^l = \tau_h$; so $F(c_i^l) = 1$.

Let w_i denote the weight associated with i th observation. If you specify the [WEIGHT](#) statement, then w_i is the normalized value of the weight variable; otherwise, it is set to 1. The weights are normalized such that they sum up to N .

An indicator function $I[e]$ takes a value of 1 or 0 if the expression e is true or false, respectively.

Estimation Methods

If the response variable is subject to both left-censoring and right-censoring effects and if you explicitly specify the `EMPIRICALCDF=TURNBULL` option, then PROC SEVSELECT uses the Turnbull's method. This section describes methods other than Turnbull's method. For Turnbull's method, see the next section "Turnbull's EDF Estimation Method" on page 1802.

The method descriptions assume that all observations are either uncensored or right-censored; that is, each observation is of the form $(y_i, t_i^l, t_i^r, \tau^l, \tau_h)$ or $(y_i, t_i^l, t_i^r, c_i^r, \tau_h)$.

If all observations are either uncensored or left-censored, then each observation is of the form $(y_i, t_i^l, t_i^r, \tau_l, c_i^l)$. It is converted to an observation $(-y_i, -t_i^r, -t_i^l, -c_i^l, \tau_h)$; that is, the signs of all the response variable values are reversed, the new left-truncation threshold is equal to the negative of the original right-truncation threshold, the new right-truncation threshold is equal to the negative of the original left-truncation threshold, and the negative of the original left-censoring limit becomes the new right-censoring limit. With this transformation, each observation is either uncensored or right-censored. The methods described for handling uncensored or right-censored data are now applicable. After the EDF estimates are computed, the observations are transformed back to the original form and EDF estimates are adjusted such $F_n(y_i) = 1 - F_n(-y_i-)$, where $F_n(-y_i-)$ denotes the EDF estimate of the value slightly less than the transformed value $-y_i$.

Further, a set of uncensored or right-censored observations can be converted to a set of observations of the form $(y_i, t_i^l, t_i^r, \delta_i)$, where δ_i is the indicator of right-censoring. $\delta_i = 0$ indicates a right-censored observation, in which case y_i is assumed to record the right-censoring limit c_i^r . $\delta_i = 1$ indicates an uncensored observation, and y_i records the exact observed value. In other words, $\delta_i = I[Y \leq C^r]$ and $y_i = \min(y_i, c_i^r)$.

Given this notation, the EDF is estimated as

$$F_n(y) = \begin{cases} 0 & \text{if } y < y^{(1)} \\ \hat{F}_n(y^{(k)}) & \text{if } y^{(k)} \leq y < y^{(k+1)}, k = 1, \dots, N-1 \\ \hat{F}_n(y^{(N)}) & \text{if } y^{(N)} \leq y \end{cases}$$

where $y^{(k)}$ denotes the k th-order statistic of the set $\{y_i\}$ and $\hat{F}_n(y^{(k)})$ is the estimate computed at that value. The definition of $\hat{F}_n$ depends on the estimation method. You can specify a particular method or let PROC SEVSELECT choose an appropriate method by using the `EMPIRICALCDF=` option in the PROC SEVSELECT statement. Each method computes $\hat{F}_n$ as follows:

NOTURNBULL This is the default method. First, censored observations, if any, are processed as follows:

- An observation that is left-censored but not right-censored is converted to an uncensored observation $(y_i^u, t_i^l, t_i^r, \tau^l, \tau_h)$, where $y_i^u = c_i^l/2$.
- An observation that is both left-censored and right-censored is converted to an uncensored observation $(y_i^u, t_i^l, t_i^r, \tau^l, \tau_h)$, where $y_i^u = (c_i^r + c_i^l)/2$.
- An observation that is right-censored but not left-censored is left unchanged.

If the processed set of observations contains any truncated or right-censored observations, the KAPLANMEIER method is used. Otherwise, the STANDARD method is used.

The observations are modified only for the purpose of computing the EDF estimates. The original censoring information is used by the parameter estimation process.

STANDARD

This method is the standard way of computing EDF. The EDF estimate at observation i is computed as follows:

$$\hat{F}_n(y_i) = \frac{1}{N} \sum_{j=1}^N w_j \cdot I[y_j \leq y_i]$$

If you do not specify any censoring or truncation information, then this method is chosen. If you explicitly specify this method, then PROC SEVSELECT ignores any censoring and truncation information that you specify in the LOSS statement.

The standard error of $\hat{F}_n(y_i)$ is computed by using the normal approximation method:

$$\hat{\sigma}_n(y_i) = \sqrt{\hat{F}_n(y_i)(1 - \hat{F}_n(y_i))/N}$$

KAPLANMEIER

The Kaplan-Meier (KM) estimator, also known as the product-limit estimator, was first introduced by Kaplan and Meier (1958) for censored data. Lynden-Bell (1971) derived a similar estimator for left-truncated data. PROC SEVSELECT uses the definition that combines both censoring and truncation information (Klein and Moeschberger 1997; Lai and Ying 1991).

The EDF estimate at observation i is computed as

$$\hat{F}_n(y_i) = 1 - \prod_{\tau \leq y_i} \left(1 - \frac{n(\tau)}{R_n(\tau)}\right)$$

where $n(\tau)$ and $R_n(\tau)$ are defined as follows:

- $n(\tau) = \sum_{k=1}^N w_k \cdot I[y_k = \tau \text{ and } \tau \leq t_k^r \text{ and } \delta_k = 1]$, which is the number of uncensored observations ($\delta_k = 1$) for which the response variable value is equal to τ and τ is observable according to the right-truncation threshold of that observation ($\tau \leq t_k^r$).
- $R_n(\tau) = \sum_{k=1}^N w_k \cdot I[y_k \geq \tau > t_k^l]$, which is the size (cardinality) of the risk set at τ . The term *risk set* has its origins in survival analysis; it contains the events that are at risk of failure at a particular time, τ . In other words, it contains the events that have survived up to time τ and might fail at or after τ . For PROC SEVSELECT, *time* is equivalent to the magnitude of the event and *failure* is equivalent to an uncensored and observable event, where observable means it satisfies the truncation thresholds.

This method is chosen when you specify at least one form of censoring or truncation.

The standard error of $\hat{F}_n(y_i)$ is computed by using Greenwood's formula (Greenwood 1926):

$$\hat{\sigma}_n(y_i) = \sqrt{(1 - \hat{F}_n(y_i))^2 \cdot \sum_{\tau \leq y_i} \left(\frac{n(\tau)}{R_n(\tau)(R_n(\tau) - n(\tau))} \right)}$$

MODIFIEDKM

The product-limit estimator used by the KAPLANMEIER method does not work well if the risk set size becomes very small. For right-censored data, the size can become

small towards the right tail. For left-truncated data, the size can become small at the left tail and can remain so for the entire range of data. This was demonstrated by Lai and Ying (1991). They proposed a modification to the estimator that ignores the effects due to small risk set sizes.

The EDF estimate at observation i is computed as

$$\hat{F}_n(y_i) = 1 - \prod_{\tau \leq y_i} \left(1 - \frac{n(\tau)}{R_n(\tau)} \cdot I[R_n(\tau) \geq cN^\alpha] \right)$$

where the definitions of $n(\tau)$ and $R_n(\tau)$ are identical to those used for the KAPLAN-MEIER method described previously.

You can specify the values of c and α by using the **C=** and **ALPHA=** options. If you do not specify a value for c , the default value used is $c = 1$. If you do not specify a value for α , the default value used is $\alpha = 0.5$.

As an alternative, you can also specify an absolute lower bound, say L , on the risk set size by using the **RSLB=** option, in which case $I[R_n(\tau) \geq cN^\alpha]$ is replaced by $I[R_n(\tau) \geq L]$ in the definition.

The standard error of $\hat{F}_n(y_i)$ is computed by using Greenwood's formula (Greenwood 1926):

$$\hat{\sigma}_n(y_i) = \sqrt{(1 - \hat{F}_n(y_i))^2 \cdot \sum_{\tau \leq y_i} \left(\frac{n(\tau)}{R_n(\tau)(R_n(\tau) - n(\tau))} \cdot I[R_n(\tau) \geq cN^\alpha] \right)}$$

Turnbull's EDF Estimation Method

If the response variable is subject to both left-censoring and right-censoring effects and if you explicitly specify the **EMPIRICALCDF=TURNBULL** option, then the SEVSELECT procedure uses a method proposed by Turnbull (1976) to compute the nonparametric estimates of the cumulative distribution function. The original Turnbull's method is modified using the suggestions made by Frydman (1994) when truncation effects are present.

Let the input data consist of N observations in the form of quintuplets of values $(y_i, t_i^l, t_i^r, c_i^r, c_i^l)$, $i = 1, \dots, N$ with notation described in the section “**Notation**” on page 1799. For each observation, let $A_i = (c_i^r, c_i^l]$ be the censoring interval; that is, the response variable value is known to lie in the interval A_i , but the exact value is not known. If an observation is uncensored, then $A_i = (y_i - \epsilon, y_i]$ for any arbitrarily small value of $\epsilon > 0$. If an observation is censored, then the value y_i is ignored. Similarly, for each observation, let $B_i = (t_i^l, t_i^r]$ be the truncation interval; that is, the observation is drawn from a truncated (conditional) distribution $F(y, B_i) = P(Y \leq y | Y \in B_i)$.

Two sets, L and R , are formed using A_i and B_i as follows:

$$L = \{c_i^r, 1 \leq i \leq N\} \cup \{t_i^r, 1 \leq i \leq N\}$$

$$R = \{c_i^l, 1 \leq i \leq N\} \cup \{t_i^l, 1 \leq i \leq N\}$$

The sets L and R represent the left endpoints and right endpoints, respectively. A set of disjoint intervals $C_j = [q_j, p_j]$, $1 \leq j \leq M$ is formed such that $q_j \in L$ and $p_j \in R$ and $q_j \leq p_j$ and $p_j < q_{j+1}$. The value of M is dependent on the nature of censoring and truncation intervals in the input data. Turnbull (1976)

showed that the maximum likelihood estimate (MLE) of the EDF can increase only inside intervals C_j . In other words, the MLE estimate is constant in the interval (p_j, q_{j+1}) . The likelihood is independent of the behavior of F_n inside any of the intervals C_j . Let s_j denote the increase in F_n inside an interval C_j . Then, the EDF estimate is as follows:

$$F_n(y) = \begin{cases} 0 & \text{if } y < q_1 \\ \sum_{k=1}^j s_k & \text{if } p_j < y < q_{j+1}, 1 \leq j \leq M-1 \\ 1 & \text{if } y > p_M \end{cases}$$

PROC SEVSELECT computes the estimates $F_n(p_j+) = F_n(q_{j+1}-) = \sum_{k=1}^j s_k$ at points p_j and q_{j+1} and computes $F_n(q_1-) = 0$ at point q_1 , where $F_n(x+)$ denotes the limiting estimate at a point that is infinitesimally larger than x when approaching x from values larger than x and where $F_n(x-)$ denotes the limiting estimate at a point that is infinitesimally smaller than x when approaching x from values smaller than x .

PROC SEVSELECT uses the expectation-maximization (EM) algorithm proposed by Turnbull (1976), who referred to the algorithm as the self-consistency algorithm. By default, the algorithm runs until one of the following criteria is met:

- Relative-error criterion: The maximum relative error between the two consecutive estimates of s_j falls below a threshold ϵ . If l indicates an index of the current iteration, then this can be formally stated as

$$\arg \max_{1 \leq j \leq M} \left\{ \frac{|s_j^l - s_j^{l-1}|}{s_j^{l-1}} \right\} \leq \epsilon$$

You can control the value of ϵ by specifying the **EPS=** suboption of the EDF=TURNBULL option in the PROC SEVSELECT statement. The default value is 1.0E-8.

- Maximum-iteration criterion: The number of iterations exceeds an upper limit that you specify for the **MAXITER=** suboption of the EDF=TURNBULL option in the PROC SEVSELECT statement. The default number of maximum iterations is 500.

The self-consistent estimates obtained in this manner might not be maximum likelihood estimates. Gentleman and Geyer (1994) suggested the use of the Kuhn-Tucker conditions for the maximum likelihood problem to ensure that the estimates are MLE. If you specify the **ENSUREMLE** suboption of the EDF=TURNBULL option in the PROC SEVSELECT statement, then PROC SEVSELECT computes the Kuhn-Tucker conditions at the end of each iteration to determine whether the estimates $\{s_j\}$ are MLE. If you do not specify any truncation effects, then the Kuhn-Tucker conditions derived by Gentleman and Geyer (1994) are used. If you specify any truncation effects, then PROC SEVSELECT uses modified Kuhn-Tucker conditions that account for the truncation effects. An integral part of checking the conditions is to determine whether an estimate s_j is zero or whether an estimate of the Lagrange multiplier or the reduced gradient associated with the estimate s_j is zero. PROC SEVSELECT declares these values to be zero if they are less than or equal to a threshold δ . You can control the value of δ by specifying the **ZEROPROB=** suboption of the EDF=TURNBULL option in the PROC SEVSELECT statement. The default value is 1.0E-8. The algorithm continues until the Kuhn-Tucker conditions are satisfied or the number of iterations exceeds the upper limit. The relative-error criterion stated previously is not used when you specify the ENSUREMLE option.

The standard errors for Turnbull's EDF estimates are computed by using the asymptotic theory of the maximum likelihood estimators (MLE), even though the final estimates might not be MLE. Turnbull's

estimator essentially attempts to maximize the likelihood L , which depends on the parameters s_j ($j = 1, \dots, M$). Let $\mathbf{s} = \{s_j\}$ denote the set of these parameters. If $\mathbf{G}(\mathbf{s}) = \nabla^2(-\log(L(\mathbf{s})))$ denotes the Hessian matrix of the negative of log likelihood, then the variance-covariance matrix of $\mathbf{s}$ is estimated as $\hat{\mathbf{C}}(\mathbf{s}) = \mathbf{G}^{-1}(\mathbf{s})$. Given this matrix, the standard error of $F_n(y)$ is computed as

$$\sigma_n(y) = \sqrt{\sum_{k=1}^j \left(\hat{C}_{kk} + 2 \cdot \sum_{l=1}^{k-1} \hat{C}_{kl} \right)}, \text{ if } p_j < y < q_{j+1}, 1 \leq j \leq M-1$$

The standard error is undefined outside of these intervals.

EDF Estimates and Truncation

If you specify truncation, then the estimate $\hat{F}_n(y)$ that is computed by any method other than the STANDARD method is a *conditional* estimate. In other words, $\hat{F}_n(y) = \Pr(Y \leq y | \tau_G < Y \leq \tau_H)$, where G and H denote the (unknown) distribution functions of the left-truncation threshold variable T^l and the right-truncation threshold variable T^r , respectively, τ_G denotes the smallest left-truncation threshold with a nonzero cumulative probability, and τ_H denotes the largest right-truncation threshold with a nonzero cumulative probability. Formally, $\tau_G = \inf\{s : G(s) > 0\}$ and $\tau_H = \sup\{s : H(s) > 0\}$. For computational purposes, PROC SEVSELECT estimates τ_G and τ_H by $t_{\min}^l$ and $t_{\max}^r$, respectively, defined as

$$t_{\min}^l = \min\{t_k^l : 1 \leq k \leq N\}$$

$$t_{\max}^r = \max\{t_k^r : 1 \leq k \leq N\}$$

These estimates of $t_{\min}^l$ and $t_{\max}^r$ are used to compute the conditional estimates of the CDF as described in the section “[Truncation and Conditional CDF Estimates](#)” on page 1790.

If you specify left-truncation *with* the probability of observability p , then PROC SEVSELECT uses the additional information provided by p to compute an estimate of the EDF that is not conditional on the left-truncation information. In particular, for each left-truncated observation i with response variable value y_i and truncation threshold t_i^l , an observation j is added with *weight* $w_j = (1 - p)/p$ and $y_j = t_j^l$. Each added observation is assumed to be uncensored and untruncated. Then, your specified EDF method is used by assuming no left-truncation. The EDF estimate that is obtained using this method is not conditional on the left-truncation information. For the KAPLANMEIER and MODIFIEDKM methods with uncensored or right-censored data, definitions of $n(\tau)$ and $R_n(\tau)$ are modified to account for the added observations. If N^a denotes the total number of observations including the added observations, then $n(\tau)$ is defined as $n(\tau) = \sum_{k=1}^{N^a} w_k I[y_k = \tau \text{ and } \tau \leq t_k^r \text{ and } \delta_k = 1]$, and $R_n(\tau)$ is defined as $R_n(\tau) = \sum_{k=1}^{N^a} w_k I[y_k \geq \tau]$. In the definition of $R_n(\tau)$, the left-truncation information is not used, because it was used along with p to add the observations.

If the original data are a combination of left- and right-censored data and if you specify the EMPIRICALCDF=TURNBULL option, then Turnbull’s method is applied to the appended set that contains no left-truncated observations.

Supplying EDF Estimates to Functions and Subroutines

The parameter initialization subroutines in distribution models and some predefined utility functions require EDF estimates. For more information, see the sections “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811 and “[Predefined Utility Functions](#)” on page 1824.

PROC SEVSELECT supplies the EDF estimates to these subroutines and functions by using two arrays, x and F , the dimension of each array, and a type of the EDF estimates. The type identifies how the EDF estimates are computed and stored. They are as follows:

- Type 1 specifies that EDF estimates are computed using the STANDARD method; that is, the data that are used for estimation are neither censored nor truncated.
- Type 2 specifies that EDF estimates are computed using either the KAPLANMEIER or the MODIFIEDKM method; that is, the data that are used for estimation are subject to truncation and one type of censoring (left or right, but not both).
- Type 3 specifies that EDF estimates are computed using the TURNBULL method; that is, the data that are used for estimation are subject to both left- and right-censoring. The data might or might not be truncated.

For Types 1 and 2, the EDF estimates are stored in arrays x and F of dimension N such that the following holds,

$$F_n(y) = \begin{cases} 0 & \text{if } y < x[1] \\ F[k] & \text{if } x[k] \leq y < x[k+1], k = 1, \dots, N-1 \\ F[N] & \text{if } x[N] \leq y \end{cases}$$

where $[k]$ denotes k th element of the array ($[1]$ denotes the first element of the array).

For Type 3, the EDF estimates are stored in arrays x and F of dimension N such that the following holds:

$$F_n(y) = \begin{cases} 0 & \text{if } y < x[1] \\ \text{undefined} & \text{if } x[2k-1] \leq y < x[2k], k = 1, \dots, (N-1)/2 \\ F[2k] = F[2k+1] & \text{if } x[2k] \leq y < x[2k+1], k = 1, \dots, (N-1)/2 \\ F[N] & \text{if } x[N] \leq y \end{cases}$$

Although the behavior of EDF is theoretically undefined for the interval $[x[2k-1], x[2k])$, for computational purposes, all predefined functions and subroutines assume that the EDF increases linearly from $F[2k-1]$ to $F[2k]$ in that interval if $x[2k-1] < x[2k]$. If $x[2k-1] = x[2k]$, which can happen when the EDF is estimated from a combination of uncensored and interval-censored data, the predefined functions and subroutines assume that $F_n(x[2k-1]) = F_n(x[2k]) = F[2k]$.

Statistics of Fit

PROC SEVSELECT computes and reports various statistics of fit to indicate how well the estimated model fits the data. The statistics belong to two categories: likelihood-based statistics and EDF-based statistics. Neg2LogLike, AIC, AICC, and SBC are likelihood-based statistics, and KS, AD, and CvM are EDF-based statistics.

In the distributed mode of execution, in which data are distributed across the grid nodes, the EDF estimates are computed by using the local data. The EDF-based statistics are computed by using these local EDF estimates. The reported value of each EDF-based statistic is an average of the values of the statistic that are computed by all the grid nodes where the data reside.

Also, for large data tables, in both single-machine and distributed modes of execution, the EDF estimates are computed by using a fraction of the input data that is governed by either the [INITSAMPLE](#) option or the default sample size.

Because of this nature of computing the EDF estimates, the EDF-based statistics of fit are an approximation of the values that would have been computed if the entire input data table were used for computing the EDF estimates. So the values that are reported for EDF-based statistics should be used only for comparing different models. The reported values should not be interpreted as true estimates of the corresponding statistics.

The likelihood-based statistics are reported for the entire input data table in both single-machine and distributed modes of execution.

The following subsections provide definitions of each category of statistics.

Likelihood-Based Statistics of Fit

Let $y_i, i = 1, \dots, N$, denote the response variable values. Let L be the likelihood as defined in the section “[Likelihood Function](#)” on page 1791. Let p denote the number of model parameters that are estimated. Note that $p = p_d + (k - k_r)$, where p_d is the number of distribution parameters, k is the number of all regression parameters, and k_r is the number of regression parameters that are found to be linearly dependent (redundant) on other regression parameters. By default, the value of p_d includes the distribution parameters that you have defined as constant by using the [dist_CONSTANTPARM](#) subroutine. You can exclude them by specifying the [NOCONSTFITSTATS](#) option in the PROC SEVSELECT statement. Given this notation, the likelihood-based statistics are defined as follows:

Neg2LogLike The log likelihood is reported as

$$\text{Neg2LogLike} = -2 \log(L)$$

The multiplying factor -2 makes it easy to compare it to the other likelihood-based statistics. A model that has a smaller value of Neg2LogLike is deemed better.

AIC Akaike’s information criterion (AIC) is defined as

$$\text{AIC} = -2 \log(L) + 2p$$

A model that has a smaller AIC value is deemed better.

AICC The corrected Akaike’s information criterion (AICC) is defined as

$$\text{AICC} = -2 \log(L) + \frac{2Np}{N - p - 1}$$

A model that has a smaller AICC value is deemed better. It corrects the finite-sample bias that AIC has when N is small compared to p . AICC is related to AIC as

$$\text{AICC} = \text{AIC} + \frac{2p(p+1)}{N-p-1}$$

As N becomes large compared to p , AICC converges to AIC. AICC is usually recommended over AIC as a model selection criterion.

SBC

The Schwarz Bayesian information criterion (SBC) is defined as

$$\text{SBC} = -2 \log(L) + p \log(N)$$

A model that has a smaller SBC value is deemed better.

EDF-Based Statistics

This class of statistics is based on the difference between the estimate of the cumulative distribution function (CDF) and the estimate of the empirical distribution function (EDF). A model that has a smaller value of the chosen EDF-based statistic is deemed better.

Let $y_i, i = 1, \dots, N$, denote the sample of N values of the response variable. Let w_i denote the normalized weight of the i th observation. If w_i^o denotes the original, unnormalized weight of the i th observation, then $w_i = N w_i^o / (\sum_{i=1}^N w_i^o)$. Let N_u denote the number of observations with unique (nonduplicate) values of the response variable. Let $W_i = \sum_{j=1}^N w_j I[y_j = y_i]$ denote the total weight of observations with a value y_i , where I is an indicator function. Let $r_i = \sum_{j=1}^N w_j I[y_j \leq y_i]$ denote the total weight of observations with a value less than or equal to y_i . Let $W = \sum_{i=1}^{N_u} W_i$ denote the total weight of all observations. Use of normalized weights implies that $W = N$.

Let $F_n(y_i)$ denote the EDF estimate that is computed by using the method that you specify in the **EMPIRICALCDF=** option. Let $Z_i = \hat{F}(y_i)$ denote the estimate of the CDF. Let $F_n(Z_i)$ denote the EDF estimate of Z_i values that are computed using the same method that is used to compute the EDF of y_i values. Using the probability integral transformation, if $F(y)$ is the true distribution of the random variable Y , then the random variable $Z = F(y)$ is uniformly distributed between 0 and 1 (D'Agostino and Stephens 1986, Ch. 4). Thus, comparing $F_n(y_i)$ with $\hat{F}(y_i)$ is equivalent to comparing $F_n(Z_i)$ with $\hat{F}(Z_i) = Z_i$ (uniform distribution).

Note the following two points regarding which CDF estimates are used for computing the test statistics:

- If you specify regression effects, then the CDF estimates Z_i that are used for computing the EDF test statistics are from a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.
- If the EDF estimates are conditional because of the truncation information, then each unconditional estimate Z_i is converted to a conditional estimate using the method described in the section “[Truncation and Conditional CDF Estimates](#)” on page 1790.

In the following, it is assumed that Z_i denotes an appropriate estimate of the CDF if you specify any truncation or regression effects. Given this, the EDF-based statistics of fit are defined as follows:

KS The Kolmogorov-Smirnov (KS) statistic computes the largest vertical distance between the CDF and the EDF. It is formally defined as follows:

$$KS = \sup_y |F_n(y) - F(y)|$$

If the STANDARD method is used to compute the EDF, then the following formula is used:

$$\begin{aligned} D^+ &= \max_i \left(\frac{r_i}{W} - Z_i \right) \\ D^- &= \max_i \left(Z_i - \frac{r_{i-1}}{W} \right) \\ KS &= \sqrt{W} \max(D^+, D^-) + \frac{0.19}{\sqrt{W}} \end{aligned}$$

Note that r_0 is assumed to be 0.

If the method used to compute the EDF is any method other than the STANDARD method, then the following formula is used:

$$\begin{aligned} D^+ &= \max_i (F_n(Z_i) - Z_i), \text{ if } F_n(Z_i) \geq Z_i \\ D^- &= \max_i (Z_i - F_n(Z_i)), \text{ if } F_n(Z_i) < Z_i \\ KS &= \sqrt{W} \max(D^+, D^-) + \frac{0.19}{\sqrt{W}} \end{aligned}$$

AD The Anderson-Darling (AD) statistic is a quadratic EDF statistic that is proportional to the expected value of the weighted squared difference between the EDF and CDF. It is formally defined as follows:

$$AD = N \int_{-\infty}^{\infty} \frac{(F_n(y) - F(y))^2}{F(y)(1 - F(y))} dF(y)$$

If the STANDARD method is used to compute the EDF, then the following formula is used:

$$AD = -W - \frac{1}{W} \sum_{i=1}^{N_u} W_i [(2r_i - 1) \log(Z_i) + (2W + 1 - 2r_i) \log(1 - Z_i)]$$

If the method used to compute the EDF is any method other than the STANDARD method, then the statistic can be computed by using the following two pieces of information:

- If the EDF estimates are computed using the KAPLANMEIER or MODIFIEDKM methods, then EDF is a step function such that the estimate $F_n(z)$ is a constant equal to $F_n(Z_{i-1})$ in interval $[Z_{i-1}, Z_i]$. If the EDF estimates are computed using the TURNBULL method, then there are two types of intervals: one in which the EDF curve is constant and the other in which the EDF curve is theoretically undefined. For computational purposes, it is assumed that the EDF curve is linear for the latter type of the interval. For each method, the EDF estimate $F_n(y)$ at y can be written as

$$F_n(z) = F_n(Z_{i-1}) + S_i(z - Z_{i-1}), \text{ for } z \in [Z_{i-1}, Z_i]$$

where S_i is the slope of the line defined as

$$S_i = \frac{F_n(Z_i) - F_n(Z_{i-1})}{Z_i - Z_{i-1}}$$

For the KAPLANMEIER or MODIFIEDKM method, $S_i = 0$ in each interval.

- Using the probability integral transform $z = F(y)$, the formula simplifies to

$$AD = N \int_{-\infty}^{\infty} \frac{(F_n(z) - z)^2}{z(1-z)} dz$$

The computation formula can then be derived from the approximation,

$$\begin{aligned} AD &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(F_n(z) - z)^2}{z(1-z)} dz \\ &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(F_n(Z_{i-1}) + S_i(z - Z_{i-1}) - z)^2}{z(1-z)} dz \\ &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(P_i - Q_i z)^2}{z(1-z)} dz \end{aligned}$$

where $P_i = F_n(Z_{i-1}) - S_i Z_{i-1}$, $Q_i = 1 - S_i$, and K is the number of points at which the EDF estimate are computed. For the TURNBULL method, $K = 2k$ for some k .

Assuming $Z_0 = 0$, $Z_{K+1} = 1$, $F_n(0) = 0$, and $F_n(Z_K) = 1$ yields the computation formula,

$$\begin{aligned} AD &= -N(Z_1 + \log(1 - Z_1) + \log(Z_K) + (1 - Z_K)) \\ &\quad + N \sum_{i=2}^K [P_i^2 A_i - (Q_i - P_i)^2 B_i - Q_i^2 C_i] \end{aligned}$$

where $A_i = \log(Z_i) - \log(Z_{i-1})$, $B_i = \log(1 - Z_i) - \log(1 - Z_{i-1})$, and $C_i = Z_i - Z_{i-1}$.

If EDF estimates are computed using the KAPLANMEIER or MODIFIEDKM method, then $P_i = F_n(Z_{i-1})$ and $Q_i = 1$, which simplifies the formula as

$$\begin{aligned} AD &= -N(1 + \log(1 - Z_1) + \log(Z_K)) \\ &\quad + N \sum_{i=2}^K [F_n(Z_{i-1})^2 A_i - (1 - F_n(Z_{i-1}))^2 B_i] \end{aligned}$$

CvM The Cramér–von Mises (CvM) statistic is a quadratic EDF statistic that is proportional to the expected value of the squared difference between the EDF and CDF. It is formally defined as follows:

$$CvM = N \int_{-\infty}^{\infty} (F_n(y) - F(y))^2 dF(y)$$

If the STANDARD method is used to compute the EDF, then the following formula is used:

$$CvM = \frac{1}{12W} + \sum_{i=1}^{N_u} W_i \left(Z_i - \frac{(2r_i - 1)}{2W} \right)^2$$

If the method used to compute the EDF is any method other than the STANDARD method, then the statistic can be computed by using the following two pieces of information:

- As described previously for the AD statistic, the EDF estimates are assumed to be piecewise linear such that the estimate $F_n(y)$ at y is

$$F_n(z) = F_n(Z_{i-1}) + S_i(z - Z_{i-1}), \text{ for } z \in [Z_{i-1}, Z_i]$$

where S_i is the slope of the line defined as

$$S_i = \frac{F_n(Z_i) - F_n(Z_{i-1})}{Z_i - Z_{i-1}}$$

For the KAPLANMEIER or MODIFIEDKM method, $S_i = 0$ in each interval.

- Using the probability integral transform $z = F(y)$, the formula simplifies to

$$\text{CvM} = N \int_{-\infty}^{\infty} (F_n(z) - z)^2 dz$$

The computation formula can then be derived from the following approximation,

$$\begin{aligned} \text{CvM} &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (F_n(z) - z)^2 dz \\ &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (F_n(Z_{i-1}) + S_i(z - Z_{i-1}) - z)^2 dz \\ &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (P_i - Q_i z)^2 dz \end{aligned}$$

where $P_i = F_n(Z_{i-1}) - S_i Z_{i-1}$, $Q_i = 1 - S_i$, and K is the number of points at which the EDF estimate are computed. For the TURNBULL method, $K = 2k$ for some k .

Assuming $Z_0 = 0$, $Z_{K+1} = 1$, and $F_n(0) = 0$ yields the following computation formula,

$$\text{CvM} = N \frac{Z_1^3}{3} + N \sum_{i=2}^{K+1} \left[P_i^2 A_i - P_i Q_i B_i - \frac{Q_i^2}{3} C_i \right]$$

where $A_i = Z_i - Z_{i-1}$, $B_i = Z_i^2 - Z_{i-1}^2$, and $C_i = Z_i^3 - Z_{i-1}^3$.

If EDF estimates are computed using the KAPLANMEIER or MODIFIEDKM method, then $P_i = F_n(Z_{i-1})$ and $Q_i = 1$, which simplifies the formula as

$$\text{CvM} = \frac{N}{3} + N \sum_{i=2}^{K+1} [F_n(Z_{i-1})^2(Z_i - Z_{i-1}) - F_n(Z_{i-1})(Z_i^2 - Z_{i-1}^2)]$$

which is similar to the formula proposed by Koziol and Green (1976).

Defining a Severity Distribution Model with the FCMP Procedure

A *severity distribution model* consists of a set of functions and subroutines that are defined using the FCMP procedure. The FCMP procedure is part of Base SAS software. Each function or subroutine must be named as `<distribution-name>_<keyword>`, where *distribution-name* is the identifying short name of the distribution and *keyword* identifies one of the functions or subroutines. The total length of the name should not exceed 32. Each function or subroutine must have a specific signature, which consists of the number of arguments, sequence and types of arguments, and return value type. The summary of all the recognized function and subroutine names and their expected behavior is given in [Table 26.6](#).

Consider the following points when you define a distribution model:

- When you define a function or subroutine requiring parameter arguments, the names and order of those arguments must be the same. Arguments other than the parameter arguments can have any name, but they must satisfy the requirements on their type and order.
- When the SEVSELECT procedure invokes any function or subroutine, it provides the necessary input values according to the specified signature, and expects the function or subroutine to prepare the output and return it according to the specification of the return values in the signature.
- You can use most of the SAS programming statements and SAS functions that you can use in a DATA step for defining the FCMP functions and subroutines. However, there are a few differences in the capabilities of the DATA step and the FCMP procedure. To learn more, see the documentation of the FCMP procedure in the [Base SAS Procedures Guide](#).
- You must specify either the PDF or the LOGPDF function. Similarly, you must specify either the CDF or the LOGCDF function. All other functions are optional, except when necessary for correct definition of the distribution. It is strongly recommended that you define the PARMINIT subroutine to provide a good set of initial values for the parameters. The information that PROC SEVSELECT provides to the PARMINIT subroutine enables you to use popular initialization approaches based on the method of moments and the method of percentile matching, but you can implement any algorithm to initialize the parameters by using the values of the response variable and the estimate of its empirical distribution function.
- The LOWERBOUNDS subroutines should be defined if the lower bound on at least one distribution parameter is different from the default lower bound of 0. If you define a LOWERBOUNDS subroutine but do not set a lower bound for some parameter inside the subroutine, then that parameter is assumed to have no lower bound (or a lower bound of $-\infty$). Hence, it is recommended that you explicitly return the lower bound for each parameter when you define the LOWERBOUNDS subroutine.
- The UPPERBOUNDS subroutines should be defined if the upper bound on at least one distribution parameter is different from the default upper bound of ∞ . If you define an UPPERBOUNDS subroutine but do not set an upper bound for some parameter inside the subroutine, then that parameter is assumed to have no upper bound (or a upper bound of ∞). Hence, it is recommended that you explicitly return the upper bound for each parameter when you define the UPPERBOUNDS subroutine.
- If you want to use the distribution in a model with regression effects, then make sure that the first parameter of the distribution is the scale parameter itself or a log-transformed scale parameter. If the first parameter is a log-transformed scale parameter, then you must define the SCALETRANSFORM function.

- In general, it is not necessary to define the gradient and Hessian functions, because the SEVSELECT procedure uses an internal system to evaluate the required derivatives. The internal system typically computes the derivatives analytically. But it might not be able to do so if your function definitions use other functions that it cannot differentiate analytically. In such cases, derivatives are approximated using a finite difference method and a note is written to the SAS log to indicate the components that are differentiated using such approximations. PROC SEVSELECT does reasonably well with these finite difference approximations. But, if you know of a way to compute the derivatives of such components analytically, then you should define the gradient and Hessian functions.

In order to use your distribution with PROC SEVSELECT, you need to record the FCMP library that contains the functions and subroutines for your distribution and other FCMP libraries that contain FCMP functions or subroutines used within your distribution's functions and subroutines. Specify all those libraries in the CMPLIB= system option by using the OPTIONS global statement. For more information about the OPTIONS statement, see *SAS Global Statements: Reference*. For more information about the CMPLIB= system option, see *SAS System Options: Reference*.

Each predefined distribution mentioned in the section “[Predefined Distributions](#)” on page 1779 has a distribution model associated with it. The functions and subroutines of all those models are available in the Sashelp.Svrtldist library. The order of the parameters in the signatures of the functions and subroutines is the same as listed in [Table 26.4](#). You do not need to use the CMPLIB= option in order to use the predefined distributions with PROC SEVSELECT. However, if you need to use the functions or subroutines of the predefined distributions in SAS statements other than the PROC SEVSELECT step (such as in a DATA step), then specify the Sashelp.Svrtldist library in the CMPLIB= system option by using the OPTIONS global statement prior to using them.

[Table 26.6](#) shows functions and subroutines that define a distribution model, and subsections after the table provide more detail. The functions are listed in alphabetical order of the keyword suffix.

Table 26.6 List of Functions and Subroutines That Define a Distribution Model

Name	Type	Required	Expected to Return
<i>dist_CDF</i>	Function	YES ¹	Cumulative distribution function value
<i>dist_CDFGRADIENT</i>	Subroutine	NO	Gradient of the CDF
<i>dist_CDFHESSIAN</i>	Subroutine	NO	Hessian of the CDF
<i>dist_CONSTANTPARM</i>	Subroutine	NO	Constant parameters
<i>dist_DESCRIPTION</i>	Function	NO	Description of the distribution
<i>dist_LOGCDF</i>	Function	YES ¹	Log of cumulative distribution function value
<i>dist_LOGCDFGRADIENT</i>	Subroutine	NO	Gradient of the LOGCDF
<i>dist_LOGCDFHESSIAN</i>	Subroutine	NO	Hessian of the LOGCDF
<i>dist_LOGPDF</i>	Function	YES ²	Log of probability density function value
<i>dist_LOGPDFGRADIENT</i>	Subroutine	NO	Gradient of the LOGPDF
<i>dist_LOGPDFHESSIAN</i>	Subroutine	NO	Hessian of the LOGPDF
<i>dist_LOGSDF</i>	Function	NO	Log of survival function value
<i>dist_LOGSDFGRADIENT</i>	Subroutine	NO	Gradient of the LOGSDF
<i>dist_LOGSDFHESSIAN</i>	Subroutine	NO	Hessian of the LOGSDF
<i>dist_LOWERBOUNDS</i>	Subroutine	NO	Lower bounds on parameters
<i>dist_PARMINIT</i>	Subroutine	NO	Initial values for parameters
<i>dist_PDF</i>	Function	YES ²	Probability density function value
<i>dist_PDFGRADIENT</i>	Subroutine	NO	Gradient of the PDF
<i>dist_PDFHESSIAN</i>	Subroutine	NO	Hessian of the PDF
<i>dist_QUANTILE</i>	Function	NO	Quantile for a particular CDF value
<i>dist_SCALETRANSFORM</i>	Function	NO	Type of relationship between the first distribution parameter and the scale parameter
<i>dist_SDF</i>	Function	NO	Survival function value
<i>dist_SDFGRADIENT</i>	Subroutine	NO	Gradient of the SDF
<i>dist_SDFHESSIAN</i>	Subroutine	NO	Hessian of the SDF
<i>dist_UPPERBOUNDS</i>	Subroutine	NO	Upper bounds on parameters

Notes:

1. Either the *dist_CDF* or the *dist_LOGCDF* function must be defined.
2. Either the *dist_PDF* or the *dist_LOGPDF* function must be defined.

The signature syntax and semantics of each function or subroutine are as follows:

dist_CDF

defines a function that returns the value of the cumulative distribution function (CDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type:* Function
- *Required:* YES
- *Number of arguments:* $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments:*
 - x Numeric value of the random variable at which the CDF value should be evaluated
 - $p1$ Numeric value of the first parameter
 - $p2$ Numeric value of the second parameter
 - $\vdots$
 - pm Numeric value of the m th parameter
- *Return value:* Numeric value that contains the CDF value $F(x; p_1, p_2, \dots, p_m)$

If you want to consider this distribution as a candidate distribution when you estimate a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$F(x; p_1, p_2, \dots, p_m) = F\left(\frac{x}{p_1}; 1, p_2, \dots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$F(x; p_1, p_2, \dots, p_m) = F\left(\frac{x}{\exp(p_1)}; 0, p_2, \dots, p_m\right)$$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_CDF(x, P1, P2);
  /* Code to compute CDF by using x, P1, and P2 */

  F = <computed CDF>;
  return (F);
endsub;
```

dist_CONSTANTPARM

defines a subroutine that specifies constant parameters. A parameter is *constant* if it is required for defining a distribution but is not subject to optimization in PROC SEVSELECT. Constant parameters are required to be part of the model in order to compute the PDF or the CDF of the distribution. Typically, values of these parameters are known a priori or estimated using some means other than the maximum likelihood method used by PROC SEVSELECT. You can estimate them inside the *dist_PARMINIT* subroutine. Once initialized, the parameters remain constant in the context of PROC SEVSELECT; that is, they retain their initial value. PROC SEVSELECT estimates only the nonconstant parameters.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: k , where k is the number of constant parameters
- *Sequence and type of arguments*:

```
constant parameter 1  Name of the first constant parameter
                    ⋮
constant parameter  $k$   Name of the  $k$ th constant parameter
```

- *Return value*: None

Here is a sample structure of the subroutine for a distribution named ‘FOO’ that has P3 and P5 as its constant parameters, assuming that distribution has at least three parameters:

```
subroutine FOO_CONSTANTPARM(p5, p3);
endsub;
```

Note the following points when you specify the constant parameters:

- At least one distribution parameter must be free to be optimized; that is, if a distribution has total m parameters, then k must be strictly less than m .
- If you want to use this distribution for modeling regression effects, then the first parameter must not be a constant parameter.
- The order of arguments in the signature of this subroutine does not matter as long as each argument’s name matches the name of one of the parameters that are defined in the signature of the *dist_PDF* function.
- The constant parameters must be specified in signatures of all the functions and subroutines that accept distribution parameters as their arguments.
- You must provide a nonmissing initial value for each constant parameter by using one of the supported parameter initialization methods.

*dist_***DESCRIPTION**

defines a function that returns a description of the distribution.

- *Type*: Function
- *Required*: NO
- *Number of arguments*: None
- *Sequence and type of arguments*: Not applicable
- *Return value*: Character value containing a description of the distribution

Here is a sample structure of the function for a distribution named ‘FOO’:

```

function FOO_DESCRIPTION() $48;
  length desc $48;
  desc = "A model for a continuous distribution named foo";
  return (desc);
endsub;

```

dist_LOGcore

defines a function that returns the natural logarithm of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, or SDF.

- *Type*: Function
- *Required*: YES only if *core* is PDF or CDF and you have not defined that *core* function; otherwise, NO
- *Number of arguments*: $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments*:
 - x Numeric value of the random variable at which the natural logarithm of the *core* function should be evaluated
 - p1 Numeric value of the first parameter
 - p2 Numeric value of the second parameter
 - ⋮
 - pm Numeric value of the m th parameter
- *Return value*: Numeric value that contains the natural logarithm of the *core* function

Here is a sample structure of the function for the core function PDF of a distribution named 'FOO':

```

function FOO_LOGPDF(x, P1, P2);
  /* Code to compute LOGPDF by using x, P1, and P2 */

  l = <computed LOGPDF>;
  return (l);
endsub;

```

dist_LOWERBOUNDS

defines a subroutine that returns lower bounds for the parameters of the distribution. If this subroutine is not defined for a particular distribution, then the SEVSELECT procedure assumes a lower bound of 0 for each parameter. If a lower bound of l_i is returned for a parameter p_i , then the SEVSELECT procedure assumes that $l_i < p_i$ (strict inequality). If a missing value is returned for some parameter, then the SEVSELECT procedure assumes that there is no lower bound for that parameter (equivalent to a lower bound of $-\infty$).

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: m , where m is the number of distribution parameters
- *Sequence and type of arguments*:

- | | |
|----|---|
| p1 | Output argument that returns the lower bound on the first parameter. You must specify this in the OUTARGS statement inside the subroutine's definition. |
| p2 | Output argument that returns the lower bound on the second parameter. You must specify this in the OUTARGS statement inside the subroutine's definition. |
| ⋮ | |
| pm | Output argument that returns the lower bound on the <i>m</i> th parameter. You must specify this in the OUTARGS statement inside the subroutine's definition. |
- *Return value:* The results, lower bounds on parameter values, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```
subroutine FOO_LOWERBOUNDS(p1, p2);
    outargs p1, p2;

    p1 = <lower bound for P1>;
    p2 = <lower bound for P2>;
endsub;
```

*dist_***PARMINIT**

defines a subroutine that returns the initial values for the distribution's parameters given an empirical distribution function (EDF) estimate.

- *Type:* Subroutine
- *Required:* NO
- *Number of arguments:* $m + 4$, where m is the number of distribution parameters
- *Sequence and type of arguments:*

- | | |
|-------|---|
| dim | Input numeric value that contains the dimension of the x, nx, and F array arguments. |
| x{*} | Input numeric array of dimension <i>dim</i> that contains values of the random variables at which the EDF estimate is available. It can be assumed that x contains values in an increasing order. In other words, if $i < j$, then $x[i] < x[j]$. |
| nx{*} | Input numeric array of dimension <i>dim</i> . Each $nx[i]$ contains the number of observations in the original data that have the value $x[i]$. |
| F{*} | Input numeric array of dimension <i>dim</i> . Each $F[i]$ contains the EDF estimate for $x[i]$. This estimate is computed by the SEVSELECT procedure based on the options that you specify in the LOSS statement and the EMPIRICALCDF= option. |
| Ftype | Input numeric value that contains the type of the EDF estimate that is stored in x and F. For definitions of types, see the section " Supplying EDF Estimates to Functions and Subroutines " on page 1805. |
| p1 | Output argument that returns the initial value of the first parameter. You must specify this in the OUTARGS statement inside the subroutine's definition. |

p2 Output argument that returns the initial value of the second parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.
 ⋮

pm Output argument that returns the initial value of the m th parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.

- *Return value:* The results, initial values of the parameters, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```
subroutine FOO_PARMINIT(dim, x{*}, nx{*}, F{*}, Ftype, p1, p2);
  outargs p1, p2;

  /* Code to initialize values of P1 and P2 by using
    dim, x, nx, and F */

  p1 = <initial value for p1>;
  p2 = <initial value for p2>;
endsub;
```

dist_PDF

defines a function that returns the value of the probability density function (PDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type:* Function
- *Required:* YES
- *Number of arguments:* $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments:*

x Numeric value of the random variable at which the PDF value should be evaluated

p1 Numeric value of the first parameter

p2 Numeric value of the second parameter

⋮

pm Numeric value of the m th parameter

- *Return value:* Numeric value that contains the PDF value $f(x; p_1, p_2, \dots, p_m)$

If you want to consider this distribution as a candidate distribution when you estimate a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$f(x; p_1, p_2, \dots, p_m) = \frac{1}{p_1} f\left(\frac{x}{p_1}; 1, p_2, \dots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$f(x; p_1, p_2, \dots, p_m) = \frac{1}{\exp(p_1)} f\left(\frac{x}{\exp(p_1)}; 0, p_2, \dots, p_m\right)$$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_PDF(x, P1, P2);
    /* Code to compute PDF by using x, P1, and P2 */

    f = <computed PDF>;
    return (f);
endsub;
```

*dist_***QUANTILE**

defines a function that returns the quantile of the distribution at the specified value of the CDF for the specified values of distribution parameters.

- *Type:* Function
- *Required:* NO
- *Number of arguments:* $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments:*
 - cdf Numeric value of the cumulative distribution function (CDF) for which the quantile should be evaluated
 - p1 Numeric value of the first parameter
 - p2 Numeric value of the second parameter
 - ⋮
 - pm Numeric value of the m th parameter
- *Return value:* Numeric value that contains the quantile $F^{-1}(\text{cdf}; p_1, p_2, \dots, p_m)$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_QUANTILE(c, P1, P2);
    /* Code to compute quantile by using c, P1, and P2 */

    Q = <computed quantile>;
    return (Q);
endsub;
```

*dist_***SCALETRANSFORM**

defines a function that returns a keyword to identify the transform that needs to be applied to the scale parameter to convert it to the first parameter of the distribution.

If you want to use this distribution for modeling regression effects, then the first parameter of this distribution must be a scale parameter. However, for some distributions, a typical or convenient parameterization might not have a scale parameter, but one of the parameters can be a simple transform of the scale parameter. As an example, consider a typical parameterization of the lognormal distribution with two parameters, location μ and shape σ , for which the PDF is defined as follows:

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$$

You can reparameterize this distribution to contain a parameter θ instead of the parameter μ such that $\mu = \log(\theta)$. The parameter θ would then be a scale parameter. However, if you want to specify the distribution in terms of μ and σ (which is a more recognized form of the lognormal distribution) and still allow it as a candidate distribution for estimating regression effects, then instead of writing another distribution with parameters θ and σ , you can simply define the distribution with μ as the first parameter and specify that it is the logarithm of the scale parameter.

- *Type:* Function
- *Required:* NO
- *Number of arguments:* None
- *Sequence and type of arguments:* Not applicable
- *Return value:* Character value that contains one of the following keywords:

LOG	specifies that the first parameter is the logarithm of the scale parameter.
IDENTITY	specifies that the first parameter is a scale parameter without any transformation.

If you do not specify this function, then the IDENTITY transform is assumed.

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_SCALETRANSFORM() $8;
  length xform $8;
  xform = "IDENTITY";
  return (xform);
endsub;
```

dist_SDF

defines a function that returns the value of the survival distribution function (SDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type:* Function
- *Required:* NO
- *Number of arguments:* $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments:*

x	Numeric value of the random variable at which the SDF value should be evaluated
p1	Numeric value of the first parameter
p2	Numeric value of the second parameter
⋮	
pm	Numeric value of the m th parameter

- *Return value:* Numeric value that contains the SDF value $S(x; p_1, p_2, \dots, p_m)$

If you want to consider this distribution as a candidate distribution when estimating a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$S(x; p_1, p_2, \dots, p_m) = S\left(\frac{x}{p_1}; 1, p_2, \dots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$S(x; p_1, p_2, \dots, p_m) = S\left(\frac{x}{\exp(p_1)}; 0, p_2, \dots, p_m\right)$$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_SDF(x, P1, P2);
  /* Code to compute SDF by using x, P1, and P2 */

  S = <computed SDF>;
  return (S);
endsub;
```

dist_UPPERBOUNDS

defines a subroutine that returns upper bounds for the parameters of the distribution. If this subroutine is not defined for a particular distribution, then the SEVSELECT procedure assumes that there is no upper bound for any of the parameters. If an upper bound of u_i is returned for a parameter p_i , then the SEVSELECT procedure assumes that $p_i < u_i$ (strict inequality). If a missing value is returned for some parameter, then the SEVSELECT procedure assumes that there is no upper bound for that parameter (equivalent to an upper bound of ∞).

- *Type:* Subroutine
- *Required:* NO
- *Number of arguments:* m , where m is the number of distribution parameters
- *Sequence and type of arguments:*

p_1	Output argument that returns the upper bound on the first parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.
p_2	Output argument that returns the upper bound on the second parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.
$\vdots$	
p_m	Output argument that returns the upper bound on the m th parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.
- *Return value:* The results, upper bounds on parameter values, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```

subroutine FOO_UPPERBOUNDS(p1, p2);
    outargs p1, p2;

    p1 = <upper bound for P1>;
    p2 = <upper bound for P2>;
endsub;

```

*dist_core*GRADIENT

defines a subroutine that returns the gradient vector of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, SDF, LOGPDF, LOGCDF, or LOGSDF.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: $m + 2$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

x	Numeric value of the random variable at which the gradient should be evaluated
p_1	Numeric value of the first parameter
p_2	Numeric value of the second parameter
$\vdots$	
p_m	Numeric value of the m th parameter
$\text{grad}\{*\}$	Output numeric array of size m that contains the gradient vector evaluated at the specified values. If h denotes the value of the <i>core</i> function, then the expected order of the values in the array is as follows: $\frac{\partial h}{\partial p_1} \frac{\partial h}{\partial p_2} \dots \frac{\partial h}{\partial p_m}$

- *Return value*: Numeric array that contains the gradient evaluated at x for the parameter values $(p_1, p_2, \dots, p_m)$

Here is a sample structure of the function for the core function CDF of a distribution named 'FOO':

```

subroutine FOO_CDFGRADIENT(x, P1, P2, grad{*});
    outargs grad;

    /* Code to compute gradient by using x, P1, and P2 */
    grad[1] = <partial derivative of CDF w.r.t. P1
              evaluated at x, P1, P2>;
    grad[2] = <partial derivative of CDF w.r.t. P2
              evaluated at x, P1, P2>;
endsub;

```


dist_coreHESSIAN

defines a subroutine that returns the Hessian matrix of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, SDF, LOGPDF, LOGCDF, or LOGSDF.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: $m + 2$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

x	Numeric value of the random variable at which the Hessian matrix should be evaluated
p1	Numeric value of the first parameter
p2	Numeric value of the second parameter
⋮	
pm	Numeric value of the m th parameter
hess{*}	Output numeric array of size $m(m + 1)/2$ that contains the lower triangular portion of the Hessian matrix in a packed vector form, evaluated at the specified values. If h denotes the value of the <i>core</i> function, then the expected order of the values in the array is as follows: $\frac{\partial^2 h}{\partial p_1^2} \mid \frac{\partial^2 h}{\partial p_1 \partial p_2} \frac{\partial^2 h}{\partial p_2^2} \mid \dots \mid \frac{\partial^2 h}{\partial p_1 \partial p_m} \frac{\partial^2 h}{\partial p_2 \partial p_m} \dots \frac{\partial^2 h}{\partial p_m^2}$

- *Return value*: Numeric array that contains the lower triangular portion of the Hessian matrix evaluated at x for the parameter values $(p_1, p_2, \dots, p_m)$

Here is a sample structure of the subroutine for the core function LOGSDF of a distribution named 'FOO':

```
subroutine FOO_LOGSDFHESSIAN(x, P1, P2, hess{*});
    outargs hess;

    /* Code to compute Hessian by using x, P1, and P2 */
    hess[1] = <second order partial derivative of LOGSDF
              w.r.t. P1 evaluated at x, P1, P2>;
    hess[2] = <second order partial derivative of LOGSDF
              w.r.t. P1 and P2 evaluated at x, P1, P2>;
    hess[3] = <second order partial derivative of LOGSDF
              w.r.t. P2 evaluated at x, P1, P2>;
endsub;
```

Predefined Utility Functions

The following predefined utility functions are provided with the SEVSELECT procedure and are available in the Sashelp.SvrtDIST library:

SVRTUTIL_EDF

This function computes the empirical distribution function (EDF) estimate at the specified value of the random variable given the EDF estimate for a sample.

- *Type:* Function
- *Signature:* SVRTUTIL_EDF(y, n, x{*}, F{*}, Ftype)
- *Argument description:*

y	Value of the random variable at which the EDF estimate is desired
n	Dimension of the x and F input arrays
x{*}	Input numeric array of dimension n that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension n in which each $F[i]$ contains the EDF estimate for $x[i]$. These values must be sorted in nondecreasing order.
Ftype	Type of the empirical estimate that is stored in the x and F arrays. For definitions of types, see the section “ Supplying EDF Estimates to Functions and Subroutines ” on page 1805.
- *Return value:* The EDF estimate at y

The type of the sample EDF estimate determines how the EDF estimate at y is computed. For more information, see the section “[Supplying EDF Estimates to Functions and Subroutines](#)” on page 1805.

SVRTUTIL_EMPLIMMOMENT

This function computes the empirical estimate of the limited moment of specified order for the specified upper limit, given the EDF estimate for a sample.

- *Type:* Function
- *Signature:* SVRTUTIL_EMPLIMMOMENT(k, u, n, x{*}, F{*}, Ftype)
- *Argument description:*

k	Order of the desired empirical limited moment
u	Upper limit on the value of the random variable to be used in the computation of the desired empirical limited moment
n	Dimension of the x and F input arrays
x{*}	Input numeric array of dimension n that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension n in which each $F[i]$ contains the EDF estimate for $x[i]$. These values must be sorted in nondecreasing order.

Ftype Type of the empirical estimate that is stored in the x and F arrays. For definitions of types, see the section “[Supplying EDF Estimates to Functions and Subroutines](#)” on page 1805.

- *Return value:* The desired empirical limited moment

The empirical limited moment is computed by using the empirical estimate of the CDF. If $F_n(x)$ denotes the EDF at x , then the empirical limited moment of order k with upper limit u is defined as

$$E_n[(X \wedge u)^k] = k \int_0^u (1 - F_n(x))x^{k-1} dx$$

The SVRTUTIL_EMPLIMMOMENT function uses the piecewise linear nature of $F_n(x)$ as described in the section “[Supplying EDF Estimates to Functions and Subroutines](#)” on page 1805 to compute the integration.

SVRTUTIL_HILLCUTOFF

This function computes an estimate of the value where the right tail of a distribution is expected to begin. The function implements the algorithm described in Danielsson et al. 2001. The description of the algorithm uses the following notation:

n	Number of observations in the original sample
B	Number of bootstrap samples to draw
m_1	Size of the bootstrap sample in the first step of the algorithm ($m_1 < n$)
$x_{(i)}^{j,m}$	i th-order statistic of j th bootstrap sample of size m ($1 \leq i \leq m, 1 \leq j \leq B$)
$x_{(i)}$	i th-order statistic of the original sample ($1 \leq i \leq n$)

Given the input sample x and values of B and m_1 , the steps of the algorithm are as follows:

1. Take B bootstrap samples of size m_1 from the original sample.
2. Find the integer k_1 that minimizes the bootstrap estimate of the mean squared error:

$$k_1 = \arg \min_{1 \leq k < m_1} Q(m_1, k)$$

3. Take B bootstrap samples of size $m_2 = m_1^2/n$ from the original sample.
4. Find the integer k_2 that minimizes the bootstrap estimate of the mean squared error:

$$k_2 = \arg \min_{1 \leq k < m_2} Q(m_2, k)$$

5. Compute the integer k_{opt} , which is used for computing the cutoff point:

$$k_{\text{opt}} = \frac{k_1^2}{k_2} \left(\frac{\log(k_1)}{2 \log(m_1) - \log(k_1)} \right)^{2 - 2 \log(k_1) / \log(m_1)}$$

6. Set the cutoff point equal to $x_{(k_{\text{opt}}+1)}$.

The bootstrap estimate of the mean squared error is computed as

$$Q(m, k) = \frac{1}{B} \sum_{j=1}^B \text{MSE}_j(m, k)$$

The mean squared error of j th bootstrap sample is computed as

$$\text{MSE}_j(m, k) = (M_j(m, k) - 2(\gamma_j(m, k))^2)^2$$

where $M_j(m, k)$ is a control variate proposed by Danielsson et al. 2001,

$$M_j(m, k) = \frac{1}{k} \sum_{i=1}^k \left(\log(x_{(m-i+1)}^{j,m}) - \log(x_{(m-k)}^{j,m}) \right)^2$$

and $\gamma_j(m, k)$ is the Hill's estimator of the tail index (Hill 1975),

$$\gamma_j(m, k) = \frac{1}{k} \sum_{i=1}^k \log(x_{(m-i+1)}^{j,m}) - \log(x_{(m-k)}^{j,m})$$

This algorithm has two tuning parameters, B and m_1 . The number of bootstrap samples B is chosen based on the availability of computational resources. The optimal value of m_1 is chosen such that the following ratio, $R(m_1)$, is minimized:

$$R(m_1) = \frac{(Q(m_1, k_1))^2}{Q(m_2, k_2)}$$

The SVRTUTIL_HILLCUTOFF utility function implements the preceding algorithm. It uses the grid search method to compute the optimal value of m_1 .

- *Type:* Function
- *Signature:* SVRTUTIL_HILLCUTOFF(n , $x\{*\}$, b , s , $status$)
- *Argument description:*

n	Dimension of the array x
$x\{*\}$	Input numeric array of dimension n that contains the sample
b	Number of bootstrap samples used to estimate the mean squared error. If b is less than 10, then a default value of 50 is used.
s	Approximate number of steps used to search the optimal value of m_1 in the range $[n^{0.75}, n - 1]$. If s is less than or equal to 1, then a default value of 10 is used.
$status$	Output argument that contains the status of the algorithm. If the algorithm succeeds in computing a valid cutoff point, then <i>status</i> is set to 0. If the algorithm fails, then <i>status</i> is set to 1.
- *Return value:* The cutoff value where the right tail is estimated to start. If the size of the input sample is inadequate ($n \leq 5$), then a missing value is returned and *status* is set to a missing value. If the algorithm fails to estimate a valid cutoff value (*status* = 1), then the fifth-largest value in the input sample is returned.

SVRTUTIL_PERCENTILE

This function computes the specified empirical percentile given the EDF estimates.

- *Type:* Function
- *Signature:* SVRTUTIL_PERCENTILE(p, n, x{*}, F{*}, Ftype)
- *Argument description:*

p	Desired percentile. The value must be in the interval (0,1). The function returns the 100 p th percentile.
n	Dimension of the x and F input arrays
x{*}	Input numeric array of dimension n that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension n in which each $F[i]$ contains the EDF estimate for $x[i]$. These values must be sorted in nondecreasing order.
Ftype	Type of the empirical estimate that is stored in the x and F arrays. For definitions of types, see the section “ Supplying EDF Estimates to Functions and Subroutines ” on page 1805.
- *Return value:* The 100 p th percentile of the input sample

The method used to compute the percentile depends on the type of the EDF estimate (Ftype argument).

Ftype = 1 Smoothed empirical estimates are computed using the method described in Klugman, Panjer, and Willmot (1998). Let $\lfloor x \rfloor$ denote the greatest integer less than or equal to x . Define $g = \lfloor p(n+1) \rfloor$ and $h = p(n+1) - g$. Then the empirical percentile $\hat{\pi}_p$ is defined as

$$\hat{\pi}_p = (1-h)x[g] + hx[g+1]$$

This method does not work if $p < 1/(n+1)$ or $p > n/(n+1)$. If $p < 1/(n+1)$, then the function returns $\hat{\pi}_p = x[1]/2$, which assumes that the EDF is 0 in the interval $[0, x[1])$. If $p > n/(n+1)$, then $\hat{\pi}_p = x[n]$.

Ftype = 2 If $p < F[1]$, then $\hat{\pi}_p = x[1]/2$, which assumes that the EDF is 0 in the interval $[0, x[1])$. If $|p - F[i]| < \epsilon$ for some value of i and $i < n$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = \frac{x[i] + x[i+1]}{2}$$

where ϵ is a machine-precision constant as returned by the SAS function CONSTANT(‘MACEPS’). If $F[i-1] < p < F[i]$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = x[i]$$

If $p \geq F[n]$, then $\hat{\pi}_p = x[n]$.

Ftype = 3 If $p < F[1]$, then $\hat{\pi}_p = x[1]/2$, which assumes that the EDF is 0 in the interval $[0, x[1])$. If $|p - F[i]| < \epsilon$ for some value of i and $i < n$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = \frac{x[i] + x[i+1]}{2}$$

where ϵ is a machine-precision constant as returned by the SAS function `CONSTANT('MACEPS')`. If $F[i - 1] < p < F[i]$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = x[i - 1] + (p - F[i - 1]) \frac{x[i] - x[i - 1]}{F[i] - F[i - 1]}$$

If $p \geq F[n]$, then $\hat{\pi}_p = x[n]$.

SVRTUTIL_RAWMOMENTS

This subroutine computes the raw moments of a sample.

- *Type:* Subroutine
- *Signature:* `SVRTUTIL_RAWMOMENTS(n, x{*}, nx{*}, nRaw, raw{*})`
- *Argument description:*

<code>n</code>	Dimension of the <code>x</code> and <code>nx</code> input arrays
<code>x{*}</code>	Input numeric array of dimension <code>n</code> that contains distinct values of the random variable that are observed in the sample
<code>nx{*}</code>	Input numeric array of dimension <code>n</code> in which each <code>nx[i]</code> contains the number of observations in the sample that have the value <code>x[i]</code>
<code>nRaw</code>	Desired number of raw moments. The output array <code>raw</code> contains the first <code>nRaw</code> raw moments.
<code>raw{*}</code>	Output array of raw moments. The k th element in the array (<code>raw{k}</code>) contains the k th raw moment, where $1 \leq k \leq nRaw$.
- *Return value:* Numeric array `raw` that contains the first `nRaw` raw moments. The array contains missing values if the sample has no observations (that is, if all the values in the `nx` array add up to zero).

SVRTUTIL_SORT

This function sorts an array of numeric values in an ascending or descending order.

- *Type:* Subroutine
- *Signature:* `SVRTUTIL_SORT(n, x{*}, flag)`
- *Argument description:*

<code>n</code>	Dimension of the input array <code>x</code>
<code>x{*}</code>	Numeric array that contains the values to be sorted at input. The subroutine uses the same array to return the sorted values.
<code>flag</code>	A numeric value that controls the sort order. If <code>flag</code> is 0, then the values are sorted in an ascending order. If <code>flag</code> has any value other than 0, then the values are sorted in descending order.
- *Return value:* Numeric array `x`, which is sorted in place (that is, the sorted array is stored in the same storage area occupied by the input array `x`)

You can use the following predefined functions when you use the FCMP procedure to define functions and subroutines. They are summarized here for your information. For more information, see the FCMP procedure documentation in *Base SAS Procedures Guide*.

INVCDF

This function computes the quantile from any continuous probability distribution by numerically inverting the CDF of that distribution. You need to specify the CDF function of the distribution, the values of its parameters, and the cumulative probability to compute the quantile.

LIMMOMENT

This function computes the limited moment of order k with upper limit u for any continuous probability distribution. The limited moment is defined as

$$\begin{aligned} E[(X \wedge u)^k] &= \int_0^u x^k f(x) dx + \int_u^\infty u^k f(x) dx \\ &= \int_0^u x^k f(x) dx + u^k (1 - F(u)) \end{aligned}$$

where $f(x)$ and $F(x)$ denote the PDF and the CDF of the distribution, respectively. The LIMMOMENT function uses the following alternate definition, which can be derived using integration-by-parts:

$$E[(X \wedge u)^k] = k \int_0^u (1 - F(x)) x^{k-1} dx$$

You need to specify the CDF function of the distribution, the values of its parameters, and the values of k and u to compute the limited moment.

Scoring Functions

Scoring refers to the act of evaluating a distribution function, such as LOGPDF, SDF, or QUANTILE, on an observation by using the fitted parameter estimates of that distribution. You can do scoring in a DATA step by using the `OUTEST=` data table that you create using PROC SEVSELECT. However, that approach requires some cumbersome programming. To simplify the scoring process, you can use PROC SEVSELECT to create scoring functions for each fitted distribution.

As an example, assume that you fit the Pareto distribution by using PROC SEVSELECT and that it converges. Further assume that you want to use the fitted distribution to evaluate the probability of observing a loss value greater than some set of regulatory limits $\{L\}$ that are encoded in a data table. You can simplify this scoring process as follows. First, in the PROC SEVSELECT step that fits your distributions, you create the scoring functions library by specifying the `OUTSCORELIB` statement as illustrated in the following steps:

```
proc sevselect data=mylib.input;
  loss lossclaim;
  dist pareto;
  outscorelib outlib=sasuser.fitdist;
run;
```

Upon successful completion, if the Pareto distribution model has converged, then the library Sasuser.Fitdist contains the `SEV_SDF` scoring function in addition to other scoring functions, such as `SEV_PDF`, `SEV_LOGPDF`, and so on. Further, PROC SEVSELECT also sets the CMPLIB system option to include the Sasuser.Fitdist library. If the set of limits $\{L\}$ is recorded in the variable Limit in the scoring data set Work.Limits, then you can submit the following DATA step to compute the probability of seeing a loss greater than each limit:

```

data prob;
  set work.limits;
  exceedance_probability = sev_sdf(limit);
run;

```

Without using scoring functions, you can still perform this scoring task, but the DATA step that you need to accomplish it becomes more complicated and less flexible. For example, you would need to read the parameter estimates from some output created by PROC SEVSELECT. To do that, you would need to know the parameter names, which are different for different distributions; this in turn would require you to write a specific DATA step for each distribution or to write a SAS macro. Scoring functions enable you to accomplish that task much more easily.

If you fit multiple distributions, then you can specify the COMMONPACKAGE option in the OUTSCORELIB statement as follows:

```

proc sevselect data=mylib.input;
  loss lossclaim;
  dist exp pareto weibull;
  outscorelib outlib=sasuser.fitdist commonpackage;
run;

```

The preceding step creates scoring functions such as *SEV_SDF_Exp*, *SEV_SDF_Pareto*, and *SEV_SDF_Weibull*. You can use them to compare the probabilities of exceeding the limit for different distributions by using the following DATA step:

```

data prob;
  set work.limits;
  exceedance_exp = sev_sdf_exp(limit);
  exceedance_pareto = sev_sdf_pareto(limit);
  exceedance_weibull = sev_sdf_weibull(limit);
run;

```

Formal Description

PROC SEVSELECT creates a scoring function for each distribution function. A distribution function is defined as any function named *dist_suffix*, where *dist* is the name of a distribution that you specify in the DIST statement and the function's signature is identical to the signature of the required distribution function, such as *dist_CDF* or *dist_LOGCDF*. For example, for the function 'FOO_BAR' to be a distribution function, you must specify the distribution 'FOO' in the DIST statement, and you must define 'FOO_BAR' in the following manner if the distribution 'FOO' has parameters named 'P1' and 'P2':

```

function FOO_BAR(y, P1, P2);
  /* Code to compute BAR by using y, P1, and P2 */
  R = <computed BAR>;
  return (R);
endsub;

```

For more information about the signature that defines a distribution function, see the description of the *dist_CDF* function in the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 1811.

The name and package of the scoring function of a distribution function depend on whether you specify the COMMONPACKAGE option in the OUTSCORELIB statement.

When you do not specify the `COMMONPACKAGE` option, the scoring function that corresponds to the distribution function *dist_suffix* is named *SEV_suffix*, where *SEV_* is the standard prefix of all scoring functions. The scoring function is created in a package named *dist*. Each scoring function accepts only one argument, the value of the loss variable, and returns the same value as the value that is returned by the corresponding distribution function for the final estimates of the distribution's parameters. For example, for the preceding 'FOO_BAR' distribution function, the scoring function named 'SEV_BAR' is created in the package named 'FOO' and 'SEV_BAR' has the following signature:

```
function SEV_BAR(y);
  /* returns value of FOO_BAR for the supplied value
    of y and fitted values of P1, P2 */
endsub;
```

If you specify the `COMMONPACKAGE` option in the `OUTSCORELIB` statement, then the scoring function that corresponds to the distribution function *dist_suffix* is named *SEV_suffix_dist*, where *SEV_* is the standard prefix of all scoring functions. The scoring function is created in a package named *sevfit*. For example, for the preceding 'FOO_BAR' distribution function, if you specify the `COMMONPACKAGE` option, the scoring function named 'SEV_BAR_FOO' is created in the *sevfit* package and 'SEV_BAR_FOO' has the following signature:

```
function SEV_BAR_FOO(y);
  /* returns value of FOO_BAR for the supplied value
    of y and fitted values of P1, P2 */
endsub;
```

Scoring Functions for the Scale Regression Model

If you use the `SCALEMODEL` statement to specify a scale regression model, then `PROC SEVSELECT` generates the scoring functions when you specify only singleton continuous effects. If you specify interaction or classification effects, then scoring functions are not generated.

For a scale regression model, the estimate of the scale parameter or the log-transformed scale parameter of the distribution depends on the values of the regressors. So `PROC SEVSELECT` creates a scoring function that has the following signature, where *x{*}* represents the array of regressors:

```
function SEV_BAR(y, x{*});
  /* returns value of FOO_BAR for the supplied value of x and fitted values of P1, P2 */
endsub;
```

As an illustration of using this form, assume that you submit the following `PROC SEVSELECT` step to create the scoring library `Sasuser.Scalescore`:

```
proc sevselect data=mylib.input;
  loss lossclaim;
  scalemodel x1-x3;
  dist pareto;
  outscorelib outlib=sasuser.scalescore;
run;
```

Your scoring data set must contain all the regressors that you specify in the `SCALEMODEL` statement. You can submit the following `DATA` step to score observations by using the scale regression model:

```

data prob;
  array regvals{*} x1-x3;
  set work.limits;
  exceedance_probability = sev_sdf(limit, regvals);
run;

```

PROC SEVSELECT creates two utility functions—SEV_NUMREG and SEV_REGNAME—in the OUTLIB= library, which return the number of regressors and name of a given regressor, respectively. They are described in detail in the next section. These utility functions are useful when you do not have easy access to the regressor names in the SCALEMODEL statement. You can use the utility functions as follows:

```

data prob;
  array regvals{10} _temporary_;
  set work.limits;
  do i = 1 to sev_numreg();
    regvals(i) = input(vvaluex(sev_regname(i)), best12.);
  end;
  exceedance_probability = sev_sdf(limit, regvals);
run;

```

The dimension of the regressor values array that you supply to the scoring function must be equal to $K + L$, where K is the number of regressors that you specify in the SCALEMODEL statement irrespective of whether PROC SEVSELECT deems any of those regressors to be redundant. L is 1 if you specify an OFFSET= variable in the SCALEMODEL statement, and 0 otherwise.

Utility Functions and Subroutines in the OUTLIB= Library

In addition to creating the scoring functions for all distribution functions, PROC SEVSELECT creates the following utility functions and subroutines in the OUTLIB= library.

SEV_NUMPARM | SEV_NUMPARM_dist

is a function that returns the number of distribution parameters and has the following signature:

- *Type:* Function
- *Number of arguments:* 0
- *Sequence and type of arguments:* Not applicable
- *Return value:* Numeric value that contains the number of distribution parameters

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then a function named SEV_NUMPARM is created in the package of each distribution. Here is a sample structure of the code that PROC SEVSELECT uses to define the function:

```

function SEV_NUMPARM();
  n = <number of distribution parameters>;
  return (n);
endsub;

```

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution *dist*, the function named SEV_NUMPARM_dist is created in the *sevfit* package. SEV_NUMPARM_dist has the same structure as the SEV_NUMPARM function that is described previously.

SEV_PARMEST | SEV_PARMEST_dist

is a subroutine that returns the estimate and standard error of a specified distribution parameter and has the following signature:

- *Type:* Subroutine
- *Number of arguments:* 3
- *Sequence and type of arguments:*
 - index* specifies the numeric value of the index of the distribution parameter for which you want the information. The value of *index* must be in the interval $[1, m]$, where m is the number of parameters in the distribution to which this subroutine belongs.
 - est* specifies the output argument that returns the estimate of the requested parameter.
 - stderr* specifies the output argument that returns the standard error of the requested parameter.
- *Return value:* Estimate and standard error of the requested distribution parameter that are returned in the output arguments *est* and *stderr*, respectively

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then a subroutine named SEV_PARMEST is created in the package of each distribution. Here is a sample structure of the code that PROC SEVSELECT uses to define the subroutine:

```
subroutine SEV_PARMEST(index, est, stderr);
  outargs est, stderr;
  est = <value of the estimate for the distribution parameter
        at position 'index'>;
  stderr = <value of the standard error for distribution parameter
           at position 'index'>;
endsub;
```

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution *dist*, the subroutine named SEV_PARMEST_dist is created in the *sevfit* package. SEV_PARMEST_dist has the same structure as the SEV_PARMEST subroutine that is described previously.

If you use the SCALEMODEL statement to specify a scale regression model, and if you specify only singleton continuous effects, then for *index*=1, the returned estimates are of θ_0 (the base value of the scale parameter) or $\log(\theta_0)$ if the distribution has a log-scale parameter. For more information about θ_0 , see the section “[Estimating Regression Effects](#)” on page 1794.

SEV_PARMNAME | SEV_PARMNAME_dist

is a function that returns the name of a specified distribution parameter and has the following signature:

- *Type:* Function
- *Number of arguments:* 1
- *Sequence and type of arguments:*
 - index* specifies the numeric value of the index of the distribution parameter for which you want the information. The value of *index* must be in the interval $[1, m]$, where m is the number of parameters in the distribution to which this function belongs.

- *Return value:* Character value that contains the name of the distribution parameter that appears at the position *index* in the distribution's definition

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then a function named SEV_PARMNAME is created in the package of each distribution.

Here is a sample structure of the code that PROC SEVSELECT uses to define the function:

```
function SEV_PARMNAME(index) $32;
    name = <name of the distribution parameter at position 'index'>;
    return (name);
endsub;
```

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution *dist*, a function named SEV_PARMNAME_*dist* is created in the *sevfit* package. SEV_PARMNAME_*dist* has the same structure as the SEV_PARMNAME function that is described previously.

If you use the SCALEMODEL statement to specify a scale regression model, and if you specify only singleton continuous effects, then the following helper functions and subroutines are also created in the OUTLIB= library.

SEV_NUMREG

is a function that returns the number of regressors and has the following signature:

- *Type:* Function
- *Number of arguments:* 0
- *Sequence and type of arguments:* Not applicable
- *Return value:* Numeric value that contains the number of regressors that you specify in the SCALEMODEL statement. If you specify an OFFSET= option variable in the SCALEMODEL statement, then the returned value is equal to 1 plus the number of regressors that you specify in the SCALEMODEL statement.

Here is a sample structure of the code that PROC SEVSELECT uses to define the function:

```
function SEV_NUMREG();
    m = <number of regressors>;
    if (<offset variable is specified>) then m = m + 1;
    return (m);
endsub;
```

This function does not depend on any distribution, so it is always created in the *sevfit* package.

SEV_REGEST | SEV_REGEST_*dist*

is a subroutine that returns the estimate and standard error of a specified regression parameter and has the following signature:

- *Type:* Subroutine
- *Number of arguments:* 3
- *Sequence and type of arguments:*

index specifies the numeric value of the index of the regression parameter for which you want the information. The value of *index* must be in the interval $[1, K]$, where K is the number of regressors that is returned by the SEV_NUMREG function. If you specify an OFFSET= option variable in the SCALEMODEL statement, then an *index* value of K corresponds to the offset variable.

est specifies the output argument that returns the estimate of the requested regression parameter.

stderr specifies the output argument that returns the standard error of the requested regression parameter.

- *Return value:* Estimate and standard error of the requested regression parameter that are returned in the output arguments *est* and *stderr*, respectively

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then a subroutine named SEV_REGEST is created in the package of each distribution. Here is a sample structure of the code that PROC SEVSELECT uses to define the subroutine:

```
subroutine SEV_REGEST(index, est, stderr);
  outargs est, stderr;
  est = <value of the estimate for the regression parameter
        at position 'index'>;
  stderr = <value of the standard error for regression parameter
           at position 'index'>;
endsub;
```

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution *dist*, the subroutine named SEV_REGEST_*dist* is created in the *sevfit* package. SEV_REGEST_*dist* has the same structure as the SEV_REGEST subroutine that is described previously.

If the regressor that corresponds to the specified *index* value is a redundant regressor, the returned values of both *est* and *stderr* are equal to the special missing value of .R. If you specify an OFFSET= option variable in the SCALEMODEL statement and if the *index* value corresponds to the offset variable—that is, it is equal to the value that the SEV_NUMREG function returns—then the returned value of *est* is equal to 1 and the returned value of *stderr* is equal to the special missing value of .F.

SEV_REGNAME

is a function that returns the name of a specified regressor and has the following signature:

- *Type:* Function
- *Number of arguments:* 1
- *Sequence and type of arguments:*

index specifies the numeric value of the index of the regressor for which you want the name. The value of *index* must be in the interval $[1, K]$, where K is the number of regressors that is returned by the SEV_NUMREG function. If you specify an OFFSET= option variable in the SCALEMODEL statement, then an *index* value of K corresponds to the offset variable.

- *Return value:* Character value that contains the name of the regressor that appears at the position *index* in the SCALEMODEL statement. If you specify an OFFSET= option variable in the SCALEMODEL statement, then for an *index* value of *K*, the returned value contains the name of the offset variable.

Here is a sample structure of the code that PROC SEVSELECT uses to define the function:

```
function SEV_REGNAME(index) $32;
    name = <name of regressor at position 'index'>;
    return (name);
endsub;
```

This function does not depend on any distribution, so it is always created in the *sevfit* package.

Custom Objective Functions

You can use a series of programming statements that use variables in the DATA= data table to assign a value to an objective function symbol. You must specify the objective function symbol by using the **OBJECTIVE=** option in the PROC SEVSELECT statement.

The objective function can be programmed such that it applies to any distribution that is used in the model. For that purpose, PROC SEVSELECT recognizes the following *keyword* functions in the programming statements:

<code>_PDF_(x)</code>	evaluates the probability density function (PDF) of a distribution at the current value of a data table variable <i>x</i> .
<code>_CDF_(x)</code>	evaluates the cumulative distribution function (CDF) of a distribution at the current value of a data table variable <i>x</i> .
<code>_SDF_(x)</code>	evaluates the survival distribution function (SDF) of a distribution at the current value of a data table variable <i>x</i> .
<code>_LOGPDF_(x)</code>	evaluates the natural logarithm of the PDF of a distribution at the current value of a data table variable <i>x</i> .
<code>_LOGCDF_(x)</code>	evaluates the natural logarithm of the CDF of a distribution at the current value of a data table variable <i>x</i> .
<code>_LOGSDF_(x)</code>	evaluates the natural logarithm of the SDF of a distribution at the current value of a data table variable <i>x</i> .
<code>_EDF_(x)</code>	evaluates the empirical distribution function (EDF) estimate at the current value of a data table variable <i>x</i> . Internally, PROC SEVSELECT computes the estimate by using the SVRTUTIL_EDF function as described in the section “ Predefined Utility Functions ” on page 1824. The EDF estimate that the SVRTUTIL_EDF function requires is computed by using a subset of observations in the current BY group or a subset of the input data table if you do not specify the BY statement. The subset of observations is randomly chosen on each worker node by using the values that you specify in the INITSAMPLE option.

EMPLIMMOMENT(k, u)

evaluates the empirical limited moment of order k at the current value of a data table variable u that represents the upper limit of the limited moment. The order k can also be a data table variable. Internally, PROC SEVSELECT computes the moment by using the SVRTUTIL_EMPLIMMOMENT function as described in the section “[Predefined Utility Functions](#)” on page 1824. The EDF estimate that the SVRTUTIL_EMPLIMMOMENT function requires is computed by using a subset of observations in the current BY group or a subset of the input data table if you do not specify the BY statement. The subset of observations is randomly chosen on each worker node by using the values that you specify in the [INITSAMPLE](#) option.

LIMMOMENT(k, u)

evaluates the limited moment of order k at the current value of a data table variable u that represents the upper limit of the limited moment. The order k can be a data table variable or a constant. Internally, for each candidate distribution, PROC SEVSELECT computes the moment by using the LIMMOMENT function as described in the section “[Predefined Utility Functions](#)” on page 1824.

All the preceding functions are right-hand-side functions. They act as placeholders for distribution-specific functions, with the exception of `_EDF_` and `_EMPLIMMOMENT_` functions.

As an example, suppose that the data table `mylib.Test` contains a response variable Y and a left-truncation threshold variable T . The following statements use the values in this data table to fit a model with distribution D such that the parameters of the model minimize the value of the objective function symbol `MYOBJ`:

```
options cmplib=(work.mydist);
proc sevselect data=mylib.test objective=myobj;
  loss y / lt=t;

  myobj = -_LOGPDF_(y);
  if (not(missing(t))) then
    myobj = myobj + log(1-_CDF_(t));

  dist d;
run;
```

You designate the symbol `MYOBJ` as an objective function symbol by using the [OBJECTIVE=](#) option in the PROC SEVSELECT statement. The response variable Y and left-truncation variable T are specified in the LOSS statement. The distribution D is specified in the DIST statement. The remaining statements constitute a program that computes the value of the `MYOBJ` symbol.

Let the distribution D have the parameters $P1$ and $P2$. To estimate the model for this distribution, PROC SEVSELECT internally converts the generic program to the following program specific to distribution D :

```
myobj = -D_LOGPDF(y, p1, p2);
if (not(missing(t))) then
  myobj = myobj + log(1-D_CDF(t, p1, p2));
```

Note that the generic keyword functions `_LOGPDF_` and `_CDF_` have been replaced by the distribution-specific functions `D_LOGPDF` and `D_CDF`, respectively, and the argument list of each is expanded to include the distribution parameters. You must have defined the `D_LOGPDF` and `D_CDF` functions previously; they are assumed to be available in the library `Work.Mydist` that you specify in the `CMPLIB=` option.

The program is executed for each observation in the data table `mylib.Test` to compute the value of `MYOBJ` by using the values of variables `Y` and `T` in that observation and internally computed values of the model parameters `P1` and `P2`. The values of `MYOBJ` are then added over all the observations of the data table or over all the observations of the current `BY` group if you specify the `BY` statement. The resulting aggregate value is the value of the objective function, and it is supplied to the optimizer. If the optimizer requires derivatives of the objective function, then `PROC SEVSELECT` automatically differentiates `MYOBJ` with respect to the parameters `P1` and `P2`. The optimizer iterates through various combinations of the values of `P1` and `P2`; in each iteration, it computes a new value of the objective function and the needed derivatives of it, until it finds a combination that minimizes the objective function.

Note the following points when you define your own program to compute the custom objective function:

- The value of the objective function is always minimized by `PROC SEVSELECT`. If you want to maximize the value of a certain objective, then add a statement that assigns the negated value of the maximization objective to the objective function symbol that you specify in the `OBJECTIVE=` option. Minimization of the negated objective is equivalent to maximization of the original objective.
- The contributions of individual observations are always added together to compute the overall objective function in a given iteration of the optimizer. If you specify the `WEIGHT` statement, then the contribution of each observation is weighted by multiplying it by the normalized value of the weight variable for that observation.
- If you are fitting multiple distributions in one `PROC SEVSELECT` step and you use any of the keyword functions in your program, then it is recommended that you not explicitly use the parameters of any of the specified distributions in your programming statements.
- If you use a specific keyword function in your programming statements, then the corresponding distribution functions must be defined in a library that you specify in the `CMPLIB=` system option or in `Sashelp.Svrtlist`, the predefined functions library. In the preceding example, it is assumed that the functions `D_LOGPDF` and `D_CDF` are defined in the library `Work.Mydist`, which you specify in the `CMPLIB=` option.
- You can use most `DATA` step statements and functions in your program. The `DATA` step file and the data set I/O statements (for example, `INPUT`, `FILE`, `SET`, and `MERGE`) are not available. However, some `PUT` statement functionality is supported. For more information, see the section “`PROC FCMP` and `DATA` Step Differences” in *Base SAS Procedures Guide*. In addition to the differences that are listed in that section, the following differences exist:
 - You can use only numeric-valued variables in `PROC SEVSELECT` programming statements. This restriction also implies that you cannot use SAS functions or call routines that require character-valued arguments, unless you pass those arguments as constant (literal) strings or characters.
 - You cannot use functions that create lagged versions of a variable in `PROC SEVSELECT` programming statements. If you need lagged versions, then you can use a `DATA` step before the `PROC SEVSELECT` step to add those versions to the input data table.
- When you are writing your programming statements, avoid defining variables that begin with an underscore (`_`), because they might conflict with internal variables that `PROC SEVSELECT` creates.

Custom Objective Functions and Regression Effects

If you specify regression effects by using the SCALEMODEL statement, then PROC SEVSELECT automatically adds a statement before your programming statements to compute the value of the scale parameter or the log-transformed scale parameter of the distribution by using the values of the regression variables and internally created regression parameters. For example, if your specification of the SCALEMODEL statement results in three regression effects, x_1 , x_2 , and x_3 , then for a model that contains the distribution D with scale parameter S, PROC SEVSELECT adds to the beginning of your program a statement that is equivalent to the following statement:

$$S = _SEVTHETA0 * \exp(_SEVBETA1 * x_1 + _SEVBETA2 * x_2 + _SEVBETA3 * x_3);$$

If a model contains a distribution D1 with a log-transformed scale parameter M, PROC SEVSELECT adds to the beginning of your program a statement that is equivalent to the following statement:

$$M = _SEVTHETA0 + _SEVBETA1 * x_1 + _SEVBETA2 * x_2 + _SEVBETA3 * x_3;$$

The internal symbols $_SEVTHETA0$, $_SEVBETA1$, $_SEVBETA2$, and $_SEVBETA3$ represent the regression parameters that are associated with the intercept and the regression effects x_1 , x_2 , and x_3 , respectively.

Because the names of the internal regression parameter symbols start with the prefix $_SEV$, if you use a variable in your program whose name begins with $_SEV$, then PROC SEVSELECT writes an error message to the SAS log and stops processing.

Input Data Tables

PROC SEVSELECT accepts DATA= and INEST= data tables as input data tables. This section details the information that they are expected to contain.

DATA= Data Table

The DATA= data table is expected to contain the values of the analysis variables that you specify in the CLASS, EFFECT, LOSS, and SCALEMODEL statements.

If you specify the BY statement, then the DATA= data table must contain all the BY variables that you specify in the BY statement. However, it is not required to be sorted by the BY variables.

INEST= Data Table

The INEST= data table is expected to contain the initial values of the parameters for the parameter estimation process.

If you specify the BY statement, then the INEST= data table must contain all the BY variables that you specify in the BY statement. The data table is not required to be sorted by BY variables, and it is not required to contain all the BY groups present in the DATA= data table. For the BY groups that are not present in the INEST= data table, the default parameter initialization method is used.

The INEST= data table must be created by using the OUTEST= option in a PROC SEVSELECT step that uses the same BY and SCALEMODEL statements as the PROC SEVSELECT step where you specify the INEST= option.

Output Data Tables

PROC SEVSELECT produces the OUTEST=, OUTMODELINFO=, and OUTSTAT= data tables when requested by their respective options in the PROC SEVSELECT statement. It also produces the OUT= data table when you specify the OUTPUT statement. The data tables and their contents are described in the following sections.

OUT= Data Table

The OUT= data table that you specify in the OUTPUT statement records the estimates of the scoring functions and quantiles that you specify in the OUTPUT statement.

For each distribution that you specify in the DIST statement, the OUT= data table contains one variable for each scoring function that you specify in the FUNCTIONS= option and one variable for each quantile that you specify in the QUANTILES= option. The prefix of the variable's name is <distribution-name>_, whereas the suffix of the variable's name is determined by the information that you specify in the respective option or by the default method that PROC SEVSELECT uses. For more information about variable names, see the description of the OUTPUT statement.

The OUT= data table also contains the variables that you specify in the COPYVARS= option. If you specify the BY statement and if you want PROC SEVSELECT to copy the BY variables from the DATA= data table to the OUT= data table, then you must specify them in the COPYVARS= option.

The number of observations in the OUT= data table depends on the options that you specify in the OUTPUT statement and whether or not you specify the SCALEMODEL statement.

If either of the following conditions is met, then the number of observations in the OUT= data table is equal to the number of observations in the DATA= data table:

- You specify the SCALEMODEL statement.
- You specify the FUNCTIONS= option in the OUTPUT statement such that at least one scoring function does not have a constant, nonmissing argument.

If neither of the preceding conditions is met, then the number of observations in the OUT= data table is equal to the number of BY groups, which is equal to 1 if you do not specify the BY statement.

OUTEST= Data Table

The OUTEST= data table records the estimates of the model parameters. It also contains estimates of their standard errors and optionally their covariance structure. If you specify BY variables, then the data are organized in BY groups and the data table contains variables that you specify in the BY statement.

If you do not specify the COVOUT option, then the data table contains the following variables:

<code>_MODEL_</code>	identifying name of the distribution model. The observation contains information about this distribution.
<code>_TYPE_</code>	type of the estimates reported in this observation. It can take one of the following two values:

EST point estimates of model parameters
 STDERR standard error estimates of model parameters

`_STATUS_` status of the reported estimates. The possible values are listed in the section “[_STATUS_ Variable Values](#)” on page 1844.

`<Parameter 1> ... <Parameter M>`

M variables, named after the parameters of all candidate distributions, that contain estimates of the respective parameters. M is the cardinality of the union of parameter name sets from all candidate distributions. In an observation, estimates are populated only for parameters that correspond to the distribution that is indicated by the `_MODEL_` variable. If `_TYPE_` is EST, then the estimates are missing if the model does not converge. If `_TYPE_` is STDERR, then the estimates are missing if covariance estimates cannot be obtained.

If you specify regression effects, then the estimate that is reported for the first parameter of each distribution is the estimate of the base value of the scale or log-transformed scale parameter. For more information, see the section “[Estimating Regression Effects](#)” on page 1794.

`<Regression Parameter 1> ... <Regression Parameter K>`

If your effect specification in the [SCALEMODEL](#) statement results in K regression parameters and if you do not specify the [SELECTOUT](#) option, then the OUTEST= data table contains K regression parameter variables.

The name of each variable is formed by using the name of the effect and the levels of the CLASS variables that the effect might contain. If the effect name or level names are too long, then the variable name is constructed by using a part of the effect name and parts of the levels of the CLASS variables. The label of the variable is more descriptive than the name of the variable. The variables contain different types of estimates for their respective regression parameters.

If `_TYPE_` is EST, then a variable contains the estimate of the regression coefficient (β_j) of the respective regression parameter, unless one of the following conditions applies:

- If the model does not converge, then it contains an ordinary missing value.
- If the regression parameter is linearly dependent on other regression parameters for a particular BY group and
 - if you specify the [ZEROEST](#) option, then it contains a value of 0.
 - if you do not specify the [ZEROEST](#) option, then it contains the special missing value .R.
- If you specify the SELECTION statement and if the regression parameter is not part of the selected model for a particular BY group, then
 - if you specify the [ZEROEST](#) option, then it contains a value of 0.
 - if you do not specify the [ZEROEST](#) option, then it contains the special missing value .N.

If `_TYPE_` is STDERR, then a variable contains the estimate of the standard error of the respective regression parameter, unless one of the following conditions applies:

- If covariance estimates cannot be obtained, then it contains an ordinary missing value.
- If the regression parameter is linearly dependent on other regression parameters for a particular BY group, then it contains the special missing value .R.
- If you specify the SELECTION statement and if the regression parameter is not part of the selected model for a particular BY group, then it contains the special missing value .N.

<Offset Variable>

If you specify an OFFSET= variable in the SCALEMODEL statement, then the OUTEST= data table contains a variable that is named after the offset variable. If _TYPE_ is EST, then the value of this variable is 1. If _TYPE_ is STDERR, then the value of this variable is the special missing value .F.

If you specify the COVOUT option in the PROC SEVSELECT statement, then the OUTEST= data table contains additional observations that contain the estimates of the covariance structure. In addition to the variables listed and described previously, the data table contains the following variables that are either new or have a modified description:

<u>_TYPE_</u>	type of the estimates reported in this observation. For observations that contain rows of the covariance structure, the value is COV.
<u>_STATUS_</u>	status of the reported estimates. For observations that contain rows of the covariance structure, the status is 0 if covariance estimation was successful. If estimation fails, the status is 1 and a single observation is reported with _TYPE_=COV and missing values for all the parameter variables.
<u>_NAME_</u>	name of the parameter for the row of covariance matrix that is reported in the current observation.

If _TYPE_ is COV, then a variable contains the estimate of the covariance between the respective parameter and the parameter that is named in the _NAME_ variable, unless one of the following conditions applies:

- If covariance estimates cannot be obtained, then it contains an ordinary missing value.
- If the variable corresponds to a regression parameter and
 - if the regression parameter is linearly dependent on other regression parameters for a particular BY group, then it contains the special missing value .R.
 - if you specify the SELECTION statement and if the regression parameter is not in the selected model for a particular BY group, then it contains the special missing value .N.

If you specify the SELECTION statement and the SELECTOUT option, then the OUTEST= data table contains only the regression parameters that correspond to the final selected effects. For the selected parameters, the values are written according to the rules stated previously for _TYPE_ values of EST, STDERR, and COV.

If you specify the BY and CLASS statements and if the names and values of some CLASS variables are long, then it is possible for a regression parameter with the same name to have different labels in different BY groups. In such cases, the OUTEST= data table contains rows with _TYPE_='COV' and an additional

variable called `_LABEL_`. You can use the values of `_NAME_` and `_LABEL_` variables in such rows to obtain the label for multilabel regression parameters in each BY group. When the `OUTEST=` data table does not contain the `_LABEL_` variable, then the label of a regression parameter is the label of the variable that contains the estimate of that regression parameter.

OUTMODELINFO= Data Table

The `OUTMODELINFO=` data table records the information about each candidate distribution that you specify in the `DIST` statement. It contains the following variables:

<code>_MODEL_</code>	identifying name of the distribution model. The observation contains information about this distribution.
<code>_DEPVAR_</code>	name of the loss variable.
<code>_DESCRIPTION_</code>	descriptive name of the model. This has a nonmissing value only if the <code>DESCRIPTION</code> function has been defined for this model.
<code>_VALID_</code>	validity of the distribution definition. This has a value of 1 for valid definitions and a value of 0 for invalid definitions. If the definition is invalid, then <code>PROC SEVSELECT</code> writes the reason for invalidity to the SAS log.
<code>_PARAMNAME1 ... _PARAMNAMEM</code>	M variables that contain names of parameters of the distribution model, where M is the maximum number of parameters across all the specified distribution models. For a given distribution with m parameters, values of the variables <code>_PARAMNAMEj</code> ($j > m$) are missing.

OUTSTAT= Data Table

The `OUTSTAT=` data table records statistics of fit and model selection information. If you specify BY variables, then the data are organized in BY groups and the data table contains variables that you specify in the BY statement. The data table contains the following variables:

<code>_MODEL_</code>	identifying name of the distribution model. The observation contains information about this distribution.
<code>_NMODELPARAM_</code>	number of parameters in the distribution.
<code>_NESTPARAM_</code>	number of estimated parameters. This includes the regression parameters, if you specify any regression effects.
<code>_NOBS_</code>	number of nonmissing observations that are used for parameter estimation.
<code>_STATUS_</code>	status of the parameter estimation process for this model. The possible values are listed in the section “ _STATUS_ Variable Values ” on page 1844.
<code>_SELECTED_</code>	indicator of the best distribution model. If the value is 1, then this model is the best model for the current BY group according to the specified model selection criterion.
<Custom Objective>	value of the custom objective function that is obtained at the end of the parameter estimation process. <code>PROC SEVSELECT</code> creates this variable only if you specify the <code>OBJECTIVE=</code> option.

Neg2LogLike	value of the log likelihood, multiplied by -2 , that is obtained at the end of the parameter estimation process.
AIC	value of Akaike's information criterion (AIC) that is obtained at the end of the parameter estimation process.
AICC	value of the corrected Akaike's information criterion (AICC) that is obtained at the end of the parameter estimation process.
BIC	value of the Schwarz Bayesian information criterion (SBC) that is obtained at the end of the parameter estimation process.
KS	value of the Kolmogorov-Smirnov (KS) statistic that is obtained at the end of the parameter estimation process.
AD	value of the Anderson-Darling (AD) statistic that is obtained at the end of the parameter estimation process.
CVM	value of the Cramér-von Mises (CvM) statistic that is obtained at the end of the parameter estimation process.

The values of the variables that correspond to the fit statistics and the `_SELECTED_` variable are missing if the parameter estimation process does not converge for the model in the `_MODEL_` variable.

`_STATUS_` Variable Values

The `_STATUS_` variable in the `OUTEST=` and `OUTSTAT=` data tables contains a value that indicates the status of the parameter estimation process for the respective distribution model. The variable can take the following values in the `OUTEST=` data table for `_TYPE_=EST` observations and in the `OUTSTAT=` data table:

- 0 The parameter estimation process converged for this model.
- 301 The parameter estimation process might not have converged for this model because there is no improvement in the objective function value. This might indicate that the initial values of the parameters are optimal, or you can try different convergence criteria in the `NLOPTIONS` statement.
- 302 The parameter estimation process might not have converged for this model because the number of iterations exceeded the maximum allowed value. You can try setting a larger value for the `MAXITER=` options in the `NLOPTIONS` statement.
- 303 The parameter estimation process might not have converged for this model because the number of objective function evaluations exceeded the maximum allowed value. You can try setting a larger value for the `MAXFUNC=` options in the `NLOPTIONS` statement.
- 304 The parameter estimation process might not have converged for this model because the time taken by the process exceeded the maximum allowed value. You can try setting a larger value for the `MAXTIME=` option in the `NLOPTIONS` statement.
- 400 The parameter estimation process did not converge for this model.

The `_STATUS_` variable can take the following values in the `OUTEST=` data table for `_TYPE_=STDERR` and `_TYPE_=COV` observations:

- 0 The covariance and standard error estimates are available and valid.
- 1 The covariance and standard error estimates are not available, because the process of computing covariance estimates failed.

Displayed Output

The SEVSELECT procedure optionally produces displayed output by using the Output Delivery System (ODS). All output is controlled by the PRINT= option in the PROC SEVSELECT statement. [Table 26.7](#) relates the ODS tables to PRINT= options.

Table 26.7 ODS Tables Produced in PROC SEVSELECT

ODS Table Name	Description	Option
AllFitStatistics	Statistics of fit for all the distribution models	PRINT=ALLFITSTATS
BSplineDetails	Basis details for B-spline effects	PRINT=ALL and EFFECT SPLINE statement
ClassLevels	Level information for classification variables	Default for CLASS statement
CollectionLevelInfo	Level details for collection effects	PRINT=ALL and EFFECT COLLECTION statement
ConvergenceStatus	Convergence status of parameter estimation process	PRINT=CONVSTATUS
CorrB	Correlations of parameter estimates	PRINT=CORRB
CovB	Covariances of parameter estimates	PRINT=COVB
DescStats	Descriptive statistics for the response variable	PRINT=DESCSTATS
Distribution	Information about number of parameters in a distribution's model	Default
DistributionInfo	Detailed information about all distributions	PRINT=DISTINFO
EntryCandidates	Details about candidates for entry into the model	PRINT=ALL and SELECTION statement
EstimationDetails	Details of the estimation process for all the distribution models	PRINT=ESTIMATIONDETAILS
FitStatistics	Statistics of fit	PRINT=STATISTICS
InitialValues	Initial parameter values and bounds	PRINT=INITIALVALUES
IterationHistory	Optimization iteration history	PRINT=NLOHISTORY
MMLevelInfo	Level information for multimember effects	PRINT=ALL and EFFECT MULTIMEMBER statement
ModelSelection	Model selection summary	PRINT=SELECTION

Table 26.7 *continued*

ODS Table Name	Description	Option
OptimizationSummary	Optimization summary	PRINT=NLOSUMMARY
ParameterEstimates	Final parameter estimates	PRINT=ESTIMATES
PolyDetails	Number of variables and columns, polynomial degree, and standardization method	PRINT=ALL and EFFECT POLY statement
PolyScaling	Centering and scaling details	PRINT=ALL and EFFECT POLY statement
RegDescStats	Descriptive statistics for the regression effects that do not contain a CLASS variable	PRINT=DESCSTATS
RemovalCandidates	Details about candidates for removal from the model	PRINT=ALL and SELECTION statement
SelectedEffects	List of effects selected for the model	Default for SELECTION statement
SelectionEstimationDetails	Details of the estimation process for all selection steps of a distribution's model	PRINT=ESTIMATIONDETAILS and SELECTION statement
SelectionInfo	Information about the settings for model selection	Default for SELECTION statement
SelectionReason	Reason why the particular model was selected	Default for SELECTION statement
SelectionSummary	Summary information about model selection steps	Default for SELECTION statement
SplineKnots	Knot and boundary knot values	PRINT=ALL and EFFECT SPLINE statement
StopReason	Reason for termination of model selection	Default for SELECTION statement
TPFSplineDetails	Truncated power function (TPF) spline basis details	PRINT=ALL and EFFECT SPLINE statement
Timing	Timing information for various computational stages of the procedure	PRINT=ALL

If you do not specify the PRINT= option, then by default PROC SEVSELECT produces the ModelSelection, ConvergenceStatus, OptimizationSummary, FitStatistics, and ParameterEstimates ODS tables.

If you specify the CLASS statement and if you do not specify the **NOCLPRINT** option, then by default PROC SEVSELECT produces the ClassLevels table.

If you specify the SELECTION statement, then by default PROC SEVSELECT produces the SelectedEffects, SelectionInfo, SelectionReason, SelectionSummary, and StopReason ODS tables.

The following list presents further details of the contents of some of the tables:

AllFitStatistics (PRINT=ALLFITSTATS)

displays the comparison of all the statistics of fit for all the models in one table. The table does not include the models whose parameter estimation process does not converge. If no models converge, then this table is not produced. If the table contains more than one model, then the best model according to each statistic is indicated by an asterisk (*) in that statistic's column.

CorrB (PRINT=CORRB)

displays the estimated correlation matrix of parameters of the fitted model. You can control the display of this matrix by specifying the SHAPE= and NOUNITDIAG suboptions of the PRINT([SYMMETRIC=](#)) option. The estimates are not displayed for models whose parameter estimation process does not converge.

CovB (PRINT=COVB)

displays the estimated variance-covariance matrix of parameters of the fitted model. You can control the display of this matrix by specifying the SHAPE= suboption of the PRINT([SYMMETRIC=](#)) option. The estimates are not displayed for models whose parameter estimation process does not converge.

DistributionInfo (PRINT=DISTINFO)

displays the information about all the candidate distribution. It includes the name, the description, the number of distribution parameters, and whether the distribution is valid for the specified modeling task.

FitStatistics (PRINT=STATISTICS)

displays the statistics of fit for each model. The statistics of fit are not displayed for models whose parameter estimation process does not converge.

ModelSelection (PRINT=SELECTION)

displays the model selection table. The table shows the convergence status of each candidate model and the value of the selection criterion along with an indication of the selected model.

ParameterEstimates (PRINT=ESTIMATES)

displays the estimates of parameters of the fitted model. The estimates are not displayed for models whose parameter estimation process does not converge.

ODS Graphics

Statistical procedures use ODS Graphics to create graphs as part of their output. ODS Graphics is described in detail in the “Statistical Graphics Using ODS” chapter in *SAS/STAT User's Guide*.

Before you create graphs, ODS Graphics must be enabled (for example, by using the ODS GRAPHICS ON statement). For more information about enabling and disabling ODS Graphics, see the section “Enabling and Disabling ODS Graphics” in that chapter.

The overall appearance of graphs is controlled by ODS styles. Styles and other aspects of using ODS Graphics are discussed in the section “A Primer on ODS Statistical Graphics” in that chapter.

This section describes how the SEVSELECT procedure uses ODS to create graphics.

ODS Graph Names

PROC SEVSELECT assigns a name to each graph that it creates by using ODS. You can use these names to selectively reference the graphs. The names are listed in [Table 26.8](#).

Table 26.8 ODS Graphics Produced by PROC SEVSELECT

ODS Graph Name	Plot Description	PLOTS= Option
Comparative Plots		
CDFPlot	Comparative CDF plot	CDF
PDFPlot	Comparative PDF plot	PDF
ConditionalPDFPlot	Comparative conditional PDF plot	CONDITIONALPDF
Per Distribution Plots		
CDFDistPlot	CDF plot per distribution	CDFPERDIST
PDFDistPlot	PDF plot per distribution	PDFPERDIST
ConditionalPDFDistPlot	Conditional PDF plot per distribution	CONDITIONALPDFPERDIST
PPPlot	P-P plot of CDF and EDF	PP
QQPlot	Q-Q plot	QQ

Comparative CDF Plot

The comparative CDF plot helps you visually compare the cumulative distribution function (CDF) estimates of all the candidate distribution models to the empirical distribution function (EDF) estimate. The plot does not contain CDF estimates for models whose parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the CDF or EDF estimates.

If you specify truncation, then conditional CDF estimates are plotted. Otherwise, unconditional CDF estimates are plotted. The conditional estimates are computed by using the method that is described in the section “[Truncation and Conditional CDF Estimates](#)” on page 1790.

If you specify regression effects, then the plotted CDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

Comparative PDF Plot

The comparative PDF plot helps you visually compare the probability density function (PDF) estimates of all the candidate distribution models. The plot does not contain PDF estimates for models whose parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the PDF estimates.

If you specify the HISTOGRAM option, then the plot also contains the histogram of response variable values. If you specify the KERNEL option, then the plot also contains the kernel density estimate of the response variable values.

If you specify regression effects, then the plotted PDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

Comparative Conditional PDF Plot

The comparative conditional PDF plot helps you visually compare the conditional probability density function (PDF) estimates of all the candidate distribution models in up to three ranges of values of the response variable. The plot does not contain conditional PDF estimates for models whose parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the conditional PDF estimates.

For a region that has values in the interval $(L, R]$, PROC SEVSELECT computes the conditional PDF estimate, $\hat{f}^c$, for a distribution as $\hat{f}^c(y) = \hat{f}(y)/(\hat{F}(R) - \hat{F}(L))$, where $\hat{f}$ and $\hat{F}$ denote the estimated unconditional PDF and CDF, respectively, of the distribution. You can control the regions and the values of L and R for each region by specifying the **LEFTQ=**, **RIGHTQ=**, and **SHOWREGION=** suboptions of the **PLOT=CONDITIONALPDF** option.

If you specify the **HISTOGRAM** option, then the plot also contains the histogram of response variable values in each region.

If you specify regression effects, then the plotted conditional PDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

CDF Plot per Distribution

The CDF plot per distribution shows the CDF estimates of each candidate distribution model unless that model's parameter estimation process does not converge. The plot also contains estimates of the EDF. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the CDF or EDF estimates.

This plot shows the lower and upper pointwise confidence limits for the EDF estimates. For an EDF estimate F_n with standard error σ_n , they are computed as $\max(0, F_n - z_{(1-\alpha/2)}\sigma_n)$ and $\min(1, F_n + z_{(1-\alpha/2)}\sigma_n)$, respectively, where z_p is the p th quantile from the standard normal distribution and α denotes the confidence level that you specify in the **EDFALPHA=** option (the default is $\alpha = 0.05$).

If you specify truncation, then conditional CDF estimates are plotted. Otherwise, unconditional CDF estimates are plotted. The conditional estimates are computed by using the method that is described in the section “[Truncation and Conditional CDF Estimates](#)” on page 1790.

If you specify regression effects, then the plotted CDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

PDF Plot per Distribution

The PDF plot per distribution shows the PDF estimates of each candidate distribution model unless that model's parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the PDF estimates.

If you specify the **HISTOGRAM** option, then the plot also contains the histogram of response variable values. If you specify the **KERNEL** option, then the plot also contains the kernel density estimate of the response variable values.

If you specify regression effects, then the plotted PDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

Conditional PDF Plot per Distribution

The conditional PDF plot per distribution shows the conditional PDF estimates of each candidate distribution model unless that model's parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the conditional PDF estimates.

For a region that has values in the interval $(L, R]$, PROC SEVSELECT computes the conditional PDF estimate, $\hat{f}^c$, for a distribution as $\hat{f}^c(y) = \hat{f}(y)/(\hat{F}(R) - \hat{F}(L))$, where $\hat{f}$ and $\hat{F}$ denote the estimated unconditional PDF and CDF, respectively, of the distribution. You can control the regions and the values of L and R for each region by specifying the **LEFTQ=**, **RIGHTQ=**, and **SHOWREGION=** suboptions of the **PLOT=CONDITIONALPDFPERDIST** option.

If you specify the **HISTOGRAM** option, then the plot also contains the histogram of response variable values in each region.

If you specify regression effects, then the plotted PDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

P-P Plot of CDF and EDF

The P-P plot of CDF and EDF is the probability-probability plot that compares the CDF estimates of a distribution to the EDF estimates. A plot is not prepared for models whose parameter estimation process does not converge. The horizontal axis represents the CDF estimates of a candidate distribution, and the vertical axis represents the EDF estimates.

This plot can be interpreted as displaying the data that are used for computing the EDF-based statistics of fit for the given candidate distribution. As described in the section “[EDF-Based Statistics](#)” on page 1807, these statistics are computed by comparing the EDF, denoted by $F_n(y)$, to the CDF, denoted by $F(y)$, at each of the response variable values y . Using the probability inverse transform $z = F(y)$, this is equivalent to comparing the EDF of z , denoted by $F_n(z)$, to the CDF of z , denoted by $F(z)$ (D’Agostino and Stephens 1986, Ch. 4). Because the CDF of z is a uniform distribution ($F(z) = z$), the EDF-based statistics can be computed by comparing the estimate of the EDF of z to the estimate of z itself, which is the estimate of the CDF. The horizontal axis of the plot represents $\hat{z} = \hat{F}(y)$. The vertical axis represents the estimated EDF of z , $\hat{F}_n(z)$. The plot contains a scatter plot of $(\hat{z}, \hat{F}_n(z))$ points and a reference line, $F_n(z) = z$, that represents the expected uniform distribution of z . Points that are scattered closer to the reference line indicate a better fit than points that are scattered farther away from the reference line.

If you specify truncation, then the EDF estimates are conditional, as described in the section “[EDF Estimates and Truncation](#)” on page 1804. So conditional estimates of CDF are displayed; these estimates are computed by using the method that is described in the section “[Truncation and Conditional CDF Estimates](#)” on page 1790.

If you specify regression effects, then the displayed CDF estimates, both unconditional and conditional, are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

Q-Q Plot

The Q-Q plot is a quantile-quantile scatter plot that compares the empirical quantiles to the quantiles from a candidate distribution. A plot is not prepared for models whose parameter estimation process does not converge. The horizontal axis represents the quantiles from a candidate distribution, and the vertical axis represents the empirical quantiles.

Each point in the plot corresponds to a specific value of the EDF estimate, F_n . The Y coordinate is the value of the response variable for which F_n is computed. The X coordinate is computed by using one of the two following methods for a candidate distribution named *dist*:

- If you have defined the *dist_QUANTILE* function that satisfies the requirements listed in the section “[dist_QUANTILE](#)” on page 1819, then that function is invoked by using F_n and estimated distribution parameters as arguments. The QUANTILE function is defined in the Sashelp.Svrtldist library for all the predefined distributions.
- If the *dist_QUANTILE* function is not defined, then PROC SEVSELECT numerically inverts the *dist_CDF* function at the CDF value of F_n for the estimated distribution parameters. If the *dist_CDF* function is not defined, then the *exp(dist_LOGCDF)* function is inverted. If the inversion fails, the corresponding point is not plotted in the Q-Q plot.

If you specify truncation, then the EDF estimates are conditional, as described in the section “[EDF Estimates and Truncation](#)” on page 1804. The CDF inversion process, whether done numerically or by evaluating the *dist_QUANTILE* function, needs to accept an unconditional CDF value. So the F_n value is first transformed to an unconditional estimate F_n^u as

$$F_n^u = F_n \cdot (\hat{F}(t_{\max}^r) - \hat{F}(t_{\min}^l)) + \hat{F}(t_{\min}^l)$$

where $\hat{F}(t_{\max}^r)$ and $\hat{F}(t_{\min}^l)$ are as defined in the section “[Truncation and Conditional CDF Estimates](#)” on page 1790.

If you specify regression effects, then the value of the first distribution parameter is determined by using the method that is equivalent to specifying the DFMIXTURE=MEAN option in the SCALEMODEL statement. This is described in the section “[CDF and PDF Estimates with Regression Effects](#)” on page 1797.

Examples: SEVSELECT Procedure

Example 26.1: Defining a Model for the Gaussian Distribution with a Scale Parameter

If you want to estimate the influence of regression effects, then the model needs to be parameterized to have a scale parameter. Although this might not be always possible, it is possible for certain distributions. This example illustrates it for the Gaussian distribution. To obtain a parameterization with scale parameter, you replace the location parameter μ with another parameter, $\alpha = \mu/\sigma$, and define the PDF (f) and the CDF (F) as follows:

$$f(x; \sigma, \alpha) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{x}{\sigma} - \alpha\right)^2\right)$$

$$F(x; \sigma, \alpha) = \frac{1}{2} \left(1 + \operatorname{erf}\left(\frac{1}{\sqrt{2}} \left(\frac{x}{\sigma} - \alpha\right)\right)\right)$$

Then, you can verify that σ is the scale parameter, because both of the following equalities are true:

$$f(x; \sigma, \alpha) = \frac{1}{\sigma} f\left(\frac{x}{\sigma}; 1, \alpha\right)$$

$$F(x; \sigma, \alpha) = F\left(\frac{x}{\sigma}; 1, \alpha\right)$$

NOTE: The Gaussian distribution is not a commonly used severity distribution. It is used in this example primarily to illustrate the concept of parameterizing a distribution such that it has a scale parameter. Although the distribution has a support over the entire real line, you can fit the distribution with PROC SEVSELECT only if the input sample contains nonnegative values.

The following statements use the alternate parameterization to define a new model named NORMAL_S. The definition is stored in the library Work.Sevexmpl.

```
/*----- Define normal distribution with scale parameter -----*/
proc fcmp library=sashelp.svrtldist outlib=work.sevexmpl.models;
  function normal_s_pdf(x, Sigma, Alpha);
    /* Sigma : Scale & Standard Deviation */
    /* Alpha : Scaled mean */
    return ( exp(-(x/Sigma - Alpha)**2/2) /
              (Sigma * sqrt(2*constant('PI')))) );
  endsub;

  function normal_s_cdf(x, Sigma, Alpha);
    /* Sigma : Scale & Standard Deviation */
    /* Alpha : Scaled mean */
    z = x/Sigma - Alpha;
    return (0.5 + 0.5*erf(z/sqrt(2)));
  endsub;

  subroutine normal_s_parinit(dim, x[*], nx[*], F[*], Ftype, Sigma, Alpha);
```

```

    outargs Sigma, Alpha;
    array m[2] / nosymbols;
    /* Compute estimates by using method of moments */
    call svrtutil_rawmoments(dim, x, nx, 2, m);
    Sigma = sqrt(m[2] - m[1]**2);
    Alpha = m[1]/Sigma;
endsub;

subroutine normal_s_lowerbounds(Sigma, Alpha);
    outargs Sigma, Alpha;
    Alpha = .; /* Alpha has no lower bound */
    Sigma = 0; /* Sigma > 0 */
endsub;
quit;

```

An important point to note is that the scale parameter *Sigma* is the first distribution parameter (after the 'x' argument) listed in the signatures of NORMAL_S_PDF and NORMAL_S_CDF functions. *Sigma* is also the first distribution parameter listed in the signatures of other subroutines. This is required by PROC SEVSELECT, so that it can identify which is the scale parameter. When you specify regression effects, PROC SEVSELECT checks whether the first parameter of each candidate distribution is a scale parameter (or a log-transformed scale parameter if [dist_SCALETRANSFORM](#) subroutine is defined for the distribution with LOG as the transform). If it is not, then an appropriate message is written the SAS log and that distribution is not fitted.

Let the following DATA step statements simulate a sample from the normal distribution where the parameter σ is affected by the regressors as follows:

$$\sigma = \exp(1 + 0.5 X_1 + 0.75 X_3 - 2 X_4 + X_5)$$

The sample is simulated such that the regressor X_2 is linearly dependent on regressors X_1 and X_3 .

```

/*--- Simulate a normal distribution's sample affected by regressors ---*/
data testnorm_reg(keep=y x1-x5 Sigma);
    array x{*} x1-x5;
    array b{6} _TEMPORARY_ (1 0.5 . 0.75 -2 1);
    call streaminit(34567);
    label y='Normal Response Influenced by Regressors';

    do n = 1 to 100;
        /* simulate regressors */
        do i = 1 to dim(x);
            x(i) = rand('UNIFORM');
        end;
        /* make x2 linearly dependent on x1 */
        x(2) = 5 * x(1);

        /* compute log of the scale parameter */
        logSigma = b(1);
        do i = 1 to dim(x);
            if (i ne 2) then
                logSigma = logSigma + b(i+1) * x(i);
        end;

        Sigma = exp(logSigma);
    end;

```

```

        y = rand('NORMAL', 25, Sigma);
        output;
    end;
run;

```

You can use a DATA step as follows to load the data set `Work.Test_reg` into a data table in your session that is associated with the `mylib` libref. The DATA step assumes that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```

data mylib.testnorm_reg;
    set testnorm_reg;
run;

```

The following statements use PROC SEVSELECT to fit the NORMAL_S distribution model along with some of the predefined distributions to the simulated sample:

```

/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);

/*----- Fit models with PROC SEVSELECT -----*/
proc sevselect data=mylib.testnorm_reg print=all;
    loss y;
    scalemodel x1-x5;
    dist Normal_s burr logn pareto weibull;
run;

```

The “Model Selection” table in [Output 26.1.1](#) indicates that all the models, except the Burr distribution model, have converged. Also, only three models, Normal_s, Burr, and Weibull, seem to have a good fit for the data. The table that compares all the fit statistics indicates that Normal_s model is the best according to the likelihood-based statistics; however, the Burr model is the best according to the EDF-based statistics.

Output 26.1.1 Summary of Results for Fitting the Normal Distribution with Regressors

The SEVSELECT Procedure

Model Selection							
		-2 Log					
Distribution	Converged	Likelihood		Selected			
Normal_s	Yes	603.95786		Yes			
Burr	Maybe	612.81685		No			
Logn	Yes	749.20125		No			
Pareto	Yes	841.07022		No			
Weibull	Yes	612.77496		No			

All Fit Statistics							
		-2 Log					
Distribution	Likelihood	AIC	AICC	SBC	KS	AD	CvM
Normal_s	603.95786*	615.95786*	616.86108*	631.58888*	1.52388	4.00152	0.70769
Burr	612.81685	626.81685	628.03424	645.05304	1.50448*	3.90072*	0.63399*
Logn	749.20125	761.20125	762.10448	776.83227	2.88110	16.20558	3.04825
Pareto	841.07022	853.07022	853.97345	868.70124	4.83810	31.60568	6.84046
Weibull	612.77496	624.77496	625.67819	640.40598	1.50490	3.90559	0.63458

Asterisk (*) denotes the best model in the column.

This prompts you to further evaluate why the model with Burr distribution has not converged. The initial values, convergence status, and the optimization summary for the Burr distribution are shown in [Output 26.1.2](#). The initial values table indicates that the regressor X2 is redundant, which is expected. More importantly, the convergence status indicates that it requires more than 50 iterations. PROC SEVSELECT enables you to change several settings of the optimizer by using the [NLOPTIONS](#) statement. In this case, you can increase the limit of 50 on the iterations, change the convergence criterion, or change the technique to something other than the default trust-region technique.

Output 26.1.2 Details of the Fitted Burr Distribution Model

The SEVSELECT Procedure

Burr Distribution

Model Information	
Distribution	Burr
Description	Burr Distribution (Type XII Family)
Distribution Parameters	3
Regression Parameters	4

Initial Parameter Values and Bounds			
Parameter	Initial Value	Lower Bound	Upper Bound
Theta	25.75198	1.05367E-8	Infy
Alpha	2.00000	1.05367E-8	Infy
Gamma	2.00000	1.05367E-8	Infy
x1	0.07345	-709.78271	709.78271
x2	Redundant	-709.78271	709.78271
x3	-0.14056	-709.78271	709.78271
x4	0.27064	-709.78271	709.78271
x5	-0.23230	-709.78271	709.78271

Convergence Status
Needs more than 50 iterations.

Optimization Summary	
Optimization Technique	Trust Region
Iterations	50
Function Calls	137
Log Likelihood	-306.4084238

The following PROC SEVSELECT step uses the NLOPTIONS statement to change the convergence criterion and the limits on the iterations and function evaluations, exclude the lognormal and Pareto distributions that have been confirmed previously to fit the data poorly, and exclude the redundant regressor X2 from the model:

```
/*--- Refit and compare models with higher limit on iterations ---*/
proc sevselect data=mylib.testnorm_reg print=all;
  loss y;
  scalemodel x1 x3-x5;
  dist Normal_s burr weibull;
  nloptions absfconv=2.0e-5 maxiter=100 maxfunc=500;
run;
```

The results shown in [Output 26.1.3](#) indicate that the Burr distribution has now converged. The NORMAL_S distribution is still the best distribution according to the likelihood-based criteria.

Output 26.1.3 Summary of Results after Changing Maximum Number of Iterations

The SEVSELECT Procedure

Model Selection							
Distribution	Converged	-2 Log Likelihood		Selected			
Normal_s	Yes	603.95786	Yes				
Burr	Yes	612.79276	No				
Weibull	Yes	612.77496	No				

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Normal_s	603.95786*	615.95786*	616.86108*	631.58888*	1.52388	4.00152	0.70769
Burr	612.79276	626.79276	628.01015	645.02895	1.50472*	3.90351*	0.63433*
Weibull	612.77496	624.77496	625.67819	640.40598	1.50490	3.90559	0.63458

Asterisk (*) denotes the best model in the column.

Example 26.2: Defining a Model for Mixed-Tail Distributions

In some applications, a few severity values tend to be extreme as compared to the typical values. The extreme values represent the worst-case scenarios and cannot be discarded as outliers. Instead, their distribution must be modeled to prepare for their occurrences. In such cases, it is often useful to fit one distribution to the non-extreme values and another distribution to the extreme values. The *mixed-tail* distribution mixes two distributions: one for the *body* region, which contains the non-extreme values, and another for the *tail* region, which contains the extreme values. The tail distribution is usually a generalized Pareto distribution (GPD), because it is usually good for modeling the conditional excess severity above a threshold. The body distribution can be any distribution. The following definitions are used in describing a generic formulation of a mixed-tail distribution:

$g(x)$	PDF of the body distribution
$G(x)$	CDF of the body distribution
$h(x)$	PDF of the tail distribution
$H(x)$	CDF of the tail distribution
θ	scale parameter for the body distribution
Ω	set of nonscale parameters for the body distribution
ξ	shape parameter for the GPD tail distribution
x_r	normalized value of the response variable at which the tail starts
p_n	mixing probability

Given these notations, the PDF $f(x)$ and the CDF $F(x)$ of the mixed-tail distribution are defined as

$$f(x) = \begin{cases} \frac{p_n}{G(x_b)} g(x) & \text{if } x \leq x_b \\ (1 - p_n) h(x - x_b) & \text{if } x > x_b \end{cases}$$

$$F(x) = \begin{cases} \frac{p_n}{G(x_b)} G(x) & \text{if } x \leq x_b \\ p_n + (1 - p_n) H(x - x_b) & \text{if } x > x_b \end{cases}$$

where $x_b = \theta x_r$ is the value of the response variable at which the tail starts.

These definitions indicate the following:

- The body distribution is conditional on $X \leq x_b$, where X denotes the random response variable.
- The tail distribution is the generalized Pareto distribution of the $(X - x_b)$ values.
- The probability that a response variable value belongs to the body is p_n . Consequently the probability that the value belongs to the tail is $(1 - p_n)$.

The parameters of this distribution are θ , Ω , ξ , x_r , and p_n . The scale of the GPD tail distribution θ_t is computed as

$$\theta_t = \frac{G(x_b; \theta, \Omega) (1 - p_n)}{g(x_b; \theta, \Omega) p_n} = \theta \frac{G(x_r; \theta = 1, \Omega) (1 - p_n)}{g(x_r; \theta = 1, \Omega) p_n}$$

The parameter x_r is usually initialized using a tail index estimation algorithm. One such algorithm is Hill's algorithm (Danielsson et al. 2001), which is implemented by the predefined utility function SVRTUTIL_HILLCUTOFF available to you in the Sashelp.Svrtldist library. The algorithm and the utility function are described in detail in the section “[Predefined Utility Functions](#)” on page 1824. The function computes an estimate of x_b , which can be used to compute an initial estimate of x_r as $x_r = x_b / \hat{\theta}$, where $\hat{\theta}$ is the estimate of the scale parameter of the body distribution.

The parameter p_n is usually determined by the domain expert based on the fraction of losses that are expected to belong to the tail.

The following SAS statements define the LOGNGPD distribution model for a mixed-tail distribution with the lognormal distribution as the body distribution and GPD as the tail distribution:

```
/*----- Define lognormal Body-GPD tail mixed distribution -----*/
proc fcmp library=sashelp.svrtldist outlib=work.sevexmpl.models;
  function LOGNGPD_DESCRIPTION() $256;
    length desc $256;
    desc1 = "Lognormal Body-GPD Tail Distribution";
    desc2 = " (Mu, Sigma, Xi, and Xr are free parameters;";
    desc3 = " Pn is fixed at 0.8)";
    desc = desc1 || desc2 || desc3;
    return(desc);
  endsub;

  function LOGNGPD_SCALETRANSFORM() $3;
    length xform $3;
    xform = "LOG";
    return (xform);
```

```

endsub;

subroutine LOGNGPD_CONSTANTPARM(Pn);
endsub;

function LOGNGPD_PDF(x, Mu, Sigma, Xi, Xr, Pn);
  cutoff = exp(Mu) * Xr;
  p = CDF('LOGN', cutoff, Mu, Sigma);
  if (x < cutoff + constant('MACEPS')) then do;
    return ((Pn/p)*PDF('LOGN', x, Mu, Sigma));
  end;
  else do;
    gpd_scale = p*((1-Pn)/Pn)/PDF('LOGN', cutoff, Mu, Sigma);
    h = (1+Xi*(x-cutoff)/gpd_scale)**(-1-(1/Xi))/gpd_scale;
    return ((1-Pn)*h);
  end;
endsub;

function LOGNGPD_CDF(x, Mu, Sigma, Xi, Xr, Pn);
  cutoff = exp(Mu) * Xr;
  p = CDF('LOGN', cutoff, Mu, Sigma);
  if (x < cutoff + constant('MACEPS')) then do;
    return ((Pn/p)*CDF('LOGN', x, Mu, Sigma));
  end;
  else do;
    gpd_scale = p*((1-Pn)/Pn)/PDF('LOGN', cutoff, Mu, Sigma);
    H = 1 - (1 + Xi*((x-cutoff)/gpd_scale))**(-1/Xi);
    return (Pn + (1-Pn)*H);
  end;
endsub;

subroutine LOGNGPD_PARMINIT(dim, x[*], nx[*], F[*], Ftype,
                          Mu, Sigma, Xi, Xr, Pn);
  outargs Mu, Sigma, Xi, Xr, Pn;
  array xe[1] / nosymbols;
  array nxe[1] / nosymbols;

  eps = constant('MACEPS');

  Pn = 0.8; /* Set mixing probability */
  _status_ = .;
  call streaminit(56789);
  Xb = svrtutil_hillcutoff(dim, x, 100, 25, _status_);
  if (missing(_status_) or _status_ = 1) then
    Xb = svrtutil_percentile(Pn, dim, x, F, Ftype);

  /* Initialize lognormal parameters */
  call logn_parminit(dim, x, nx, F, Ftype, Mu, Sigma);
  if (not(missing(Mu))) then
    Xr = Xb/exp(Mu);
  else
    Xr = .;

  /* Prepare arrays for excess values */

```

```

i = 1;
do while (i <= dim and x[i] < Xb+eps);
    i = i + 1;
end;
dime = dim-i+1;
if (dime > 0) then do;
    call dynamic_array(xe, dime);
    call dynamic_array(nxe, dime);
    j = 1;
    do while(i <= dim);
        xe[j] = x[i] - Xb;
        nxe[j] = nx[i];
        i = i + 1;
        j = j + 1;
    end;

    /* Initialize GPD's shape parameter using excess values */
    call gpd_parminit(dime, xe, nxe, F, Ftype, theta_gpd, Xi);
end;
else do;
    Xi = .;
end;
endsub;

subroutine LOGNGPD_LOWERBOUNDS (Mu, Sigma, Xi, Xr, Pn);
    outargs Mu, Sigma, Xi, Xr, Pn;

    Mu    = .; /* Mu has no lower bound */
    Sigma = 0; /* Sigma > 0 */
    Xi    = 0; /* Xi > 0 */
    Xr    = 0; /* Xr > 0 */
endsub;
quit;

```

Note the following points about the LOGNGPD definition:

- In this example, the parameter p_n is not estimated with the maximum likelihood method used by PROC SEVSELECT, so you need to specify it as a *constant* parameter by defining the [dist_CONSTANTPARM](#) subroutine. The signature of the LOGNGPD_CONSTANTPARM subroutine lists only the constant parameter P_n .
- The LOGNGPD_PARMINIT subroutine initializes the parameter x_r by first using the SVRTUTIL_HILLCUTOFF utility function to compute an estimate of the cutoff point $\hat{x}_b$ and then computing $x_r = \hat{x}_b / e^{\hat{\mu}}$. If SVRTUTIL_HILLCUTOFF fails to compute a valid estimate, then the SVRTUTIL_PERCENTILE utility function is used to set $\hat{x}_b$ to the p_n th percentile of the data. The parameter p_n is fixed to 0.8.
- The Sashelp.SvrtDIST library is specified with the LIBRARY= option in the PROC FCMP statement to enable the LOGNGPD_PARMINIT subroutine to use the predefined utility functions (SVRTUTIL_HILLCUTOFF and SVRTUTIL_PERCENTILE) and parameter initialization subroutines (LOGN_PARMINIT and GPD_PARMINIT).
- The LOGNGPD_LOWERBOUNDS subroutine defines the lower bounds for all parameters. This subroutine is required because the parameter μ has a non-default lower bound. The bounds for σ

and X_i must be specified. If they are not specified, they are returned as missing values, which PROC SEVSELECT interprets as having no lower bound. You do not need to specify any bounds for the constant parameter P_n , because it is not subject to optimization.

The following DATA step statements simulate a sample from a mixed-tail distribution with a lognormal body and GPD tail. The parameter p_n is fixed to 0.8, the same value used in the LOGNGPD_PARMINIT subroutine defined previously.

```
/*----- Simulate a sample for the mixed-tail distribution -----*/
data testmixdist(keep=y label='Lognormal Body-GPD Tail Sample');
  call streaminit(45678);
  label y='Response Variable';
  N = 1000;
  Mu = 1.5;
  Sigma = 0.25;
  Xi = 0.7;
  Pn = 0.8;

  /* Generate data for the lognormal body */
  Nbody = N*Pn;
  do i=1 to Nbody;
    y = exp(Mu) * rand('LOGNORMAL')**Sigma;
    output;
  end;

  /* Generate data for the GPD tail */
  cutoff = quantile('LOGNORMAL', Pn, Mu, Sigma);
  gpd_scale = (1-Pn) / pdf('LOGNORMAL', cutoff, Mu, Sigma);
  do i=Nbody+1 to N;
    y = cutoff + ((1-rand('UNIFORM'))**(-Xi) - 1)*gpd_scale/Xi;
    output;
  end;
run;
```

You can use a DATA step as follows to load the data set Work.Testmixdist into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.testmixdist;
  set testmixdist;
run;
```

The following statements use PROC SEVSELECT to fit the LOGNGPD distribution model to the simulated sample. They also fit three other predefined distributions (BURR, LOGN, and GPD). The final parameter estimates are written to the mylib.Parmest data table.

```
/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);

/*----- Fit LOGNGPD model with PROC SEVSELECT -----*/
proc sevselect data=mylib.testmixdist print=all outest=mylib.parmest;
  loss y;
  dist logngpd burr logn gpd;
run;
```

Some of the results that PROC SEVSELECT produces are shown in [Output 26.2.1](#) and [Output 26.2.2](#). The “Model Selection” table in [Output 26.2.1](#) indicates that all models converged. The “All Fit Statistics” table in [Output 26.2.1](#) shows that the model with LOGNGPD distribution has the best fit according to all the fit statistics. The Burr distribution model is the closest contender to the LOGNGPD model, but the GPD distribution model fits the data very poorly.

Output 26.2.1 Summary of Fitting Mixed-Tail Distribution

The SEVSELECT Procedure

Model Selection							
Distribution	Converged	-2 Log Likelihood		Selected			
Logngpd	Yes	3640	Yes				
Burr	Yes	3687	No				
Logn	Yes	3862	No				
Gpd	Yes	5344	No				

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logngpd	3640*	3650*	3650*	3674*	1.22054*	1.12053*	0.21314*
Burr	3687	3693	3693	3708	1.33323	2.34704	0.39000
Logn	3862	3866	3866	3875	2.20231	7.31780	0.94769
Gpd	5344	5348	5348	5358	12.27970	218.30354	44.54186

Asterisk (*) denotes the best model in the column.

The detailed results for the LOGNGPD distribution are shown in [Output 26.2.2](#). The initial values table shows the fixed value of the P_n parameter that the LOGNGPD_PARMINIT subroutine sets. The table uses the bounds columns to indicate that it is a constant parameter. The last table in the figure shows the final parameter estimates. The estimates of all free parameters are significantly different from 0. As expected, the final estimate of the constant parameter P_n has not changed from its initial value.

Output 26.2.2 Detailed Results for the LOGNGPD Distribution

The SEVSELECT Procedure

Logngpd Distribution

Model Information	
Distribution	Logngpd
Description	Lognormal Body-GPD Tail Distribution (Mu, Sigma, Xi, and Xr are free parameters; Pn is fixed at 0.8)
Distribution Parameters	5

Output 26.2.2 *continued*

Initial Parameter Values and Bounds					
Parameter	Initial Value	Lower Bound	Upper Bound		
Mu	1.14149	-Infy	Infy		
Sigma	1.03316	1.05367E-8	Infy		
Xi	0.48188	1.05367E-8	Infy		
Xr	1.62621	1.05367E-8	Infy		
Pn	0.80000	Constant	Constant		

Convergence Status	
Convergence criterion (GCONV=1E-8) satisfied.	

Optimization Summary	
Optimization Technique	Trust Region
Iterations	26
Function Calls	81
Log Likelihood	-1819.786323

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	1.61374	0.02509	64.31	<.0001
Sigma	1	0.31716	0.01541	20.59	<.0001
Xi	1	0.53177	0.08867	6.00	<.0001
Xr	1	1.19816	0.03925	30.52	<.0001
Pn	1	0.80000	Constant	.	.

The following SAS statements use the parameter estimates to compute the value where the tail region is estimated to start ($x_b = e^{\hat{\mu}} \hat{x}_r$) and the scale of the GPD tail distribution ($\theta_t = \frac{G(x_b)}{g(x_b)} \frac{(1-p_n)}{p_n}$):

```

/*----- Compute tail cutoff and tail distribution's scale -----*/
data xb_thetat(keep=x_b theta_t);
  set mylib.parmest(where=(_MODEL_='Logngpd' and _TYPE_='EST'));
  x_b = exp(Mu) * Xr;
  theta_t = (CDF('LOGN', x_b, Mu, Sigma) / PDF('LOGN', x_b, Mu, Sigma)) *
            ((1-Pn) / Pn);
run;

proc print data=xb_thetat noobs;
run;

```

Output 26.2.3 Start of the Tail and Scale of the GPD Tail Distribution

x_b	theta_t
6.01665	1.00677

The computed values of x_b and θ_t are shown as `x_b` and `theta_t` in [Output 26.2.3](#). Equipped with this additional derived information, you can now interpret the results of fitting the mixed-tail distribution as follows:

- The tail starts at $y \approx 6.02$. Optimizing the scale-normalized relative cutoff (x_r) in addition to optimizing the scale of the body region ($\theta = e^\mu$) gives you more flexibility in optimizing the absolute cutoff (x_b). If X_r is declared as a constant parameter, then x_b is optimized by virtue of optimizing the scale of the body region ($\theta = e^\mu$), and you must rely on Hill's tail index estimator to yield an initial estimate of x_b that is close to an optimal estimate. By keeping X_r as a free parameter, you account for the possibility that Hill's estimator can yield a suboptimal estimate.
- The values $y \leq 6.02$ follow the lognormal distribution with parameters $\mu \approx 1.614$ and $\sigma \approx 0.317$. These parameter estimates are reasonably close to the parameters of the body distribution that is used for simulating the sample.
- If X_t denotes the loss random variable for the tail defined as $X_t = X - x_b$, where X is the original loss variable, then for this example, $\Pr[X_t = X - 6.02 | X_t > 0]$ follows the GPD density function with scale $\theta_t \approx 1.005$ and shape $\xi \approx 0.539$.

Example 26.3: Fitting a Scaled Tweedie Model with Regressors

The Tweedie distribution is often used in the insurance industry to explain the influence of regression effects on the distribution of losses. PROC SEVSELECT provides a predefined scaled Tweedie distribution (STWEEDIE) that enables you to model the influence of regression effects on the scale parameter. The scale regression model has its own advantages such as the ability to easily account for inflation effects. This example illustrates how that model can be used to evaluate the influence of regression effects on the *mean* of the Tweedie distribution, which is useful in problems such rate-making and pure premium modeling.

Assume a Tweedie process, whose mean μ is affected by k regression effects x_j , $j = 1, \dots, k$, as follows,

$$\mu = \mu_0 \exp \left(\sum_{j=1}^k \beta_j x_j \right)$$

where μ_0 represents the base value of the mean (you can think of μ_0 as $\exp(\beta_0)$, where β_0 is the intercept). This model for the mean is identical to the popular generalized linear model for the mean with a logarithmic link function.

More interestingly, it parallels the model used by PROC SEVSELECT for the scale parameter θ ,

$$\theta = \theta_0 \exp \left(\sum_{j=1}^k \beta_j x_j \right)$$

where θ_0 represents the base value of the scale parameter. As described in the section “[Tweedie Distributions](#)” on page 1781, for the parameter range $p \in (1, 2)$, the mean of the Tweedie distribution is given by

$$\mu = \theta \lambda \frac{2-p}{p-1}$$

where λ is the Poisson mean parameter of the scaled Tweedie distribution. This relationship enables you to use the scale regression model to infer the influence of regression effects on the mean of the distribution.

Let the data set `Work.Test_Sevtw` contain a sample generated from a Tweedie distribution with dispersion parameter $\phi = 0.5$, index parameter $p = 1.75$, and the mean parameter that is affected by three regression variables x_1 , x_2 , and x_3 as follows:

$$\mu = 5 \exp(0.25 x_1 - x_2 + 3 x_3)$$

Thus, the population values of regression parameters are $\mu_0 = 5$, $\beta_1 = 0.25$, $\beta_2 = -1$, and $\beta_3 = 3$.

The following DATA step generates a sample of the Tweedie distribution:

```
/*--- Simulate a Tweedie sample that is affected by regressors ---*/
%let samplesize=300;
data test_sevtw(keep=y x1-x3
                 label='A Tweedie Sample Affected by Regressors');
  array x{*} x1-x3;
  array b{4} _TEMPORARY_ (5 0.25 -1 3);
  call streaminit(45678);
  label y='Response Influenced by Regressors';

  Phi = 0.5;
  P = 1.75;
  do n = 1 to &samplesize;
    Mu = 0;
    do i = 1 to dim(x);
      x(i) = rand('UNIFORM');
      Mu = Mu + b(i+1) * x(i);
    end;
    Mu = b(1) * exp(Mu); /* b(1) is base value of mean */

    cdf = rand('UNIFORM');
    y = quantile('TWEEDIE', cdf, P, Mu, Phi);
    output;
  end;
run;
```

You can use a DATA step as follows to load the data set `Work.Test_sevtw` into a data table in your session that is associated with the `mylib` libref. The DATA step assumes that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
data mylib.test_sevtw;
  set test_sevtw;
run;
```

The following PROC SEVSELECT step uses the sample in `mylib.Test_Sevtw` data table to estimate the parameters of the scale regression model for the predefined scaled Tweedie distribution (STWEEDIE) with the dual quasi-Newton (QUANEW) optimization technique:

```
/*--- Fit the scale parameter version of the Tweedie distribution ---*/
proc sevselect data=mylib.test_sevtw outest=mylib.estw covout print=all;
  loss y;
  scalemodel x1-x3;

  dist stweedie;
  nloptions tech=quanew;
run;
```

The dual quasi-Newton technique is used because it requires only the first-order derivatives of the objective function, and it is harder to compute reasonably accurate estimates of the second-order derivatives of Tweedie distribution's PDF with respect to the parameters.

Some of the key results that PROC SEVSELECT prepares are shown in [Output 26.3.1](#) and [Output 26.3.2](#). The model information and the convergence results are shown in [Output 26.3.1](#).

Output 26.3.1 Convergence Results for the STWEEDIE Model with Regressors

The SEVSELECT Procedure

Stweedie Distribution

Model Information	
Distribution	Stweedie
Description	Tweedie Distribution with Scale Parameter
Distribution Parameters	3
Regression Parameters	3

Convergence Status	
Convergence criterion (FCONV=2.220446E-16) satisfied.	

Optimization Summary	
Optimization Technique	Dual Quasi-Newton
Iterations	42
Function Calls	137
Log Likelihood	-1044.321485

The final parameter estimates of the STWEEDIE regression model are shown in [Output 26.3.2](#). The estimate that is reported for the parameter Theta is the estimate of the base value θ_0 . The estimates of regression coefficients β_1 , β_2 , and β_3 are indicated by the rows of x1, x2, and x3, respectively.

Output 26.3.2 Parameter Estimates for the STWEEDIE Model with Regressors

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	0.82442	0.28548	2.89	0.0042
Lambda	1	16.31209	13.75533	1.19	0.2366
P	1	1.75037	0.21870	8.00	<.0001
x1	1	0.27941	0.09877	2.83	0.0050
x2	1	-0.76724	0.10315	-7.44	<.0001
x3	1	3.03195	0.10142	29.89	<.0001

If your goal is to explain the influence of regression effects on the scale parameter, then the output displayed in [Output 26.3.2](#) is sufficient. But, if you want to compute the influence of regression effects on the mean of the distribution, then you need to do some postprocessing. Using the relationship between μ and θ , μ can be

written in terms of the parameters of the STWEEDIE model as

$$\mu = \theta_0 \exp \left(\sum_{j=1}^k \beta_j x_j \right) \lambda \frac{2-p}{p-1}$$

This shows that the parameters β_j are identical for the mean and the scale model, and the base value μ_0 of the mean model is

$$\mu_0 = \theta_0 \lambda \frac{2-p}{p-1}$$

The estimate of μ_0 and the standard error associated with it can be computed by using the property of the functions of maximum likelihood estimators (MLE). If $g(\Omega)$ represents a totally differentiable function of parameters Ω , then the MLE of g has an asymptotic normal distribution with mean $g(\hat{\Omega})$ and covariance $C = (\partial g)' \Sigma (\partial g)$, where $\hat{\Omega}$ is the MLE of Ω , Σ is the estimate of covariance matrix of Ω , and ∂g is the gradient vector of g with respect to Ω evaluated at $\hat{\Omega}$. For μ_0 , the function is $g(\Omega) = \theta_0 \lambda (2-p)/(p-1)$. The gradient vector is

$$\begin{aligned} \partial g &= \left(\frac{\partial g}{\partial \theta_0} \quad \frac{\partial g}{\partial \lambda} \quad \frac{\partial g}{\partial p} \quad \frac{\partial g}{\partial \beta_1} \cdots \frac{\partial g}{\partial \beta_k} \right) \\ &= \left(\frac{\mu_0}{\theta_0} \quad \frac{\mu_0}{\lambda} \quad \frac{-\mu_0}{(p-1)(2-p)} \quad 0 \dots 0 \right) \end{aligned}$$

You can write a DATA step that implements these computations by using the parameter and covariance estimates that the PROC SEVSELECT step prepares. The following statements show one such implementation:

```
/*--- Compute the estimate, standard error, and p-value of the mean ---*/
data Mu0(keep=Parameter Estimate Stderr T probT);
  Parameter='Mu0';
  label Estimate='Estimate' Stderr='Standard;Error'
        T='t Value' probT='Approx;Pr > |t|';

  array cov{3,3} _temporary_;
  array parms{3} _temporary_;
  array parmrow{3} theta lambda p;
  set mylib.estw(where=( _MODEL_='Stweedie' and
                        (_type_='COV' or _type_='EST'))) end=eof;

  n = _n_-1;
  if (n = 0) then do;
    do i=1 to 3;
      parms[i] = parmrow[i];
    end;
  end;
  else do;
    do i=1 to n;
      cov[n,i] = parmrow[i];
    end;
  end;
  if (eof or n = 3) then do;
```

```

mu0 = parms[1]*parms[2]*(2-parms[3])/(parms[3]-1);
Estimate = mu0;

a = mu0/parms[1]; /* dMu0_T0 */
b = mu0/parms[2]; /* dMu0_L */
c = -mu0/((parms[3]-1)*(2-parms[3])); /* dMu0_P */

varMu0 = a * (a*cov[1,1]+b*cov[2,1]+c*cov[3,1]) +
          b * (a*cov[2,1]+b*cov[2,2]+c*cov[3,2]) +
          c * (a*cov[3,1]+b*cov[3,2]+c*cov[3,3]);

Stderr = sqrt(varMu0);

df = &samplesize-6;
T = mu0/Stderr;
probtT = (1-probt(T, df))*2;
output;
stop;
end;
run;

proc print data=Mu0 noobs s='';
run;

```

The estimates of μ_0 that the preceding DATA step prepares are shown in [Output 26.3.3](#). These estimates and the estimates of β_j as shown in [Output 26.3.2](#) are reasonably close (that is, within one or two standard errors) to the parameters of the population from which the sample in mylib.Test_Sevtw data table was drawn.

Output 26.3.3 Estimate of the Base Value Mu0 of the Mean Parameter

Parameter	Estimate	Standard Error	t Value	Approx Pr > t
Mu0	4.47382	0.42251	10.5886	0

Another outcome of using the scaled Tweedie distribution to model the influence of regression effects is that the regression effects also influence the variance V of the Tweedie distribution. The variance is related to the mean as $V = \phi\mu^p$, where ϕ is the dispersion parameter. Using the relationship between the parameters TWEEDIE and STWEEDIE distributions as described in the section “[Tweedie Distributions](#)” on page 1781, the regression model for the dispersion parameter is

$$\begin{aligned}
 \log(\phi) &= (2 - p) \log(\mu) - \log(\lambda(2 - p)) \\
 &= ((2 - p) \log(\mu_0) - \log(\lambda(2 - p))) + (2 - p) \sum_{j=1}^k \beta_j x_j
 \end{aligned}$$

Subsequently, the regression model for the variance is

$$\begin{aligned}
 \log(V) &= 2 \log(\mu) - \log(\lambda(2 - p)) \\
 &= (2 \log(\mu_0) - \log(\lambda(2 - p))) + 2 \sum_{j=1}^k \beta_j x_j
 \end{aligned}$$

In summary, PROC SEVSELECT enables you to estimate regression effects on various parameters and statistics of the Tweedie model.

Example 26.4: Fitting Distributions to Interval-Censored Data

In some applications, the data available for modeling might not be exact. A commonly encountered scenario is the use of grouped data from an external agency, which for several reasons, including privacy, does not provide information about individual loss events. The losses are grouped into disjoint bins, and you know only the range and number of values in each bin. Each group is essentially interval-censored, because you know that a loss magnitude is in certain interval, but you do not know the exact magnitude. This example illustrates how you can use PROC SEVSELECT to model such data.

The following DATA step generates sample grouped data for dental insurance claims, which is taken from Klugman, Panjer, and Willmot (1998):

```
/* Grouped dental insurance claims data (Klugman, Panjer, and Willmot 1998) */
data gdental;
  input lowerbd upperbd count @@;
  datalines;
0 25 30 25 50 31 50 100 57 100 150 42 150 250 65 250 500 84
500 1000 45 1000 1500 10 1500 2500 11 2500 4000 3
;
run;
```

To analyze small data sets such as this, you might want to restrict the number of workers that are used by PROC SEVSELECT by submitting the following statements:

```
cas mysess terminate;
cas mysess sessopts=(nworkers=1);
libname mylib cas sessref=mysess;
```

The first CAS statement terminates the current CAS session named mysess. It also clears the mylib libref that was associated with the session. The second CAS statement creates another session that uses only one worker of the server. The LIBNAME statement associates the mylib libref with the new session. For more information about the CAS statement and the CAS LIBNAME engine, see [SAS Cloud Analytic Services: User's Guide](#).

If you want to load a SAS data set into a data table on the CAS server, then it is recommended that you restrict the number of workers before you load the data set. If you restrict the number of workers after you load the data, then PROC SEVSELECT needs to redistribute the data table so that it resides on the restricted number of workers; this can cause unnecessary deterioration of performance.

You can use a DATA step as follows to load the data set Work.Gdental into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.gdental;
  set gdental;
run;
```

The following PROC SEVSELECT step fits all the predefined distributions to the data in the mylib.Gdental data table:

```

/* Fit all predefined distributions */
proc sevselect data=mylib.gdental edf=turnbull print=all criterion=aicc;
  loss / rc=lowerbd lc=upperbd;
  weight count;
  dist _predef_;
run;

```

The EDF= option in the PROC SEVSELECT statement specifies that the Turnbull's method be used for EDF estimation. The LOSS statement specifies the left and right boundaries of each group as the right-censoring and left-censoring limits, respectively. The variable count records the number of losses in each group and is specified in the WEIGHT statement. Note that no response variable is specified in the LOSS statement, which is allowed as long as each observation in the input data table is censored.

Some of the key results that PROC SEVSELECT produces are shown in [Output 26.4.1](#). According to the “Model Selection” table in [Output 26.4.1](#), all distribution models have converged. The “All Fit Statistics” table in [Output 26.4.1](#) indicates that the exponential distribution (EXP) has the best fit for data according to a majority of the likelihood-based statistics and that the Burr distribution (BURR) has the best fit according to all the EDF-based statistics.

Output 26.4.1 Statistics of Fit for Interval-Censored Data

The SEVSELECT Procedure

Model Selection			
Distribution	Converged	AICC Selected	
Burr	Yes	51.41112	No
Exp	Yes	44.64768	Yes
Gamma	Yes	47.63969	No
lgauss	Yes	48.05874	No
Logn	Yes	47.34027	No
Pareto	Yes	47.16908	No
Gpd	Yes	47.16908	No
Weibull	Yes	47.47700	No

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Burr	41.41112*	47.41112	51.41112	48.31888	0.08974*	0.00103*	0.0000816*
Exp	42.14768	44.14768*	44.64768*	44.45026*	0.26412	0.09936	0.01866
Gamma	41.92541	45.92541	47.63969	46.53058	0.19569	0.04608	0.00759
lgauss	42.34445	46.34445	48.05874	46.94962	0.34514	0.12301	0.02562
Logn	41.62598	45.62598	47.34027	46.23115	0.16853	0.01884	0.00333
Pareto	41.45480	45.45480	47.16908	46.05997	0.11423	0.00739	0.0009084
Gpd	41.45480	45.45480	47.16908	46.05997	0.11423	0.00739	0.0009084
Weibull	41.76272	45.76272	47.47700	46.36789	0.17238	0.03293	0.00472

Asterisk (*) denotes the best model in the column.

When the best distributions that are chosen by the likelihood-based and EDF-based statistics are different, you need to decide which fit statistic best represents your objective. In this example, if your objective is to minimize the distance between EDF and CDF values, then you should choose the Burr distribution. On the other hand, if your objective is to maximize the likelihood of the observed data while minimizing the

model complexity, then you should choose the exponential distribution. Note that the exponential distribution has worse (lower) raw likelihood than the Burr distribution, but it has better AIC, AICC, and SBC statistics than the Burr distribution because the exponential distribution has only one parameter compared to the three parameters of the Burr distribution. Further, the small sample size of 10 helps accentuate the role of model complexity in the AIC, AICC, and SBC statistics. If the sample size would have been larger, the exponential distribution might not have won according to the likelihood-based statistics.

Example 26.5: Defining a Finite Mixture Model That Has a Scale Parameter

A finite mixture model is a stochastic model that postulates that the probability distribution of the data generation process is a mixture of a finite number of probability distributions. For example, when an insurance company analyzes loss data from multiple policies that are underwritten in different geographic regions, some regions might behave similarly, but the distribution that governs some regions might be different from the distribution that governs other regions. Further, it might not be known which regions behave similarly. Also, the larger amounts of losses might follow a different stochastic process from the stochastic process that governs the smaller amounts of losses. It helps to model all policies together in order to pool the data together and exploit any commonalities among the regions, and the use of a finite mixture model can help capture the differences in distributions across regions and ranges of loss amounts.

Formally, if f_i and F_i denote the PDF and CDF, respectively, of component distribution i and p_i represents the mixing probability that is associated with component i , then the PDF and CDF of the finite mixture of K distribution components are

$$f(x; \Theta, \mathbf{p}) = \sum_{i=1}^K p_i f_i(x; \Theta_i)$$

$$F(x; \Theta, \mathbf{p}) = \sum_{i=1}^K p_i F_i(x; \Theta_i)$$

where Θ_i denotes the parameters of component distribution i and Θ denotes the parameters of the mixture distribution, which is a union of all the Θ_i parameters. $\mathbf{p}$ denotes the set of mixing probabilities. All mixing probabilities must add up to 1 ($\sum_{i=1}^K p_i = 1$).

You can define the finite mixture of a specific number of components and specific distributions for each of the components by defining the FCMP functions for the PDF and CDF. However, in general, it is not possible to fit a scale regression model by using any finite mixture distribution unless you take special care to ensure that the mixture distribution has a scale parameter. This example provides a formulation of a two-component finite mixture model that has a scale parameter.

To start with, each component distribution must have either a scale parameter or a log-transformed scale parameter. Let θ_1 and θ_2 denote the scale parameters of the first and second components, respectively. Let $p_1 = p$ be the mixing probability, which makes $p_2 = 1 - p$ by using the constraint on $\mathbf{p}$. The PDF of the mixture of these two distributions can be written as

$$f(x; \theta_1, \theta_2, \Phi, p) = \frac{p}{\theta_1} f_1\left(\frac{x}{\theta_1}; \Phi_1\right) + \frac{1-p}{\theta_2} f_2\left(\frac{x}{\theta_2}; \Phi_2\right)$$

where Φ_1 and Φ_2 denote the sets of nonscale parameters of the first and second components, respectively, and Φ denotes a union of Φ_1 and Φ_2 . For the mixture to have the scale parameter θ , the PDF must be of the

form

$$f(x; \theta, \Phi', p) = \frac{1}{\theta} \left(p f_1\left(\frac{x}{\theta}; \Phi'_1\right) + (1 - p) f_2\left(\frac{x}{\theta}; \Phi'_2\right) \right)$$

where Φ' , Φ'_1 , and Φ'_2 denote the modified sets of nonscale parameters. One simple way to achieve this is to make $\theta_1 = \theta_2 = \theta$ and $\Phi' = \Phi$; that is, you simply equate the scale parameters of both components and keep the set of nonscale parameters unchanged. However, forcing the scale parameters to be equal in both components is restrictive, because the mixture cannot model potential differences in the scales of the two components. A better approach is to tie the scale parameters of the two components by a ratio such that $\theta_1 = \theta$ and $\theta_2 = \rho\theta$. If the ratio parameter ρ is estimated along with the other parameters, then the mixture distribution becomes flexible enough to model the variations across the scale parameters of individual components.

To summarize, the PDF and CDF are of the following form for the two-component mixture that has a scale parameter:

$$f(x; \theta, \rho, \Phi, p) = \frac{1}{\theta} \left(p f_1\left(\frac{x}{\theta}; \Phi_1\right) + (1 - p) f_2\left(\frac{x}{\theta}; \rho, \Phi_2\right) \right)$$

$$F(x; \theta, \rho, \Phi, p) = p F_1\left(\frac{x}{\theta}; \Phi_1\right) + (1 - p) F_2\left(\frac{x}{\theta}; \rho, \Phi_2\right)$$

This can be generalized to a mixture of K components by introducing the $K - 1$ ratio parameters ρ_i that relate the scale parameters of each of the K components to the scale parameter θ of the mixture distribution as follows:

$$\theta_1 = \theta$$

$$\theta_i = \rho_i \theta; i \in [2, K]$$

In order to illustrate this approach, define a mixture of two lognormal distributions by using the following PDF function:

$$f(x; \mu, \sigma_1, p_2, \rho_2, \sigma_2) = \frac{(1 - p_2)}{\sigma_1 x \sqrt{2\pi}} \exp\left(\frac{-(\log(x) - \mu)^2}{2\sigma_1^2}\right) +$$

$$\frac{p_2}{\sigma_2 x \sqrt{2\pi}} \exp\left(\frac{-(\log(x) - \mu - \log(\rho_2))^2}{2\sigma_2^2}\right)$$

You can verify that μ serves as the log of the scale parameter θ ($\mu = \log(\theta)$).

The following PROC FCMP steps encode this formulation in a distribution named SLOGNMIX2 for use with PROC SEVSELECT:

```
/*- Define mixture of 2 lognormal distributions with a log-scale parameter -*/
proc fcmp library=sashelp.svrtldist outlib=work.sevexmpl.models;
    function slognmix2_description() $128;
        return ("Mixture of two lognormals with a log-scale parameter Mu");
    endsub;

    function slognmix2_scaletransform() $8;
        return ("LOG");
    endsub;
```

```

function slognmix2_pdf(x, Mu, Sigma1, p2, Rho2, Sigma2);
    Mu1 = Mu;
    Mu2 = Mu + log(Rho2);
    pdf1 = logn_pdf(x, Mu1, Sigma1);
    pdf2 = logn_pdf(x, Mu2, Sigma2);
    return ((1-p2)*pdf1 + p2*pdf2);
endsub;

function slognmix2_cdf(x, Mu, Sigma1, p2, Rho2, Sigma2);
    Mu1 = Mu;
    Mu2 = Mu + log(Rho2);
    cdf1 = logn_cdf(x, Mu1, Sigma1);
    cdf2 = logn_cdf(x, Mu2, Sigma2);
    return ((1-p2)*cdf1 + p2*cdf2);
endsub;

subroutine slognmix2_parinit(dim, x[*], nx[*], F[*], Ftype,
                           Mu, Sigma1, p2, Rho2, Sigma2);
    outargs Mu, Sigma1, p2, Rho2, Sigma2;
    array m[1] / nosymbols;
    p2 = 0.5;
    Rho2 = 0.5;
    median = svrtutil_percentile(0.5, dim, x, F, Ftype);
    Mu = log(2*median/1.5);
    call svrtutil_rawmoments(dim, x, nx, 1, m);
    lm1 = log(m[1]);

    /* Search Rho2 that makes log(sample mean) > Mu */
    do while (lm1 <= Mu and Rho2 < 1);
        Rho2 = Rho2 + 0.01;
        Mu = log(2*median/(1+Rho2));
    end;
    if (Rho2 >= 1) then
        /* If Mu cannot be decreased enough to make it less
           than log(sample mean), then revert to Rho2=0.5.
           That will set Sigma1 and possibly Sigma2 to missing.
           PROC SEVSELECT replaces missing initial values with 0.001. */
        Mu = log(2*median/1.5);

        Sigma1 = sqrt(2.0*(log(m[1])-Mu));
        Sigma2 = sqrt(2.0*(log(m[1])-Mu-log(Rho2)));
    endsub;

subroutine slognmix2_lowerbounds(Mu, Sigma1, p2, Rho2, Sigma2);
    outargs Mu, Sigma1, p2, Rho2, Sigma2;
    Mu = .; /* Mu has no lower bound */
    Sigma1 = 0; /* Sigma1 > 0 */
    p2 = 0; /* p2 > 0 */
    Rho2 = 0; /* Rho2 > 0 */
    Sigma2 = 0; /* Sigma2 > 0 */
endsub;

subroutine slognmix2_upperbounds(Mu, Sigma1, p2, Rho2, Sigma2);

```

```

outargs Mu, Sigma1, p2, Rho2, Sigma2;
Mu = .; /* Mu has no upper bound */
Sigma1 = .; /* Sigma1 has no upper bound */
p2 = 1; /* p2 < 1 */
Rho2 = 1; /* Rho2 < 1 */
Sigma2 = .; /* Sigma2 has no upper bound */
endsub;
quit;

```

As shown in previous examples, an important aspect of defining a distribution for use with PROC SEVSELECT is the definition of the PARMINIT subroutine that initializes the parameters. For mixture distributions, in general, the parameter initialization is a nontrivial task. For a two-component mixture, some simplifying assumptions make the problem easier to handle. For the initialization of SLOGNMIX2, the initial values of p_2 and ρ_2 are fixed at 0.5, and the following two simplifying assumptions are made:

- The median of the mixture is the average of the medians of the two components:

$$F^{-1}(0.5) = (\exp(\mu_1) + \exp(\mu_2))/2 = \exp(\mu)(1 + \rho_2)/2$$

Solution of this equation yields the value of μ in terms of ρ_2 and the sample median.

- Each component has the same mean, which implies the following:

$$\exp(\mu + \sigma_1^2/2) = \exp(\mu + \log(\rho_2) + \sigma_2^2/2)$$

If X_i represents the random variable of component distribution i and X represents the random variable of the mixture distribution, then the following equation holds for the raw moment of any order k :

$$E[X^k] = \sum_{i=1}^K p_i E[X_i^k]$$

This, in conjunction with the assumption on component means, leads to the equations

$$\log(m_1) = \mu + \frac{\sigma_1^2}{2}$$

$$\log(m_1) = \mu + \log(\rho_2) + \frac{\sigma_2^2}{2}$$

where m_1 denotes the first raw moment of the sample. Solving these equations leads to the following values of σ_1 and σ_2 :

$$\sigma_1^2 = 2(\log(m_1) - \mu)$$

$$\sigma_2^2 = 2(\log(m_1) - \mu - \log(\rho_2))$$

Note that σ_1 has a valid value only if $\log(m_1) > \mu$. Among the many possible methods of ensuring this condition, the SLOGNMIX2_PARMINIT subroutine uses the method of doing a linear search over ρ_2 .

Even when the preceding assumptions are not true for a particular problem, they produce reasonable initial values to help guide the nonlinear optimizer to an acceptable optimum if the mixture of two lognormal distributions is indeed a good fit for your input data. This is illustrated by the results of the following steps that fit the SLOGNMIX2 distribution to simulated data, which have different means for the two components (12.18 and 22.76, respectively), and the median of the sample (15.95) is not equal to the average of the medians of the two components (7.39 and 20.09, respectively):

```

/*----- Simulate a lognormal mixture sample -----*/
data testlognmix(keep=y);
  call streaminit(12345);
  Mu1 = 2;
  Sigma1 = 1;
  i = 0;
  do j=1 to 2000;
    y = exp(Mu1) * rand('LOGNORMAL')**Sigma1;
    output;
  end;
  Mu2 = 3;
  Sigma2 = 0.5;
  do j=1 to 3000;
    y = exp(Mu2) * rand('LOGNORMAL')**Sigma2;
    output;
  end;
run;

/*----- Load data into the session -----*/
data mylib.testlognmix;
  set testlognmix;
run;

/*-- Fit and compare scale regression models with 2-component --*/
/*-- lognormal mixture and the standard lognormal distribution --*/
options cmplib=(work.sevexmpl);

proc sevselect data=mylib.testlognmix print=all;
  loss y;
  dist slognmix2 logn;
run;

```

The comparison of the fit statistics of SLOGNMIX2 and LOGN, as shown in [Output 26.5.1](#), confirms that the two-component mixture is certainly a better fit to these data than the single lognormal distribution.

Output 26.5.1 Comparison of Fitting One versus Two Lognormal Components to Mixture Data
The SEVSELECT Procedure

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Slognmix2	38343*	38353*	38353*	38386*	0.52221*	0.19843*	0.02728*
Logn	39073	39077	39077	39090	5.86522	66.93414	11.72703
Asterisk (*) denotes the best model in the column.							

The detailed results for the SLOGNMIX2 distribution are shown in [Output 26.5.2](#). According to the “Initial Parameter Values and Bounds” table, the initial value of ρ_2 is not 0.5, indicating that a linear search was conducted to ensure $\log(m_1) > \mu$.

Output 26.5.2 Detailed Estimation Results for the SLOGNMIX2 Distribution**The SEVSELECT Procedure****Slognmix2 Distribution**

Model Information	
Distribution	Slognmix2
Description	Mixture of two lognormals with a log-scale parameter Mu
Distribution Parameters	5

Initial Parameter Values and Bounds

Parameter	Initial Value	Lower Bound	Upper Bound
Mu	2.92006	-Infty	Infty
Sigma1	0.10455	1.05367E-8	Infty
P2	0.50000	1.05367E-8	1.00000
Rho2	0.72000	1.05367E-8	1.00000
Sigma2	0.81728	1.05367E-8	Infty

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Optimization Summary

Optimization Technique	Trust Region
Iterations	7
Function Calls	18
Log Likelihood	-19171.54807

Parameter Estimates

Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	3.00922	0.01554	193.68	<.0001
Sigma1	1	0.49516	0.01451	34.13	<.0001
P2	1	0.40619	0.02600	15.62	<.0001
Rho2	1	0.37212	0.02038	18.26	<.0001
Sigma2	1	1.00019	0.02124	47.09	<.0001

By using the relationship that $\mu_2 = \mu + \log(\rho_2)$, you can see that the final parameter estimates are reasonably close to the true parameter values that were used to simulate the input sample.

Example 26.6: Scale Regression with Rich Regression Effects

This example illustrates the use of regression effects that include CLASS variables and interaction effects.

Consider that you, as an actuary at an automobile insurance company, want to evaluate the effect of certain external factors on the distribution of the severity of the losses that your policyholders incur. Such analysis can help you determine the relative differences in premiums that you should charge to policyholders who have different characteristics. Assume that when you collect and record the information about each claim, you also collect and record some key characteristics of the policyholder and the vehicle that is involved in the claim. This example focuses on the following five factors: type of car, safety rating of the car, gender of the policyholder, education level of the policyholder, and annual household income of the policyholder (which can be thought of as a proxy for the luxury level of the car). Let these regressors be recorded in the variables CarType (1: sedan, 2: sport utility vehicle), CarSafety (scaled to be between 0 and 1, the safest being 1), Gender (1: female, 2: male), Education (1: high school graduate, 2: college graduate, 3: advanced degree holder), and Income (scaled by a factor of 1/100,000), respectively. Let the historical data about the severity of each loss be recorded in the LossAmount variable of the mylib.Losses data table. Let the data table also contain two additional variables, Deductible and Limit, that record the deductible and ground-up loss limit provisions, respectively, of the insurance policy that the policyholder has. The limit on ground-up loss is usually derived from the payment limit that a typical insurance policy states. Deductible serves as the left-truncation variable, and Limit serves as the right-censoring variable.

The following SAS statements simulate an example of the mylib.Losses data table:

```
proc format casfmtlib='myfmtlib';
    value genderFmt 1='Female'
                  2='Male';
run;

data losses(keep=gender carType education carSafety income
               lossAmount deductible limit);
    call streaminit(12345);
    array sx{8} _temporary_;
    array sbeta{9} _TEMPORARY_ (5 0.6 0.4 -0.75 -0.3 0.4 0.7 -0.5 -0.3);

    length carType $8 education $16;
    format gender genderFmt.;

    sigma = 0.5;
    do lossEventId=1 to 6000;
        /* Simulate policyholder and vehicle attributes */
        do i=1 to dim(sx);
            sx(i) = 0;
        end;

        if (rand('UNIFORM') < 0.5) then do;
            gender = 1; * female;
            sx(2) = 1;
        end;
        else do;
            gender = 2; * male;
        end;
    end;
```

```

if (rand('UNIFORM') < 0.7) then do;
    carInt = 1;
    carType = 'Sedan';
end;
else do;
    carInt = 2;
    carType = 'SUV';
    sx(1) = 1;
end;

educationLevel = rand('UNIFORM');
if (educationLevel < 0.5) then do;
    eduInt = 1;
    education = 'High School';
end;
else if (educationLevel < 0.85) then do;
    eduInt = 2;
    education = 'College';
    if (carInt=1) then
        sx(8) = 1;
    else
        sx(6) = 1;
end;
else do;
    eduInt = 3;
    education = 'AdvancedDegree';
    if (carInt=1) then
        sx(7) = 1;
    else
        sx(5) = 1;
end;

carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */
sx(3) = carSafety;

income = MAX(15000,int(rand('NORMAL', eduInt*30000, 50000)))/100000;
sx(4) = income;

/* Simulate lognormal severity */
Mu = sbeta(1);
do i=1 to dim(sx);
    Mu = Mu + sx(i) * sbeta(i+1);
end;
lossAmount = exp(Mu) * rand('LOGNORMAL')**Sigma;
loglossAmount = log(lossAmount);

deductible = lossAmount * rand('UNIFORM');
if (rand('UNIFORM') < 0.25) then
    limit = lossAmount;

output;
end;
run;

```

```
data mylib.losses;
  set losses;
run;
```

The variables CarType, Education, and Gender each contain a known, finite set of discrete values. By specifying such variables as classification variables, you can separately identify the effect of each level of the variable on the severity distribution. For example, you might be interested in finding out how the magnitude of loss for a sport utility vehicle (SUV) differs from that for a sedan. This is an example of a main effect. You might also want to evaluate how the distribution of losses that are incurred by a policyholder with a college degree who drives a SUV differs from that of a policyholder with an advanced degree who drives a sedan. This is an example of an interaction effect. You can include various such types of effects in the scale regression model. For more information about the effect types, see the section “[Specification and Parameterization of Model Effects](#)” on page 91 in Chapter 4, “[Shared Concepts](#).”

Analyzing such a rich set of regression effects can help you make more accurate predictions about the losses that a new applicant with certain characteristics might incur when he or she requests insurance for a specific vehicle, which can further help you with ratemaking decisions.

The following PROC SEVSELECT step fits the scale regression model with a lognormal distribution to data in the mylib.Losses data table, and stores the model and parameter estimate information in the mylib.Est data table on the CAS server:

```
/* Fit scale regression model with different types of regression effects */
proc sevselect data=mylib.losses outest=mylib.est print=all;
  loss lossAmount / lt=deductible rc=limit;
  class carType gender education;
  scalemodel carType gender carSafety income education*carType
             income*gender carSafety*income;
  dist logn;
run;
```

The SCALEMODEL statement in the preceding PROC SEVSELECT step includes two main effects (carType and gender), two singleton continuous effects (carSafety and income), one interaction effect (education*carType), one continuous-by-class effect (income*gender), and one polynomial continuous effect (carSafety*income).

When you specify a CLASS statement, it is recommended that you observe the “Class Level Information” table. For this example, the table is shown in [Output 26.6.1](#). Note that if you specify BY-group processing, then the class level information might change from one BY group to the next, potentially resulting in a different parameterization for each BY group.

Output 26.6.1 Class Level Information Table
The SEVSELECT Procedure

Class Level Information		
Class	Levels	Values
carType	2	SUV Sedan
gender	2	Female Male
education	3	AdvancedDegree College High School

The regression modeling results for the lognormal distribution are shown in [Output 26.6.2](#). The “Initial Parameter Values and Bounds” table is important especially because the preceding PROC SEVSELECT

step uses the default GLM parameterization, which is a singular parameterization—that is, it results in some redundant parameters. As shown in the table, the redundant parameters correspond to the last level of each classification variable; this correspondence is a defining characteristic of a GLM parameterization. An alternative would be to use the reference parameterization by specifying the `PARAM=REFERENCE` option in the `CLASS` statement, which does not generate redundant parameters for effects that contain `CLASS` variables and enables you to specify a reference level for each `CLASS` variable.

Output 26.6.2 Initial Values for the Scale Regression Model with Class and Interaction Effects

Initial Parameter Values and Bounds			
Parameter	Initial Value	Lower Bound	Upper Bound
Mu	4.85200	-709.78271	709.78271
Sigma	0.52443	1.05367E-8	Infy
carType SUV	0.54686	-709.78271	709.78271
carType Sedan	Redundant	-709.78271	709.78271
gender Female	0.34893	-709.78271	709.78271
gender Male	Redundant	-709.78271	709.78271
carSafety	-0.63504	-709.78271	709.78271
income	-0.24031	-709.78271	709.78271
carType SUV * education AdvancedDegree	0.32719	-709.78271	709.78271
carType SUV * education College	0.68899	-709.78271	709.78271
carType SUV * education High School	Redundant	-709.78271	709.78271
carType Sedan * education AdvancedDegree	-0.44650	-709.78271	709.78271
carType Sedan * education College	-0.26834	-709.78271	709.78271
carType Sedan * education High School	Redundant	-709.78271	709.78271
income * gender Female	0.00843	-709.78271	709.78271
income * gender Male	Redundant	-709.78271	709.78271
carSafety * income	-0.04744	-709.78271	709.78271

The convergence and optimization summary information in [Output 26.6.3](#) indicates that the scale regression model for the lognormal distribution has converged with the default optimization technique in five iterations.

Output 26.6.3 Optimization Summary for the Scale Regression Model with Class and Interaction Effects

Convergence Status	
Convergence criterion (GCONV=1E-8) satisfied.	
Optimization Summary	
Optimization Technique	Trust Region
Iterations	5
Function Calls	14
Log Likelihood	-19716.25785

The “Parameter Estimates” table in [Output 26.6.4](#) shows the distribution parameter estimates and estimates for various regression effects. You can use the estimates for effects that contain `CLASS` variables to infer the relative influence of various `CLASS` variable levels. For example, on average, the magnitude of losses that are incurred by the female drivers is $\exp(0.41714) \approx 1.52$ times greater than that of male drivers, and an SUV driver with an advanced degree incurs a loss that is on average $\exp(0.42454) / \exp(-0.35208) \approx 2.174$

times greater than the loss that a college-educated sedan driver incurs. Neither the continuous-by-class effect `income*gender` nor the polynomial continuous effect `carSafety*income` is significant in this example.

Output 26.6.4 Parameter Estimates for the Scale Regression with Class and Interaction Effects

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	5.12332	0.03792	135.11	<.0001
Sigma	1	0.56941	0.00744	76.56	<.0001
carType SUV	1	0.64686	0.02989	21.64	<.0001
carType Sedan	0	0	.	.	.
gender Female	1	0.41714	0.03213	12.98	<.0001
gender Male	0	0	.	.	.
carSafety	1	-0.88078	0.05532	-15.92	<.0001
income	1	-0.38045	0.05128	-7.42	<.0001
carType SUV * education AdvancedDegree	1	0.42454	0.05107	8.31	<.0001
carType SUV * education College	1	0.73228	0.03738	19.59	<.0001
carType SUV * education High School	0	0	.	.	.
carType Sedan * education AdvancedDegree	1	-0.56685	0.03525	-16.08	<.0001
carType Sedan * education College	1	-0.35208	0.02618	-13.45	<.0001
carType Sedan * education High School	0	0	.	.	.
income * gender Female	1	0.01288	0.04394	0.29	0.7694
income * gender Male	0	0	.	.	.
carSafety * income	1	0.06499	0.07547	0.86	0.3892

If you want to update the model when new claims data arrive, then you can potentially speed up the estimation process by specifying the `OUTEST=` data table that is created by the preceding PROC SEVSELECT step as an `INEST=` data table in a new PROC SEVSELECT step. To illustrate, the following PROC SEVSELECT step refits the model on the same input data as the preceding PROC SEVSELECT, but it uses the `mylib.Est` data table that is created by that step as an `INEST=` data table:

```
/* Refit scale regression model on new data */
proc sevselect data=mylib.losses inest=mylib.est print=all;
  loss lossAmount / lt=deductible rc=limit;
  class carType gender education;
  scalemodel carType gender carSafety income education*carType
             income*gender carSafety*income;
  dist logn;
run;
```

Because the `INEST=` data table is used to initialize the distribution and regression parameters, the optimization occurs in very few iterations, as shown in [Output 26.6.5](#).

Output 26.6.5 Optimization Summary for Refitting the Scale Regression Model with the `INEST=` Option

The SEVSELECT Procedure

Logn Distribution

Convergence Status
Convergence criterion (ABSGCONV=0.00001) satisfied.

Output 26.6.5 *continued*

Optimization Summary	
Optimization Technique	Trust Region
Iterations	0
Function Calls	4
Log Likelihood	-19716.25785

Example 26.7: Scale Regression Model Selection

This example extends “[Example 26.6: Scale Regression with Rich Regression Effects](#)” to illustrate the model selection features of the SEVSELECT procedure. There are two phases of model selection when you fit a scale regression model. First, you want to select an optimal set of regression effects for each distribution’s scale regression model. The SELECTION statement provides several methods to do that. Second, you want to identify the best severity distribution by comparing the final selected scale regression models for each distribution. PROC SEVSELECT displays tables to help you do that.

The following SEVSELECT step uses the same data table as in [Example 26.6](#) to find the best set of scale regression effects for three severity distributions—lognormal (Logn), Burr, and Weibull:

```
proc sevselect data=mylib.losses outest=mylib.est print=all;
  loss lossAmount / lt=deductible rc=limit;
  class carType gender education;
  scalemodel carType gender carSafety income education*carType
             income*gender carSafety*income;
  selection;
  dist logn burr weibull;
  output out=mylib.score copyvars=(carType gender education carSafety income)
         functions=(mean) quantiles=(points=0.5 0.975 names=(median var));
run;
```

The SELECTION statement without any options uses the stepwise selection method by default. PROC SEVSELECT reports the settings of the selection process in the “Selection Information” table, as shown in [Output 26.7.1](#).

Output 26.7.1 Selection Method Settings**The SEVSELECT Procedure****Logn Distribution**

Selection Information	
Selection Method	Stepwise
Select Criterion	SBC
Stop Criterion	SBC
Effect Hierarchy Enforced	None
Stop Horizon	3

For each severity distribution, PROC SEVSELECT reports the summary of the selection process, as shown in [Output 26.7.2](#) for the lognormal distribution. The “Selection Summary” table shows the order in which

regression effects enter or leave the model. In this case, no effect leaves the model after entering the model. In the first step, PROC SEVSELECT fits seven models, each with an intercept and one of the seven eligible effects in the SCALEMODEL statement. The model that contains the intercept and carType*education effect has the lowest value of the Schwarz Bayesian criterion (SBC) statistic in the first step, and hence it enters the model. In the second step, PROC SEVSELECT compares the model that does not contain the existing effect, carType*education, with models that contain that effect and one of the remaining six effects. The model that contains the intercept, carType*education, and carSafety effects is the best in the second step according to the SBC statistic. The process of removing an effect from the model or adding an effect that is not in the model continues until the stop criterion does not improve or the stop horizon condition is met. As the results in [Output 26.7.2](#) show, the optimal value of the SBC criterion is achieved in the fourth step. The final scale regression model for the lognormal distribution contains the following effects: intercept, gender, carSafety, carType*education, and income*gender.

Output 26.7.2 Stepwise Selection Summary for the Lognormal Distribution

The SEVSELECT Procedure

Logn Distribution Selection Details

Selection Summary			
Step	Effect Entered	Number Effects In	SBC
0	Intercept	1	35975.2456
1	carType*education	2	34121.0008
2	carSafety	3	33709.8419
3	income*gender	4	33298.5871
4	gender	5	33159.7903*
* Optimal Value Of Criterion			

Stepwise selection stopped because adding or removing an effect does not improve the SBC criterion.

The model at step 4 is selected.

Selected Effects: Intercept gender carSafety carType*education income*gender

The selection summary information for the Weibull distribution is shown in [Output 26.7.3](#). It shows that the best model in the sixth step is obtained by removing the carSafety*income effect, which was added in the second step. The model in the sixth step also has the lowest value of the SBC criterion; hence, it is the best model and contains the following effects: intercept, gender, carSafety, income, and carType*education.

Output 26.7.3 Stepwise Selection Summary for the Weibull Distribution**The SEVSELECT Procedure****Weibull Distribution
Selection Details**

Selection Summary			
Step	Effect Entered	Effect Removed	Number Effects In SBC
0	Intercept		1 36381.5042
1	carType*education		2 34477.5474
2	carSafety*income		3 34092.5186
3	gender		4 33684.7598
4	carSafety		5 33573.1854
5	income		6 33517.6460
6		carSafety*income	5 33509.5021*
* Optimal Value Of Criterion			

Stepwise selection stopped because adding or removing an effect does not improve the SBC criterion.

The model at step 6 is selected.

Selected Effects: Intercept gender carSafety income carType*education

PROC SEVSELECT reports the optimization summary, fit statistics, and parameter estimates for the final selected model of each severity distribution. The results for the lognormal distribution are shown in [Output 26.7.4](#). The “Parameter Estimates” table shows that all regression parameters in the final model are statistically significant. Comparison of these parameter estimates with those in [Output 26.6.4](#), which were obtained without using the SELECTION statement, shows that the statistically insignificant effect, carSafety*income, is not present in the final selected model. The comparison also shows that the stepwise selection process prefers income*gender effect over the income effect and, because of the exclusion of the income effect, the income*gender effect has become statistically significant.

Output 26.7.4 Selected Scale Regression Model for the Lognormal Distribution

Convergence Status	
Convergence criterion (GCONV=1E-8) satisfied.	
Optimization Summary	
Optimization Technique	Trust Region
Iterations	5
Function Calls	14
Log Likelihood	-16533.05056

Output 26.7.4 *continued*

Fit Statistics						
-2 Log Likelihood			33066			
Akaike's Information Criterion			33088			
Corrected Akaike's Information Criterion			33088			
Schwarz's Bayesian Information Criterion			33160			
Kolmogorov-Smirnov Statistic			11.00535			
Anderson-Darling Statistic			649.55145			
Cramer-von Mises Statistic			54.77368			

Parameter Estimates						
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t	
Mu	1	5.09549	0.03326	153.18	<.0001	
Sigma	1	0.56924	0.00811	70.18	<.0001	
gender Female	1	0.43162	0.03516	12.28	<.0001	
gender Male	0	0	.	.	.	
carSafety	1	-0.83597	0.03574	-23.39	<.0001	
carType SUV * education AdvancedDegree	1	1.02458	0.05115	20.03	<.0001	
carType SUV * education College	1	1.40166	0.03516	39.87	<.0001	
carType SUV * education High School	1	0.64207	0.03273	19.61	<.0001	
carType Sedan * education AdvancedDegree	1	-0.54784	0.03836	-14.28	<.0001	
carType Sedan * education College	1	-0.34621	0.02861	-12.10	<.0001	
carType Sedan * education High School	0	0	.	.	.	
income * gender Female	1	-0.34466	0.03490	-9.88	<.0001	
income * gender Male	1	-0.36093	0.03588	-10.06	<.0001	

After the effect selection process chooses the best scale regression model for each severity distribution, the next phase of model selection is to compare the final models across all severity distributions and identify the best severity distribution. You can do that by using the “All Fit Statistics” table, which helps you compare the final selected models according to all the fit statistics. For this example, [Output 26.7.5](#) compares the final models of the three severity distributions—lognormal, Burr, and Weibull.

Output 26.7.5 Comparison of the Selected Scale Regression Models for Different Severity Distributions

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	33066*	33088*	33088*	33160*	11.00535	649.55145	54.77368
Burr	33100	33124	33125	33203	11.28423	519.64162*	55.03613
Weibull	33424	33444	33444	33510	10.39183*	756.77108	36.41622*

Asterisk (*) denotes the best model in the column.

The lognormal distribution's final selected model is the best according to all the likelihood-based statistics, the Burr distribution's final selected model is the best according to the Anderson-Darling (AD) statistic, and the Weibull distribution's final selected model is the best according to the Kolmogorov-Smirnov (KS) and Cramér-von Mises (CvM) statistics. You can choose the fit statistic that is most suitable for your application and use it to identify the best severity distribution. Alternatively, if your goal is to estimate some loss distribution statistic, such as the value-at-risk (VaR), for a particular scenario, then instead of choosing

one severity distribution, you might want to compute and compare the desired statistic for the final scale regression model of each severity distribution that wins according to at least one fit statistic. The **OUTPUT** statement is useful for such evaluation, because it helps you *score* each observation of the input data table that you specify in the **DATA=** option. In the context of PROC SEVSELECT, scoring an observation is equivalent to computing particular statistics for the severity distribution of the losses. The previous PROC SEVSELECT step specifies the following OUTPUT statement to write the scores for each observation to the mylib.Score output table:

```
output out=mylib.score copyvars=(carType gender education carSafety income)
      functions=(mean) quantiles=(cdf=0.5 0.975 names=(median var));
```

When you specify the scale regression model, the scale parameter of each severity distribution depends on the values of the regression effects in the final selected model of that distribution. For each observation in the input table, PROC SEVSELECT computes the estimate of the scale parameter and uses it to evaluate the scoring and quantile functions that you specify in the **FUNCTIONS=** and **QUANTILES=** options, respectively. The OUTPUT statement of this example specifies that three statistics be computed for the final model of each severity distribution: the *mean* scoring function and two quantile functions, $Q_{\text{dist}}(0.5, \hat{\Theta})$ and $Q_{\text{dist}}(0.975, \hat{\Theta})$, where $Q_{\text{dist}}(p, \Theta)$ denotes the value of the quantile function for a distribution *dist* that is evaluated at the CDF value of p for the estimated distribution parameters $\hat{\Theta}$. The variable that contains the estimate of the mean function is named *mean*. The variables that contain the quantile function estimate for $p=0.5$ and $p=0.975$ are named *median* and *var*, respectively; *var* denotes the value-at-risk (VaR). In order to compute a function F in the **FUNCTIONS=** option, for each severity distribution *dist*, the *dist_F* function must be available in the function library search path that you specify in the **CMPLIB=** system option. Also, in order to compute the quantile functions faster, it is recommended that the *dist_QUANTILE* function be available in the **CMPLIB=** search path. If PROC SEVSELECT does not find a quantile function's definition, it computes the quantile function by numerically inverting the cumulative distribution function (CDF). For each predefined distribution, such as the lognormal, Burr, and Weibull distributions of this example, the *dist_MEAN* and *dist_QUANTILE* functions are already defined and available to you in the function library, SasHELP.SvrtDist, that PROC SEVSELECT includes by default.

[Output 26.7.6](#), [Output 26.7.7](#), and [Output 26.7.8](#) compare the estimates of the mean, median, and value-at-risk (*var*), respectively, of the three severity distributions for the first 10 observations that are fetched from the mylib.Score table. **NOTE:** The estimates are all available in the same table but are shown here separately for better comparison among different severity distributions. These results illustrate the following:

- The **OUT=** data table contains the variables that you specify in the **COPYVARS=** option. This helps you associate the scoring statistics with the values of the regression effects that decide the severity distribution's scale parameter.
- The estimates of a particular statistic vary by observation, because the scale parameter value depends on the values of the regression effects. You can interpret the estimate of the mean as the average loss that a particular policyholder will incur if he or she has the characteristics that are quantified in the values of the regression effects. For example, the fifth observation in [Output 26.7.6](#) shows that a male policyholder with a college degree who earns 0.70732 on a normalized scale and drives a sport utility vehicle (SUV) that has a normalized car safety rating of 0.8446 is expected to incur an average loss of 298 units if you use the lognormal distribution's selected scale regression model for estimation, or an average loss of 283 units if you use the Weibull distribution's selected scale regression model for estimation.

Output 26.7.6 Estimates of the Mean for Final Models of All Severity Distributions

Obs	gender	education	income	carType	carSafety	Logn_MEAN	Burr_MEAN	Weibull_MEAN
1	Male	College	1.00896	SUV	0.85747	264.595	279.531	249.048
2	Female	College	0.52584	SUV	0.84267	495.276	514.365	481.811
3	Female	High School	0.15000	SUV	0.84062	264.218	270.823	268.200
4	Male	AdvancedDegree	0.51942	Sedan	0.01410	90.959	93.358	88.674
5	Female	AdvancedDegree	1.27245	SUV	0.84238	262.683	272.801	241.754
6	Female	College	1.12252	Sedan	0.81153	72.069	74.220	69.784
7	Female	College	0.38431	Sedan	0.36374	135.151	137.595	133.730
8	Female	High School	0.49503	SUV	0.69762	264.384	272.216	263.578
9	Female	College	0.98769	SUV	0.06408	809.817	844.885	758.977
10	Male	College	0.67508	Sedan	0.14417	94.361	97.079	90.119

- The estimates of the median, which is a robust estimate of the central tendency, vary less across different severity distributions. So you might want to use the median loss to predict the average loss that a policyholder will incur.

Output 26.7.7 Estimates of the Median for Final Models of All Severity Distributions

Obs	gender	education	income	carType	carSafety	Logn_MEDIAN	Burr_MEDIAN	Weibull_MEDIAN
1	Male	College	1.00896	SUV	0.85747	225.018	228.080	227.398
2	Female	College	0.52584	SUV	0.84267	421.196	419.690	439.925
3	Female	High School	0.15000	SUV	0.84062	224.698	220.975	244.884
4	Male	AdvancedDegree	0.51942	Sedan	0.01410	77.354	76.174	80.966
5	Female	AdvancedDegree	1.27245	SUV	0.84238	223.392	222.589	220.737
6	Female	College	1.12252	Sedan	0.81153	61.289	60.559	63.718
7	Female	College	0.38431	Sedan	0.36374	114.936	112.269	122.104
8	Female	High School	0.49503	SUV	0.69762	224.839	222.112	240.665
9	Female	College	0.98769	SUV	0.06408	688.689	689.375	692.997
10	Male	College	0.67508	Sedan	0.14417	80.247	79.211	82.285

- The estimates of the value-at-risk (var), which depend on the heaviness of the tail of the distribution, vary significantly across severity distributions. For this example, the fitted Burr and Weibull distributions have the heaviest and lightest tails, respectively, among the three severity distributions. The VaR statistic is typically used to assess the worst-case loss that a policyholder might incur. You can interpret it as follows for the ninth observation in [Output 26.7.8](#): for a female policyholder with an advanced degree who earns 0.60157 and drives a sedan with a car safety rating of 0.69361, the probability that she will incur a loss greater than 228 units is 2.5% if you use the Burr distribution's scale regression model for estimation. You can add a CDF value of 0.995 to the CDF= suboption of the QUANTILES= option to get another value-at-risk estimate, which estimates a more extreme loss that a policyholder might incur with a probability of only 0.5%, but you might need to be prepared to cover that loss depending on the policy provisions.

Output 26.7.8 Estimates of the Value-at-Risk (VaR) for Final Models of All Severity Distributions

Obs	gender	education	income	carType	carSafety	Logn_VAR	Burr_VAR	Weibull_VAR
1	Male	College	1.00896	SUV	0.85747	686.68	801.10	585.53
2	Female	College	0.52584	SUV	0.84267	1285.35	1474.10	1132.77
3	Female	High School	0.15000	SUV	0.84062	685.71	776.14	630.55
4	Male	AdvancedDegree	0.51942	Sedan	0.01410	236.06	267.55	208.48
5	Female	AdvancedDegree	1.27245	SUV	0.84238	681.72	781.81	568.38
6	Female	College	1.12252	Sedan	0.81153	187.03	212.71	164.07
7	Female	College	0.38431	Sedan	0.36374	350.75	394.33	314.41
8	Female	High School	0.49503	SUV	0.69762	686.14	780.14	619.69
9	Female	College	0.98769	SUV	0.06408	2101.66	2421.33	1784.40
10	Male	College	0.67508	Sedan	0.14417	244.89	278.22	211.88

PROC SEVSELECT always includes the intercept in all scale regression models, so the selection process never attempts to fit a model without an intercept. In addition to the intercept, you can force other regression effects to be included in each selected model by specifying the **INCLUDE=** option in the **SCALEMODEL** statement. The following SEVSELECT step illustrates this by forcing the carSafety effect into every selected model:

```
proc sevselect data=mylib.losses outest=mylib.est print=all;
  loss lossAmount / lt=deductible rc=limit;
  class carType gender education;
  scalemodel carType gender carType*gender carSafety income education
             income*gender carSafety*income / include=(carSafety);
  selection method=forward(stop=aicc) hierarchy=single stophorizon=2 details=all;
  dist logn weibull;
run;
```

The various options in the SELECTION statement do the following:

- The **METHOD=FORWARD** option specifies that the forward elimination method of selection be used.
- The **HIERARCHY=SINGLE** option instructs PROC SEVSELECT not to add an interaction effect to the model until all the main effects in the interaction are already in the model.
- The **STOPHORIZON=2** option specifies that the **STOP=** criterion, which is the corrected Akaike's information criterion (AICC), must worsen for two steps in order for a local extremum to be detected.
- The **DETAILS=ALL** option displays details of all the steps of the selection process.

The "Selection Information" table in [Output 26.7.9](#) summarizes the selection settings.

Output 26.7.9 Selection Method Settings for Forward Selection**The SEVSELECT Procedure****Logn Distribution**

Selection Information	
Selection Method	Forward
Select Criterion	SBC
Stop Criterion	AICC
Effect Hierarchy Enforced	Single
Stop Horizon	2

The selection summary for the lognormal distribution, shown in [Output 26.7.10](#), illustrates the following:

- The model at step 0 includes both the intercept and carSafety effects. The carSafety effect always stays in the model.
- Because of the HIERARCHY=SINGLE option, the carType*gender interaction effect does not enter the model until the model contains both main effects (carType and gender) that constitute it. Similarly, the carSafety*income interaction effect between two continuous effects also does not enter the model before both carSafety and income are in the model.
- The stop criterion keeps improving for several steps, so even the smaller stop horizon of 2 does not affect the selection process for the lognormal distribution. The selection stops when all effects are in the model.
- The final model is chosen by the optimal value of the STOP= criterion, because the CHOOSE= option is not specified. For the lognormal distribution, it is the model at step 6, which contains all regression effects except the income*gender effect. If you had specified the CHOOSE=SBC option, then the final model would be the model in step 4 that excludes carType*gender, carSafety*income, and income*gender effects, because it has the smallest value for the SBC criterion.

Output 26.7.10 Forward Selection Summary for the Lognormal Distribution**The SEVSELECT Procedure****Logn Distribution
Selection Details**

Selection Summary				
Step	Effect Entered	Number Effects In	AICC	SBC
0	Intercept	1		
	carSafety	2	35766.6044	35786.1512
1	carType	3	34230.4771	34256.5378
2	gender	4	33895.0222	33927.5962
3	income	5	33656.7655	33695.8519
4	education	6	33594.6989	33646.8076*
5	carType*gender	7	33591.2653	33649.8839
6	carSafety*income	8	33590.2351*	33655.3629
7	income*gender	9	33591.8039	33663.4401
* Optimal Value Of Criterion				

Selection stopped because all effects are in the model.

The model at step 6 is selected.

Selected Effects: Intercept carType gender carType*gender carSafety income education carSafety*income

The selection summary for the Weibull distribution is shown in [Output 26.7.11](#). It indicates that the selection stopped at the local minimum of the AICC statistic at step 5 because of the shorter stop horizon.

Output 26.7.11 Forward Selection Summary for the Weibull Distribution**The SEVSELECT Procedure****Weibull Distribution
Selection Details**

Selection Summary				
Step	Effect Entered	Number Effects In	AICC	SBC
0	Intercept	1		
	carSafety	2	36202.4284	36221.9751
1	carType	3	34560.1129	34586.1736
2	gender	4	34200.9756	34233.5496
3	income	5	33935.3440	33974.4303
4	education	6	33857.1156	33909.2243*
5	carType*gender	7	33856.5340*	33915.1526
6	carSafety*income	8	33857.1143	33922.2421
* Optimal Value Of Criterion				

Selection stopped at a local minimum of the AICC criterion.

Output 26.7.11 *continued*

The model at step 5 is selected.

Selected Effects: Intercept carType gender carType*gender carSafety income education

As the results in [Output 26.7.10](#) and [Output 26.7.11](#) show, although the SELECT= criterion is reported in the “Selection Summary” table, it does not decide the final selected model. Instead, the SELECT= criterion is used to choose the best effect to add or remove in each step. To illustrate that, [Output 26.7.12](#) shows the details of step 1 for the Weibull distribution. PROC SEVSELECT produces the step details because of the DETAILS=ALL option in the SELECTION statement. Step 1 is the step after the initial step (step 0), which fits the model with the intercept and carSafety effect that the INCLUDE= option forces in. The “Entry Candidates” table of step 1 shows the candidates that PROC SEVSELECT evaluates for entering the model. For each candidate, it shows the value of the SELECT= criterion, which is the SBC statistic, of the model that includes that candidate. The interaction effects carType*gender and carSafety*income are not present in this list of candidates, because of the constraint that the HIERARCHY=SINGLE option imposes. The model that includes the carType effect has the smallest SBC statistic, so PROC SEVSELECT selects it in step 1. The “Optimization Summary” table shows that PROC SEVSELECT uses the trust region optimization method, which is the default optimization method for fitting a model in each step. You can use the [SELECTNLOTECH=](#) option in the PROC SEVSELECT statement to specify a different optimization method to use during each step of the selection. The “Parameter Estimates” table shows that step 1’s model includes the best entry candidate, carType, and the carSafety effect, which entered the model in step 0. The estimate of the intercept effect is reflected in the estimate of the scale parameter Theta, as described in the section “[Reporting Estimates of Regression Parameters](#)” on page 1797.

Output 26.7.12 Details of Step 1 of the Forward Selection Process for the Weibull Distribution**The SEVSELECT Procedure**

Weibull Distribution
Forward Selection: Step 1
Effect Entered: carType

Entry Candidates		
Rank	Effect	SBC
1	carType	34586.1736
2	gender	36000.8378
3	education	36142.6117
4	income	36150.6917

Convergence Status	
Convergence criterion (GCONV=1E-8) satisfied.	

Optimization Summary	
Optimization Technique	Trust Region
Iterations	4
Function Calls	13
Log Likelihood	-17276.05243

Output 26.7.12 *continued*

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	172.58489	4.92619	35.03	<.0001
Tau	1	1.39719	0.02266	61.65	<.0001
carType SUV	1	1.28993	0.02919	44.18	<.0001
carType Sedan	0	0	.	.	.
carSafety	1	-0.85372	0.04522	-18.88	<.0001

The fit summary of the final model for the Weibull distribution is shown in [Output 26.7.13](#). It shows that PROC SEVSELECT uses the trust region optimization method to estimate the parameters of the final model. The “Parameter Estimates” table shows that some regression parameters are not significant at the usual 95% confidence level. None of the selection criteria that PROC SEVSELECT supports depend directly on the p -values of regression effects. So a statistically insignificant effect might be included in the final model, as this example illustrates. If you do not want such effects in the final model, you can modify the selection method settings to search for a different model that contains all statistically significant effects. Alternatively, you can use your domain knowledge to decide which effects you want to retain from the selected model and then refit the model by appropriately modifying the SCALEMODEL statement.

Output 26.7.13 Selected Scale Regression Model for the Weibull Distribution

Convergence Status					
Convergence criterion (GCONV=1E-8) satisfied.					

Optimization Summary					
Optimization Technique	Trust Region				
Iterations	4				
Function Calls	12				
Log Likelihood	-16919.24895				

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	177.53313	5.63375	31.51	<.0001
Tau	1	1.63078	0.02621	62.23	<.0001
carType SUV	1	1.27004	0.03578	35.50	<.0001
carType Sedan	0	0	.	.	.
gender Female	1	0.45939	0.02703	16.99	<.0001
gender Male	0	0	.	.	.
carType SUV * gender Female	1	0.08009	0.04981	1.61	0.1079
carType SUV * gender Male	0	0	.	.	.
carType Sedan * gender Female	0	0	.	.	.
carType Sedan * gender Male	0	0	.	.	.
carSafety	1	-0.82500	0.03881	-21.26	<.0001
income	1	-0.38016	0.02832	-13.42	<.0001
education AdvancedDegree	1	-0.32602	0.03563	-9.15	<.0001
education College	1	-0.04246	0.02748	-1.55	0.1224
education High School	0	0	.	.	.

Example 26.8: Estimating Parameters Using the Cramér–von Mises Estimator

The SEVSELECT procedure enables you to estimate model parameters by minimizing your own objective function. This example illustrates how you can use PROC SEVSELECT to implement the Cramér–von Mises estimator. Let $F(y_i; \Theta)$ denote the estimate of CDF at y_i for a distribution with parameters Θ , and let $F_n(y_i)$ denote the empirical estimate of CDF (EDF) at y_i that is computed from a sample y_i , $1 \leq i \leq N$. Then, the Cramér–von Mises estimator of the parameters is defined as

$$\hat{\Theta} = \arg \min_{\Theta} \sum_{i=1}^N (F(y_i; \Theta) - F_n(y_i))^2$$

This estimator belongs to the class of minimum distance estimators. It attempts to estimate the parameters such that the squared distance between the CDF and EDF estimates is minimized.

The following PROC SEVSELECT step uses the Cramér–von Mises estimator to fit four candidate distribution models, including the LOGNGPD mixed-tail distribution model that is defined in “[Example 26.2: Defining a Model for Mixed-Tail Distributions](#)” on page 1856. The input sample is the same one that is used in that example.

```
/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);

/*----- Fit LOGNGPD model with PROC SEVSELECT by using -----
----- the Cramer-von Mises minimum distance estimator -----*/
proc sevselect data=mylib.testmixdist objective=cvmobj print=all;
  loss y;
  dist logngpd burr logn gpd;

  * Cramer-von Mises estimator (minimizes the distance *
  * between parametric and nonparametric estimates)   *;
  cvmobj = (_cdf_(y) - _edf_(y))**2;
run;
```

The OBJECTIVE= option in the PROC SEVSELECT statement specifies that the objective function cvmobj should be minimized. The programming statements compute the contribution of each observation in the input data table to the objective function cvmobj. The use of the keyword functions _CDF_ and _EDF_ makes the program applicable to all the distributions. When PROC SEVSELECT runs on a CAS server that has more than one worker node, each worker node evaluates the _EDF_ function by using a local sample of the input data.

Some of the key results that PROC SEVSELECT produces are shown in [Output 26.8.1](#). The “Model Selection” table indicates that all models converged. When you specify a custom objective function, the default selection criterion is the value of the custom objective function. The “All Fit Statistics” table indicates that LOGNGPD is the best distribution according to all the statistics of fit. Comparing the fit statistics of [Output 26.8.1](#) with those of [Output 26.2.1](#) indicates that the use of the Cramér–von Mises estimator has resulted in smaller values for all the EDF-based statistics of fit for all the models, which is expected from a minimum distance estimator.

Output 26.8.1 Summary of Cramér–von Mises Estimation**The SEVSELECT Procedure**

Model Selection				
Distribution	Converged	cvmobj	Selected	
Logngpd	Yes	0.12846	Yes	
Burr	Yes	0.22681	No	
Logn	Yes	0.16928	No	
Gpd	Yes	35.98574	No	

All Fit Statistics								
Distribution	cvmobj	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logngpd	0.12846*	3657*	3667*	3667*	3691*	0.86572*	1.04025*	0.12957*
Burr	0.22681	3724	3730	3730	3744	1.01660	2.27060	0.22826
Logn	0.16928	3908	3912	3912	3922	0.92926	2.01192	0.16956
Gpd	35.98574	5401	5405	5405	5415	9.85292	188.93299	35.99968

Asterisk (*) denotes the best model in the column.

Example 26.9: Predicting Mean and Value-at-Risk by Using Scoring Functions

If you work in the risk management department of an insurance company or a bank, then one of your primary applications of severity loss distribution models is to predict the value-at-risk (VaR) so that there is a very low probability of experiencing a loss value greater than the VaR. The probability level at which VaR is measured is prescribed by industry regulations such as Basel III and Solvency II. The VaR level is usually specified in terms of $(1 - \alpha)$, where $\alpha \in (0, 1)$ is the probability that a loss value exceeds the VaR. Typical VaR levels are 0.95, 0.975, and 0.995.

In addition to predicting the VaR, which is regarded as an estimate of the worst-case loss, businesses are often interested in predicting the average loss by estimating either the mean or median of the distribution.

The estimation of the mean and VaR combined with the scale regression model is a very potent tool for analyzing worst-case and average losses for various scenarios. For example, if the regressors that are used in a scale regression model represent some key macroeconomic and operational indicators, which are widely referred to as key risk indicators (KRIs), then you can analyze the VaR and mean loss estimates over various values for the KRIs to get a more comprehensive picture of the risk profile of your organization across various market and internal conditions.

This example illustrates the use of scoring functions to simplify the process of predicting the mean and VaR of scale regression models.

To compute the mean, you need to ensure that the function to compute the mean of a distribution is available in the function library. If you define and fit your own distribution and you want to compute its mean, then you need to use the FCMP procedure to define that function, and you need to use the CMPLIB= system option to specify the location of that function. For your convenience, the *dist_MEAN* function (which computes the mean of the *dist* distribution) is already defined in the library Sashelp.Svrtldist for each of the 10 predefined distributions. The following statements display the definitions of MEAN functions of all distributions. Note that the MEAN functions of the Burr, Pareto, and generalized Pareto distributions check the existence of the first moment for specified parameter values.

```

/*----- Definitions distribution functions that compute the mean -----*/
proc fcmp library=sashelp.svrtldist outlib=work.means.scalemod;
  function BURR_MEAN(x, Theta, Alpha, Gamma);
    if not(Alpha * Gamma > 1) then
      return (.); /* first moment does not exist */
    return (Theta*gamma(1 + 1/Gamma)*gamma(Alpha - 1/Gamma)/gamma(Alpha));
  endsub;
  function EXP_MEAN(x, Theta);
    return (Theta);
  endsub;
  function GAMMA_MEAN(x, Theta, Alpha);
    return (Theta*Alpha);
  endsub;
  function GPD_MEAN(x, Theta, Xi);
    if not(Xi < 1) then
      return (.); /* first moment does not exist */
    return (Theta/(1 - Xi));
  endsub;
  function IGAUSS_MEAN(x, Theta, Alpha);
    return (Theta);
  endsub;
  function LOGN_MEAN(x, Mu, Sigma);
    return (exp(Mu + Sigma*Sigma/2.0));
  endsub;
  function PARETO_MEAN(x, Theta, Alpha);
    if not(Alpha > 1) then
      return (.); /* first moment does not exist */
    return (Theta/(Alpha - 1));
  endsub;
  function STWEEDIE_MEAN(x, Theta, Lambda, P);
    return (Theta* Lambda * (2 - P) / (P - 1));
  endsub;
  function TWEEDIE_MEAN(x, P, Mu, Phi);
    return (Mu);
  endsub;
  function WEIBULL_MEAN(x, Theta, Tau);
    return (Theta*gamma(1 + 1/Tau));
  endsub;
quit;

```

For your further convenience, the *dist*_QUANTILE function (which computes the quantile of the *dist* distribution) is also defined in the Sashelp.Svrtldist library for each of the 10 predefined distributions. Because the MEAN and QUANTILE functions satisfy the definition of a distribution function as described in the section “[Formal Description](#)” on page 1830, PROC SEVSELECT can create scoring functions for them.

To illustrate the creation of scoring functions, let the following SAS statements simulate the input data set Work.Test_sev9 and load it into your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```

/*----- Simulate a lognormal sample -----*/
data testSevscore(keep=y x1-x5
  label='A Lognormal Sample Affected by Regressors');
  array x{*} x1-x5;

```



```

array b{6} _TEMPORARY_ (1 0.75 3 -1 0.25 5);
call streaminit(45678);
label y='Response Influenced by Regressors';
Sigma = 0.25;
do n = 1 to 500;
    Mu = b(1); /* log of base value of scale */
    do i = 1 to dim(x);
        x(i) = rand('NORMAL');
        Mu = Mu + b(i+1) * x(i);
    end;
    y = exp(Mu) * rand('LOGNORMAL')**Sigma;
    output;
end;
run;

/*----- Load data into your session -----*/
data mylib.testSevscore;
    set testSevscore;
run;

```

The following PROC SEVSELECT step fits all regression-friendly predefined distributions to the data in the data table mylib.TestSevscore and generates the scoring functions for the MEAN, QUANTILE, and other distribution functions:

```

/*----- Fit all distributions and generate scoring functions -----*/
proc sevselect data=mylib.testSevscore outest=mylib.est print=all;
    loss y;
    scalemodel x1-x5;
    dist _predefined_ stweedie;
    nloptions tech=quanew;
    outscorelib outlib=scorefuncs commonpackage;
run;

```

The OUTLIB= option in the OUTSCORELIB statement requests that the scoring functions be written to the library Work.Scorefuncs, and the COMMONPACKAGE option in the OUTSCORELIB statement requests that all the functions be written to the same package. Upon completion, PROC SEVSELECT sets the CMPLIB= system option to the following value:

```
(sashelp.svrtldist work.scorefuncs)
```

PROC SEVSELECT detects all the distribution functions that are available in the current CMPLIB= search path (which always includes the Sashelp.Svrtldist library) for the distributions that you specify in the DIST statement, and it creates the corresponding scoring functions. You can define any distribution function that has the desired signature to compute an estimate of your choice, include its library in the CMPLIB= system option, and then specify the OUTSCORELIB statement to generate the corresponding scoring functions. Specifying the COMMONPACKAGE option in the OUTSCORELIB statement causes the name of the scoring function to take the form SEV_*function-suffix*_dist. If you do not specify the COMMONPACKAGE option, PROC SEVSELECT creates a scoring function named SEV_*function-suffix* in a package named *dist*. You can invoke functions from a specific package only inside the FCMP procedure. If you want to invoke the scoring functions from a DATA step, then it is recommended that you specify the COMMONPACKAGE option when you specify multiple distributions in the DIST statement.

You can get an idea of the structure of the scoring functions that PROC SEVSELECT defines on your behalf by examining the following PROC FCMP code that the preceding PROC SEVSELECT step creates and

submits to define the scoring functions SEV_MEAN_LOGN and SEV_QUANTILE_IGAUSS:

```
proc fcmp library=(sashelp.svrtdist) outlib=work.scorefuncs.sevfit;
  function SEV_MEAN_LOGN(y, x{*});
    _logscale_=0;
    _logscale_ = _logscale_ + ( 7.64722238665070E-01 * x{1});
    _logscale_ = _logscale_ + ( 2.99209532114045E+00 * x{2});
    _logscale_ = _logscale_ + (-1.00788910539257E+00 * x{3});
    _logscale_ = _logscale_ + ( 2.58883411376730E-01 * x{4});
    _logscale_ = _logscale_ + ( 5.00927459151049E+00 * x{5});
    _logscale_ = _logscale_ + ( 9.95078801087240E-01);
    return (LOGN_MEAN(y, _logscale_, 2.31592996273020E-01));
  endsub;

  function SEV_QUANTILE_IGAUSS(y, x{*});
    _logscale_=0;
    _logscale_ = _logscale_ + ( 7.64581739155730E-01 * x{1});
    _logscale_ = _logscale_ + ( 2.99159055149487E+00 * x{2});
    _logscale_ = _logscale_ + (-1.00793496650702E+00 * x{3});
    _logscale_ = _logscale_ + ( 2.58870461417330E-01 * x{4});
    _logscale_ = _logscale_ + ( 5.00996884583256E+00 * x{5});
    _scale_ = 2.77854870199624E+00 * exp(_logscale_);
    return (IGAUSS_QUANTILE(y, _scale_, 1.815112308761660E+01));
  endsub;
quit;
```

PROC SEVSELECT creates and submits similar code to define several other scoring functions, such as SEV_PDF_dist and SEV_SDF_dist, for each distribution *dist* that you specify in the DIST statement.

To illustrate the use of scoring functions, let the following DATA step simulate the scoring data set Work.RegInput, where the values of regressors in each observation define one scenario:

```
/*--- Simulate scoring data ---*/
data reginput(keep=x1-x5);
  array x{*} x1-x5;
  call streaminit(9041);
  do n=1 to 10;
    do i = 1 to dim(x);
      x(i) = rand('NORMAL');
    end;
    output;
  end;
run;
```

The “All Fit Statistics” table in [Output 26.9.1](#) shows that the lognormal distribution’s scale model is the best and the inverse Gaussian’s scale model is a close second according to the likelihood-based statistics. So the following DATA step uses the scoring functions to compute the mean and VaR (at the 97.5% level) of the scale models of lognormal and inverse Gaussian distributions for each of the scenarios in the Work.RegInput data set:

```
/*--- Set VaR level ---*/
%let varLevel=0.975;

/*--- Compute scores (mean and var) for the ---
```

```

--- scoring data by using the scoring functions ---*/
data scores;
  array x{*} x1-x5;
  set reginput;

  igauss_mean = sev_mean_igauss(., x);
  igauss_var  = sev_quantile_igauss(&varLevel, x);
  logn_mean   = sev_mean_logn(., x);
  logn_var    = sev_quantile_logn(&varLevel, x);
run;

```

Output 26.9.1 Comparison of Fitted Scale Models for Mean and VaR Illustration**The SEVSELECT Procedure**

Distribution	All Fit Statistics						
	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Stweedie	460.65695	476.65695	476.95023	510.37382	10.44625	4770	37.08467
Burr	451.42238	467.42238	467.71565	501.13924	10.32782	4431	37.19808
Exp	1515	1527	1527	1552	8.85827	2062	23.98267
Gamma	448.28222	462.28222	462.50986	491.78448	10.42272	6068	37.19450
Igauss	444.44512	458.44512	458.67276	487.94738	10.33028	6257	37.30880
Logn	444.43670*	458.43670*	458.66434*	487.93895*	10.37035	6155	37.18553
Pareto	1515	1529	1530	1559	8.90327	2076	24.06174
Gpd	1515	1529	1529	1559	8.85827*	2062*	23.98267*
Weibull	527.28676	541.28676	541.51440	570.78902	10.48084	4947	36.36039

Asterisk (*) denotes the best model in the column.

The following DATA step accomplishes the same task by reading the parameter estimates in the OUTEST= data table that is created by the same PROC SEVSELECT step that created the scoring functions:

```

/*--- Compute scores (mean and var) for the ---
--- scoring data by using the OUTEST= data set ---*/
data scoresWithoutest(keep=x1-x5 igauss_mean igauss_var logn_mean logn_var);
  array _x{*} x1-x5;
  array _xparmIgauss_{5} _temporary_;
  array _xparmLogn_{5} _temporary_;

  retain _Theta0_ Alpha0;
  retain _Mu0_ Sigma0;
  *--- read parameter estimates for igauss and logn models ---*;
  if (_n_ = 1) then do;
    set mylib.est(where=(upcase(_MODEL_)='IGAUSS' and _TYPE_='EST'));
    _Theta0_ = Theta; Alpha0 = Alpha;
    do _i_=1 to dim(_x_);
      if (_x_(_i_) = .R) then _xparmIgauss_(_i_) = 0;
      else _xparmIgauss_(_i_) = _x_(_i_);
    end;

    set mylib.est(where=(upcase(_MODEL_)='LOGN' and _TYPE_='EST'));
    _Mu0_ = Mu; Sigma0 = Sigma;
    do _i_=1 to dim(_x_);

```

```

        if (_x(_i_) = .R) then _xparmLogn(_i_) = 0;
        else _xparmLogn(_i_) = _x(_i_);
    end;
end;

set reginput;

*--- predict mean and VaR for inverse Gaussian ---*;
* first compute X'*beta for inverse Gaussian *;
_xbeta_ = 0.0;
do _i_ = 1 to dim(_x_);
    _xbeta_ = _xbeta_ + _xparmIgauss(_i_) * _x(_i_);
end;
* now compute scale for inverse Gaussian *;
_SCALE_ = _Theta0_ * exp(_xbeta_);
igauss_mean = igauss_mean(., _SCALE_, Alpha0);
igauss_var = igauss_quantile(&varLevel, _SCALE_, Alpha0);

*--- predict mean and VaR for lognormal ---*;
* first compute X'*beta for lognormal*;
_xbeta_ = 0.0;
do _i_ = 1 to dim(_x_);
    _xbeta_ = _xbeta_ + _xparmLogn(_i_) * _x(_i_);
end;
* now compute Mu=log(scale) for lognormal *;
_MU_ = _Mu0_ + _xbeta_;
logn_mean = logn_mean(., _MU_, Sigma0);
logn_var = logn_quantile(&varLevel, _MU_, Sigma0);
run;

```

The “Values Comparison Summary” table in [Output 26.9.2](#) shows that the difference between the estimates that are produced by both methods is within the acceptable machine precision. However, if you compare the DATA step complexity of the methods, you clearly see that the method that uses the scoring functions is much easier because it saves a lot of programming effort. Further, new distribution functions are automatically discovered and converted to scoring functions by PROC SEVSELECT. That enables you to focus your efforts on writing the distribution function that computes your desired score, which needs to be done only once. Then, you can create and use the corresponding scoring functions multiple times with much less effort.

Output 26.9.2 Comparison of Mean and VaR Estimates of Two Scoring Methods

The COMPARE Procedure
 Comparison of WORK.SCORESWITHOUTEST with WORK.SCORES
 (Method=RELATIVE(0.0222), Criterion=1.0E-12)

NOTE: All values compared are within the equality criterion used. However, 40 of the values compared are not exactly equal.

References

Burr, I. W. (1942). “Cumulative Frequency Functions.” *Annals of Mathematical Statistics* 13:215–232.

- D'Agostino, R. B., and Stephens, M., eds. (1986). *Goodness-of-Fit Techniques*. New York: Marcel Dekker.
- Danielsson, J., de Haan, L., Peng, L., and de Vries, C. G. (2001). "Using a Bootstrap Method to Choose the Sample Fraction in Tail Index Estimation." *Journal of Multivariate Analysis* 76:226–248.
- Dunn, P. K., and Smyth, G. K. (2005). "Series Evaluation of Tweedie Exponential Dispersion Model Densities." *Statistics and Computing* 15:267–280.
- Frydman, H. (1994). "A Note on Nonparametric Estimation of the Distribution Function from Interval-Censored and Truncated Observations." *Journal of the Royal Statistical Society, Series B* 56:71–74.
- Gentleman, R., and Geyer, C. J. (1994). "Maximum Likelihood for Interval Censored Data: Consistency and Computation." *Biometrika* 81:618–623.
- Greenwood, M. (1926). "The Natural Duration of Cancer." In *Reports of Public Health and Related Subjects*, vol. 33, 1–26. London: Her Majesty's Stationery Office.
- Hill, B. M. (1975). "A Simple General Approach to Inference about the Tail of a Distribution." *Annals of Statistics* 3:1163–1173.
- Jørgensen, B. (1987). "Exponential Dispersion Models." *Journal of the Royal Statistical Society, Series B* 49:127–162. With discussion.
- Kaplan, E. L., and Meier, P. (1958). "Nonparametric Estimation from Incomplete Observations." *Journal of the American Statistical Association* 53:457–481.
- Klein, J. P., and Moeschberger, M. L. (1997). *Survival Analysis: Techniques for Censored and Truncated Data*. New York: Springer-Verlag.
- Klugman, S. A., Panjer, H. H., and Willmot, G. E. (1998). *Loss Models: From Data to Decisions*. New York: John Wiley & Sons.
- Koziol, J. A., and Green, S. B. (1976). "A Cramér–von Mises Statistic for Randomly Censored Data." *Biometrika* 63:466–474.
- Lai, T. L., and Ying, Z. (1991). "Estimating a Distribution Function with Truncated and Censored Data." *Annals of Statistics* 19:417–442.
- Lynden-Bell, D. (1971). "A Method of Allowing for Known Observational Selection in Small Samples Applied to 3CR Quasars." *Monthly Notices of the Royal Astronomical Society* 155:95–118.
- Rodriguez, R. N., and Balan, T. E. (2006). "Creating Statistical Graphics in SAS 9.2: What Every Statistical User Should Know." In *Proceedings of the Thirty-First Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. <http://www2.sas.com/proceedings/sugi31/192-31.pdf>.
- Turnbull, B. W. (1976). "The Empirical Distribution Function with Arbitrarily Grouped, Censored, and Truncated Data." *Journal of the Royal Statistical Society, Series B* 38:290–295.
- Tweedie, M. C. K. (1984). "An Index Which Distinguishes between Some Important Exponential Families." In *Statistics: Applications and New Directions—Proceedings of the Indian Statistical Institute Golden Jubilee International Conference*, edited by J. K. Ghosh and J. Roy, 579–604. Calcutta: Indian Statistical Institute.

Chapter 27

SMC Procedure

Contents

Overview: SMC Procedure	1902
Using CAS Sessions and CAS Engine Librefs	1903
Getting Started: SMC Procedure	1904
Autoregressive Stochastic Volatility Model	1904
Syntax: SMC Procedure	1915
Functional Summary	1915
PROC SMC Statement	1918
AUXMODEL Statement	1919
ENDSUBMIT Statement	1921
EVALUATE Statement	1921
FILTER Statement	1922
FORECAST Statement	1923
ID Statement	1925
INITSTATE Statement	1925
LEARN Statement	1928
MODEL Statement	1935
PARAMETERS Statement	1937
PRIOR Statement	1938
SCORE Statement	1940
SMOOTH Statement	1940
STATE Statement	1942
STATEVAR Statement	1945
SUBMITMODEL Statement	1945
VAR Statement	1960
Details: The SMC Procedure	1961
Sequential Monte Carlo Methods and State Space Models	1961
SMC Filtering	1964
SMC Smoothing	1969
SMC Forecasting	1971
SMC Evaluating	1972
SMC Learning	1974
SMC Algorithms for the General Case of SSMs	1974
Defining a State Space Model	1985
Distribution Argument Restriction	1990
Standard Distributions	1991
Missing Values	1995

Time Index	1995
Data Table Output	1996
ODS Table Names	2001
Examples: SMC Procedure	2002
Example 27.1: Stochastic Susceptible-Infected-Recovered Model with a Constant Transmission Rate	2002
Example 27.2: Stochastic Susceptible-Infected-Recovered Model with a Time-Varying Transmission Rate	2014
Example 27.3: Estimating the Effective Reproduction Number of COVID-19 Cases in Pennsylvania	2025
Example 27.4: Estimating the Dynamic Bayesian Network	2033
Example 27.5: The Fully Adapted Auxiliary Particle Filter Using a Gaussian ARCH Model	2046
References	2058

Overview: SMC Procedure

The sequential Monte Carlo (SMC) method has been widely applied in economics, finance, science, and engineering. It provides solutions to the inference and parameter estimation problems of the nonlinear/non-Gaussian state space models. For more information about the SMC method, see Doucet and Johansen (2011); Doucet, Godsill, and Andrieu (2000); Cappé, Godsill, and Moulines (2007). Here are a few examples in different fields:

- In risk management, asset pricing, and portfolio optimization, volatility forecasting is critical. The stochastic volatility model, as a special case of nonlinear non-Gaussian state space models, is one of the most popular tools for volatility forecasting. However, there was no closed-form solution for such a sophisticated model until the simulation-based solution, especially the SMC method, became available.
- In epidemiology, the SMC method can be used to estimate the stochastic susceptible-infected-recovered (SSIR) model, one of the most popular and accurate models used in epidemiology modeling.
- In engineering, the SMC method has been applied in navigation and object tracking.
- In Internet of Things (IoT), the SMC method can be used to estimate the states of the machines based on the observable information.
- In bioinformatics, the SMC method is used for haplotype inference.
- In marketing, the SMC method is used to model the effect of advertising on demand.

The SMC procedure supports the following:

- three types of SMC filtering algorithms: the sequential importance resampling algorithm, the bootstrap filter algorithm, and the auxiliary particle filter algorithm
- two types of SMC smoothing algorithms: the fixed-lag smoothing algorithm and the fixed-interval smoothing algorithm

- particle Markov chain Monte Carlo parameter estimation method
- proposal distribution for the initial state and transition distributions
- simulation and summary estimates of output of the filtering, smoothing, evaluating, and forecasting estimates
- output of the log likelihood up to any observation
- saving the model information and using it later to score other new data

PROC SMC requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: SMC Procedure

This section provides an example that the SMC procedure supports.

Autoregressive Stochastic Volatility Model

This example illustrates how to define an autoregressive stochastic volatility model in the SMC procedure and then estimate the parameters by using the particle Markov chain Monte Carlo (PMCMC) method.

The stochastic volatility model assumes that the volatility is a latent variable with unexpected innovation terms (Shephard 2005). The discrete-time autoregressive stochastic volatility (ARSV) model proposed by Taylor (1982) is one of the simplest stochastic volatility models. The ARSV model, in which the log variance follows a linear Gaussian process, is given as

$$y_t = \beta \exp\left(\frac{x_t}{2}\right) \xi_t, \quad \xi_t \stackrel{\text{iid}}{\sim} N(0, 1)$$

$$x_t = \phi x_{t-1} + \gamma \eta_t, \quad \eta_t \stackrel{\text{iid}}{\sim} N(0, 1), \quad \xi_t \perp \eta_t$$

where $\{y_t\}_{t \geq 1}$ denotes the log daily return series and $\{\beta \exp(\frac{x_t}{2})\}_{t \geq 1}$ denotes the daily volatility series. Suppose that the true parameter values are $\phi = 0.9$, $\gamma = 0.4$, $\beta = 1$, and $x_0 = 0.45$. In order to bring these initial settings in line with the real data, here the $\{y_t\}_{t \geq 1}$ denotes the 100-times-magnified log daily return series; that is, $y_t = 100 \ln \frac{S_t}{S_{t-1}}$, where S_t is the asset price at day t ($t \geq 1$).

The following statements simulate 280 observations by using the given stochastic volatility model:

```
/*--- parameter values ---*/
%let phi = 0.9;
%let gamma = 0.4;
%let beta = 1;

/*--- Simulate a sample for the stochastic volatility model ---*/
data asv_all;
  xlag = 0.45;
  do t = 1 to 280;
    eps = normal(1);
    x = &phi*xlag + &gamma*eps;
    eta = normal(1);
    y = &beta*exp(x/2)*eta;
    output;
    xlag = x;
  end;
run;
```

The following DATA steps load the data set asv_all into a data table in your session that is associated with the mylib libref. The DATA steps assume that your libref is named mylib, but you can substitute any appropriately defined libref. The data set asv_all is split such that the first 275 observations are used as the training set and the last 5 observations are used as the test set.

```

data asv_train;
  set asv_all;
  if (t le 275) then output;
run;

data asv_test;
  set asv_all;
  if (t > 275) then output;
run;

data mylib.asv_train;
  set asv_train;
run;

```

The model specification defines the transition and measurement distributions. For the initialization distribution of the state variable, the normal distribution whose mean is 0 and variance is 4 is used. Moreover, the prior distributions for the parameters are defined as

$$\begin{aligned}\phi &\sim \text{UNIFORM}(0, 1) \\ \gamma^2 &\sim \text{IGAMMA}(0.01, 0.01) \\ \beta^2 &\sim \text{IGAMMA}(0.01, 0.01)\end{aligned}$$

The following statements use PROC SMC to define the preceding ARSV model and then use the PMCMC method to estimate the three parameters for the simulated data set:

```

proc smc data=mylib.asv_train seed=1234;
  id time = t;
  var y;
  statevar x;
  prior phi~uniform(0,1);
  prior gamma_sq beta_sq~igamma(0.01,0.01);
  parm phi=0.6 gamma_sq=0.9 beta_sq=1.5;
  initstate x~normal(0,2);
  state x~normal(x.11*phi, sqrt(gamma_sq));
  model y~normal(0, sqrt(beta_sq)*exp(x/2));
  auxmodel y~normal(0, sqrt(beta_sq)*exp(phi*x.11/2));
  learn nparticle=256 algorithm=apf method=PMCMC(nsample=25000
    nchain=8 nbi=5000 thin=4 seed=1
    outpost=mylib.out samset samsum
    sampler=rwm(ntu=250)
    statistics=all diagnostics=all);
  score outmodel=mylib.asvmodel;
run;

```

For the three parameters, ϕ is represented by phi, γ^2 is represented by gamma_sq, and β^2 is represented by beta_sq. A state variable, x, an a dependent (observation) variable, y, are also specified according to the model specification.

The model information and parameter learning results are stored in the mylib.asvmodel. As for the PMCMC, the likelihood is estimated by the particle filter and the prior density is specified in the PRIOR statement.

The summary tables of the model information are shown in [Figure 27.1](#).

Figure 27.1 Model Information Summary

The SMC Procedure			
State Space Model Information			
Number of Parameters			3
Number of State Variables			1
Number of Dependent Variables			1
Number of Independent Variables			0
Seed Value			1234
Number of Observations			275
Number of Missing Observations			0
Initial Parameters Information			
Parameter	Initial Value	Lower Bound	Upper Bound
phi	0.60000	0	1.00000
gamma_sq	0.90000	0	Infy
beta_sq	1.50000	0	Infy

The summary tables of the posterior samples are shown in [Figure 27.2](#).

Figure 27.2 Posterior Summary

MCMC Sampler Settings	
MCMC Sampling Method	RWM
Random Seed	1
Transformed Sampling	False
Number of Chains	8
Number of Sampling Iterations	25000
Number of Burn-in Iterations	5000
Thinning Interval	4
Number of Tuning Iterations per Tuning Stage	250
Number of Tuning Stages	24
Target Metropolis Acceptance Rate	0.234
Initial RWM Proposal Scale	2.38
Tuning Weight	0.75
Acceptance Tolerance	0.075

The correlation and covariance matrices of the parameter samples that are generated by the PMCMC method are shown in [Figure 27.3](#).

Figure 27.3 Correlation and Covariance Matrices

Posterior Covariance Matrix			
Parameter	phi	gamma_sq	beta_sq
phi	0.006192	-0.00985	0.004117
gamma_sq	-0.00985	0.026624	-0.00822
beta_sq	0.004117	-0.00822	0.074281

The autocorrelation statistics, Heidelberger-Welch diagnostics, effective sample size (ESS) diagnostics, and Monte Carlo Standard Error (MCSE) diagnostics for each parameter are shown in [Figure 27.4](#).

Figure 27.4 Other Bayesian Analysis Tables

The SMC Procedure

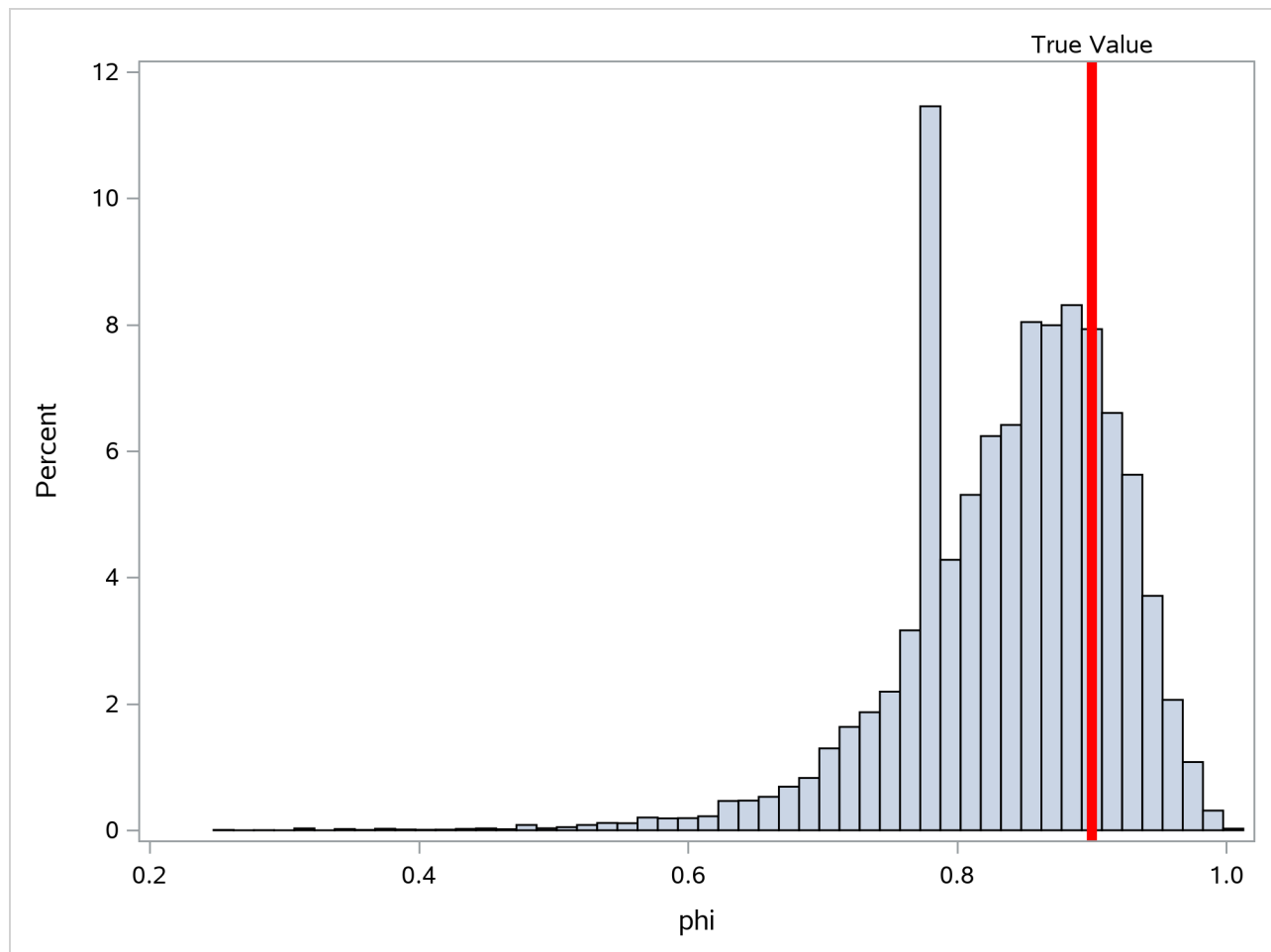
MCMC Diagnostic Summaries						
Parameter		MCSE	MCSE/SD	ESS	Autocorrelation	
					Time	ESS/N
phi	*	0.00154	0.0196	2607.9	19.1724	0.0522
gamma_sq	*	0.00517	0.0317	998.0	50.1010	0.0200
beta_sq	*	0.00940	0.0345	841.0	59.4525	0.0168
*Autocorrelation Remains						

Autocorrelation Diagnostics				
Parameter	Lag 1	Lag 5	Lag 10	Lag 50
phi	0.7387	0.3128	0.1604	0.0146
gamma_sq	0.7633	0.4115	0.2826	0.1005
beta_sq	0.7611	0.4061	0.2666	0.1165

Heidelberger-Welch Diagnostics					
Parameter	Stationarity Test			Half-Width Test	
	Largest				
	Chains Tested	Tests Passed	Iterations Discarded	Chains Tested	Tests Passed
phi	8	8	1875	8	8
gamma_sq	8	7	2500	7	7
beta_sq	8	8	1250	8	8

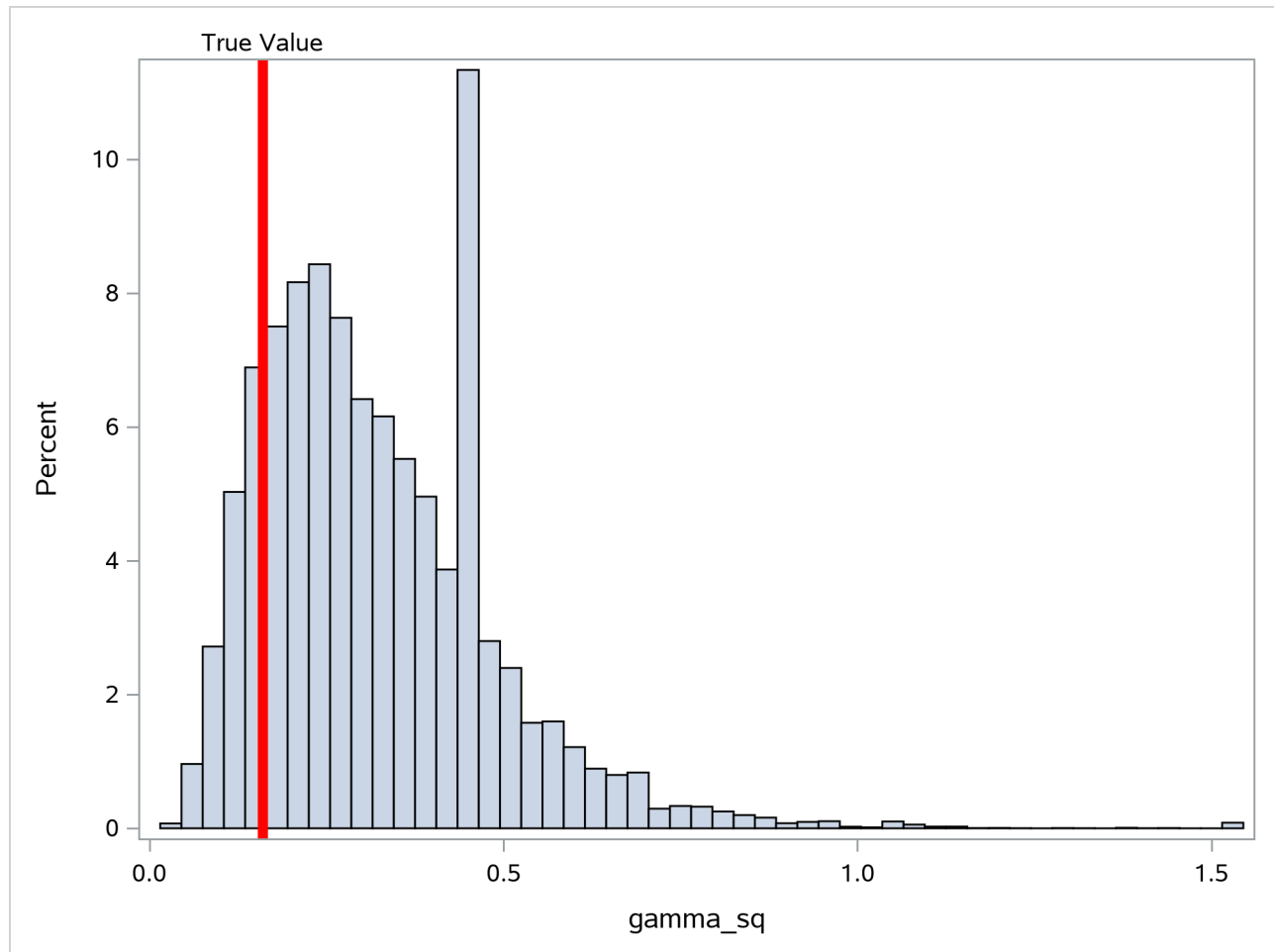
The following statements draw the histogram of the posterior samples of phi, which is shown in [Figure 27.5](#):

```
proc sgplot data=mylib.out;
  histogram phi;
  refline 0.9 /axis=x lineattrs=(thickness=5 color=red) label="True Value";
run;
```

Figure 27.5 Histogram of Posterior Samples of ϕ 

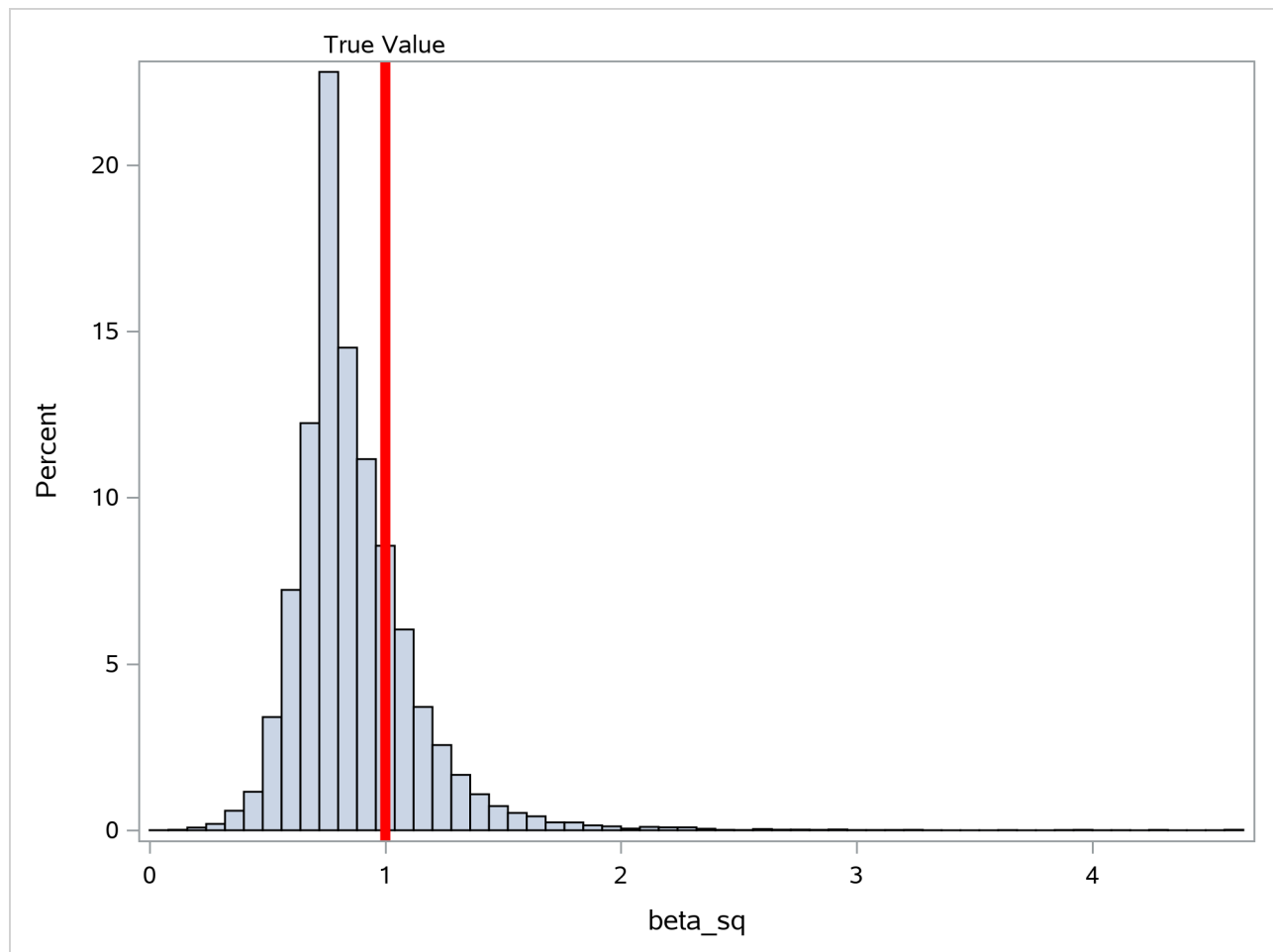
The following statements draw the histogram of the posterior samples of `gamma_sq`, which is shown in Figure 27.6:

```
proc sgplot data=mylib.out;
  histogram gamma_sq;
  refline 0.16 /axis=x lineattrs=(thickness=5 color=red) label="True Value";
run;
```

Figure 27.6 Histogram of Posterior Samples of gamma_sq

The following statements draw the histogram of the posterior samples of beta_sq, which is shown in Figure 27.7:

```
proc sgplot data=mylib.out;
  histogram beta_sq;
  refline 1 /axis=x lineattrs=(thickness=5 color=red) label="True Value";
run;
```

Figure 27.7 Histogram of Posterior Samples of beta_sq

You can apply the model information and parameter estimates from the data table `mylib.asvmodel` to the original data set by specifying the `INMODEL=` option in the `SCORE` statement. The posterior mean acquired by the PMCMC method are used as the parameters in the smoothing and forecasting process. The following statements generate the smoothing and forecasting estimates:

```
proc smc data=mylib.asv_train seed=1 nthreads=32;
  smooth nparticle=10240 parm=median alg=sir method=fixedlag(25)
    out (pct=2.5,97.5)=mylib.asvsm;
  forecast nparticle=10240 parm=median alg=sir lead=5
    out (pct=2.5,97.5)=mylib.asvfc;
  score inmodel=mylib.asvmodel;
run;
```

The following statements sort the summary table of the smoothing estimates of the state variable, `x`, in the session and create a relevant data table, `smoothRes`, in the `Work` library to record the smoothing estimates and actual values:


```

data asvsm; set mylib.asvsm(where=(VariableID=1 and TimeID>=1)); run;
proc sort data=asvsm; by timeId; run;

data smoothRes;
  set asvsm (keep=timeId Q2 P_1 P_2);
  set asv_train (keep=x t);
run;

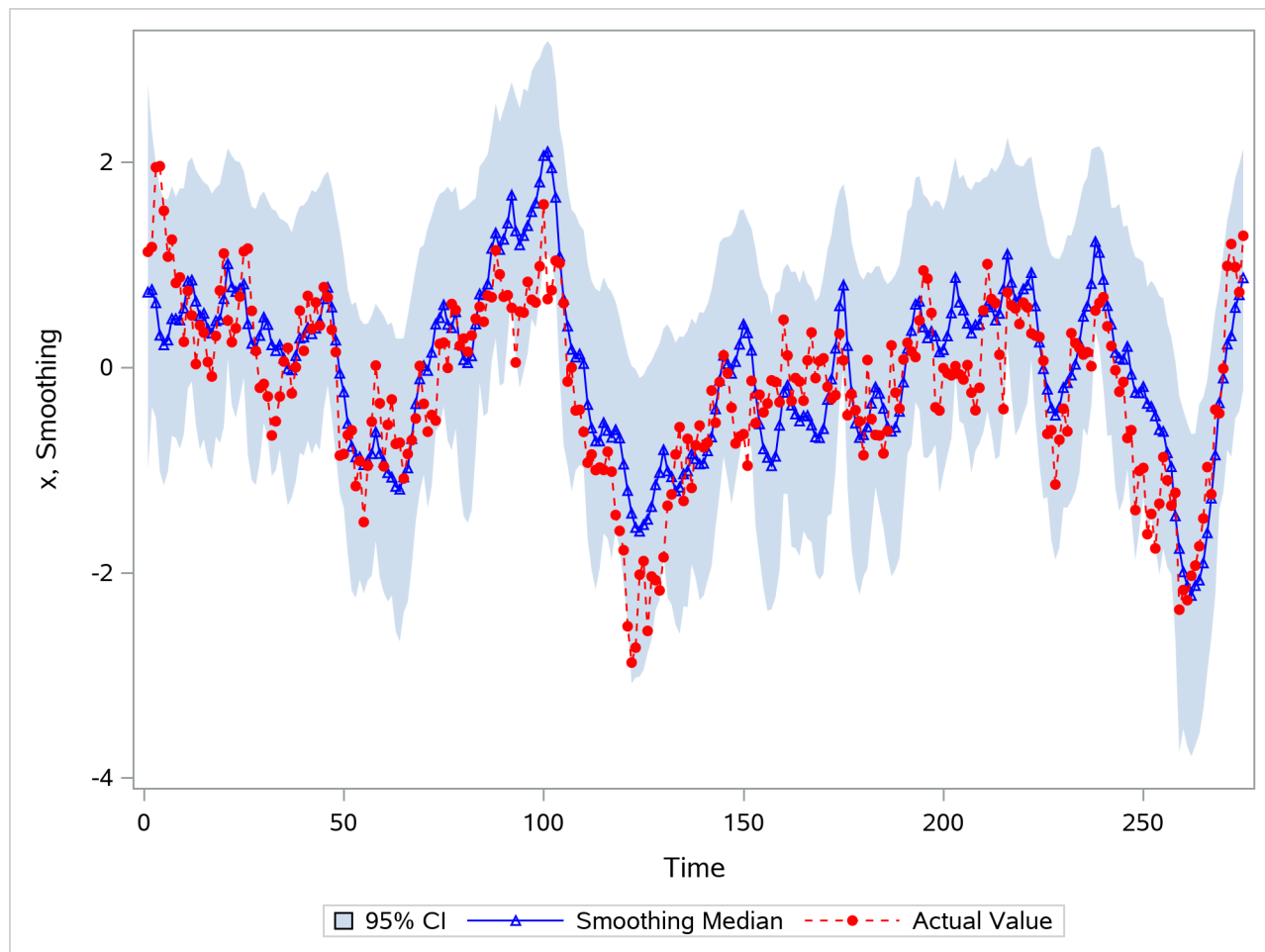
```

The following statements draw the plot of the smoothing median estimates and the actual values of x . The output is shown in [Figure 27.8](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates.

```

proc sgplot data=smoothRes;
  band x=TimeID upper=P_2 lower=P_1 / transparency=.3 legendlabel='95% CI';
  series x=TimeId y=Q2 /legendlabel='Smoothing Median'
    markers markerattrs=(color=blue size=5 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  series x=TimeId y=x /legendlabel='Actual Value'
    markers markerattrs=(color=red size=5 symbol=circlefilled)
    lineattrs=(color=red pattern=shortdash);
  yaxis label='x, Smoothing';
  xaxis label='Time';
run;

```

Figure 27.8 Smoothing Estimates vs. Actual Values, x

The following statements sort the summary table of the forecasting estimates of x in the session and create a relevant data table, `forecastRes_x`, in the Work library to record the forecasting estimates and actual values in the forecasting stage:

```
data asvfc_x; set mylib.asvfc(where=(VariableID=1)); run;
proc sort data=asvfc_x; by _t; run;

data forecastRes_x;
  set asvfc_x (keep=Q2 P_1 P_2 _t);
  set asv_test (keep=x t);
run;
```

The following statements draw the plot of the forecasting median estimates and the actual values of x . The output is shown in [Figure 27.9](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the forecasting estimates.

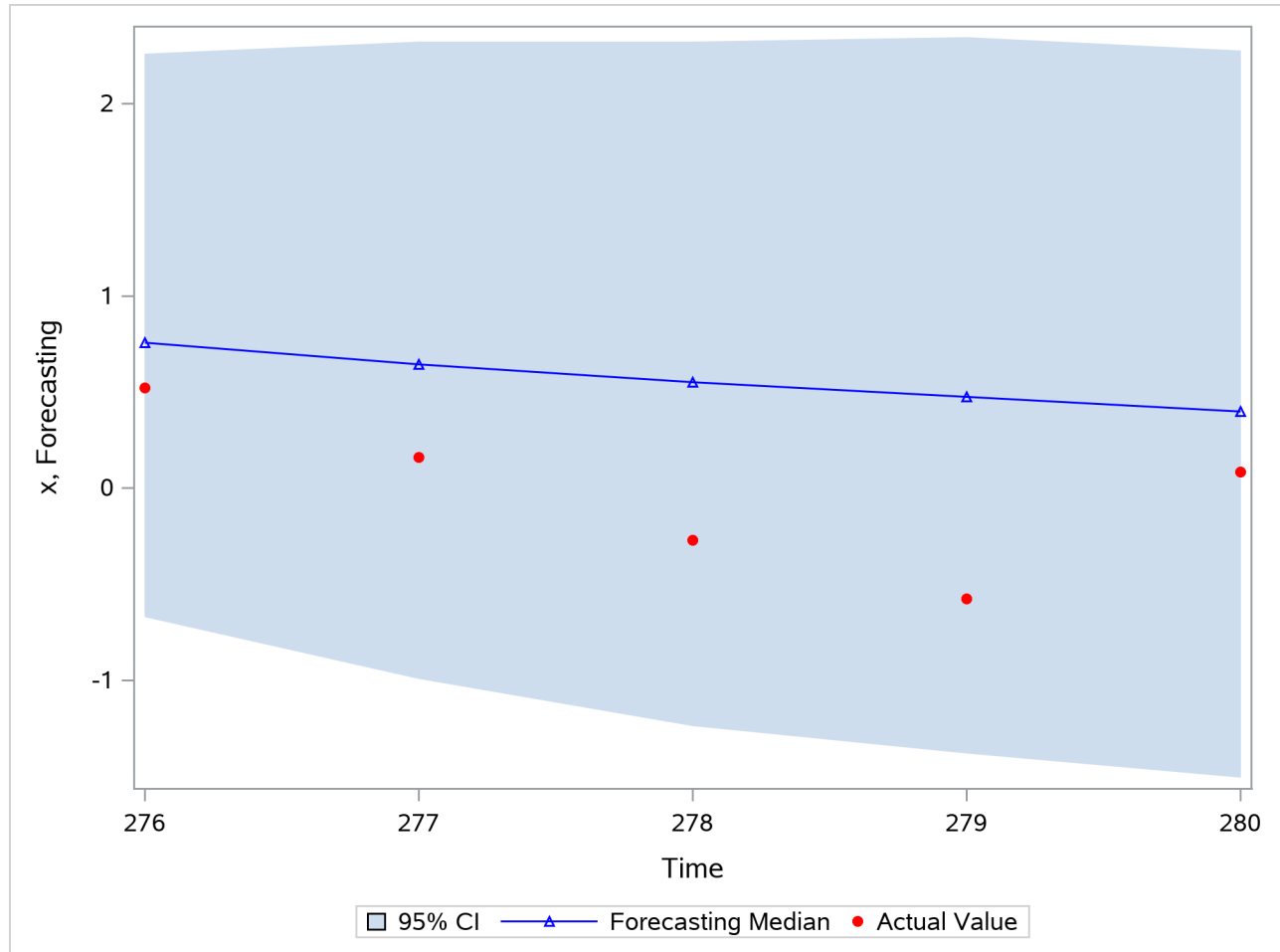
```
proc sgplot data=forecastRes_x;
  band x=t upper=P_2 lower=P_1 /transparency=.3 legendlabel='95% CI';
  series x=t y=Q2 /legendlabel='Forecasting Median'
    markers markerattrs=(color=blue size=5 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
```

```

scatter x=t y=x /legendlabel='Actual Value'
      markerattrs=(color=red size=5 symbol=circlefilled);
xaxis integer label='Time';
yaxis label='x, Forecasting';
run;

```

Figure 27.9 Forecasting Estimates vs. Actual Values, x



The following statements sort the summary table of the forecasting estimates of the dependent variable, *y*, in the session and create a relevant data table, *forecastRes_y*, in the Work library to record the forecasting estimates and actual values in the forecasting stage:

```

data asvfc_y; set mylib.asvfc(where=(VariableID=2)); run;
proc sort data=asvfc_y; by _t; run;

data forecastRes_y;
  set asvfc_y (keep=Q2 P_1 P_2 _t);
  set asv_test (keep=y _t);
run;

```

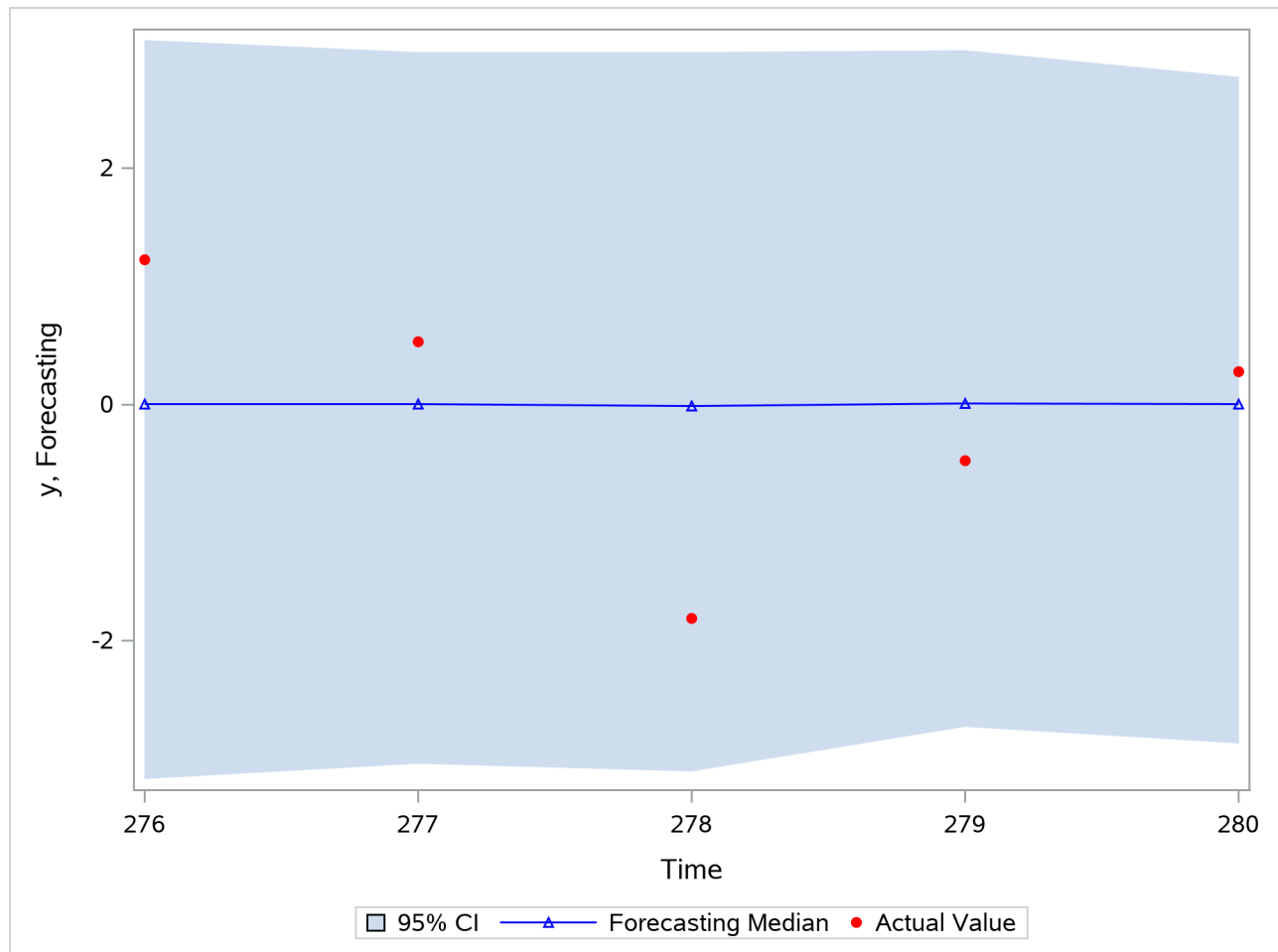
The following statements draw the plot of the forecasting median estimates and the actual values of *y*. The output is shown in [Figure 27.10](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the forecasting estimates.

```

proc sgplot data=forecastRes_y;
  band x=t upper=P_2 lower=P_1 /transparency=.3 legendlabel='95% CI';
  series x=t y=Q2 /legendlabel='Forecasting Median'
    markers markerattrs=(color=blue size=5 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  scatter x=t y=y /legendlabel='Actual Value'
    markerattrs=(color=red size=5 symbol=circlefilled);
  xaxis integer label='Time';
  yaxis label='y, Forecasting';
run;

```

Figure 27.10 Forecasting Estimates vs. Actual Values, y



Syntax: SMC Procedure

The following statements are available in the SMC procedure. Items within angle brackets (<>) are optional.

```

PROC SMC options ;
    AUXMODEL dependent-variable ~ distribution ;
    ENDSUBMIT ;
    EVALUATE < options > ;
    FILTER < options > ;
    FORECAST < options > ;
    ID < TIME= > variable ;
    INITSTATE state-variable ~ distribution < / option > ;
    LEARN < options > ;
    MODEL dependent-variable ~ distribution ;
    PARAMETERS parameter-specification < ,... , parameter-specification > < / options > ;
    PRIOR parameter-list ~ distribution ;
    SCORE options ;
    SMOOTH < options > ;
    STATE state-variable ~ distribution < / option > ;
    STATEVAR variable-list ;
    SUBMITMODEL dependent-variables < =independent-variables > < / options > ;
    VAR variable-list ;

```

Functional Summary

The statements and options available in the SMC procedure are summarized in [Table 27.1](#). The sections after the table provide a description of each statement in alphabetical order, following the PROC SMC statement.

Table 27.1 Functional Summary

Description	Statement	Option
Input Data Table Options		
Specifies the input data table	PROC SMC	DATA=
Specifies the input data table for model information	SCORE	INMODEL=
Output Data Table Options		
Writes the model information to the specified output data set	SCORE	OUTMODEL=
Writes the evaluation estimates to the specified output data set	EVALUATE	OUT=
Writes the summary of the filtering estimates to the specified output data set	FILTER	OUT=
Writes the filtering estimates to the specified output data set	FILTER	OUTSIM=

Table 27.1 *continued*

Description	Statement	Option
Writes the summary of the forecasting estimates to the specified output data set	FORECAST	OUT=
Writes the forecasting estimates to the specified output data set	FORECAST	OUTSIM=
Writes the summary of the smoothing estimates to the specified output data set	SMOOTH	OUT=
Writes the smoothing estimates to the specified output data set	SMOOTH	OUTSIM=
SMC Common Options		
Specifies the number of particles to use in the SMC algorithms	PROC SMC	NPARTICLE=
Specifies the random seed to use in the SMC algorithms	PROC SMC	SEED=
Variable and Parameter Declaration Options		
Specifies the names and initial values of the model parameters	PARAMETERS	
Specifies the lower bounds of the model parameters	PARAMETERS	LOWER=
Specifies the upper bounds of the model parameters	PARAMETERS	UPPER=
Specifies the names of state variables	STATEVAR	
Specifies the names of dependent and independent (exogenous) variables	VAR	
Model Specification Options		
Specifies the initialization distribution of the state variable	INITSTATE	
Specifies the proposal initialization distribution of the state variable	INITSTATE	PROPOSAL=
Specifies the transition distribution of the state variable	STATE	
Specifies the proposal transition distribution of the state variable	STATE	PROPOSAL=
Specifies the measurement distribution of the dependent variable	MODEL	
Specifies the auxiliary distribution of the dependent variable	AUXMODEL	
Specifies the name of the model of interest	SUBMITMODEL	NAME=
Specifies the number of parameters	SUBMITMODEL	NPARAM=
Specifies the number of state variables	SUBMITMODEL	NSTATE=
Specifies the beginning of program statements	SUBMITMODEL	
Specifies the end of program statements	ENDSUBMIT	

Table 27.1 *continued*

Description	Statement	Option
ID Variables		
Specifies the variable that identifies the time or sequence	ID	TIME=
Parameter Estimation Options		
Specifies the prior distribution and range of the parameters	PRIOR	
Specifies the number of particles to use in the parameter estimation process	LEARN	NPARTICLE=
Specifies the particle filter algorithm to use in the parameter estimation process	LEARN	ALGORITHM=
Specifies the parameter estimation method and options	LEARN	METHOD=PMCMC
Inference Options		
Specifies the particle filter algorithm to use to solve the evaluation problem	EVALUATE	ALGORITHM=
Specifies the number of particles to use in the evaluation process	EVALUATE	NPARTICLE=
Specifies the type of parameter values from the posterior distribution to use in the evaluation process	EVALUATE	PARAM=
Specifies the particle filter algorithm to use to solve the filtering problem	FILTER	ALGORITHM=
Specifies the number of particles to use in the filtering process	FILTER	NPARTICLE=
Specifies the type of parameter values from the posterior distribution to use in the filtering process	FILTER	PARAM=
Specifies the particle filter algorithm to use to get particles before the (multi-step) forecasts	FORECAST	ALGORITHM=
Specifies how many periods to forecast	FORECAST	LEAD=
Specifies the number of particles to use in the forecasting process	FORECAST	NPARTICLE=
Specifies the type of parameter values from the posterior distribution to use in the forecasting process	FORECAST	PARAM=
Specifies that particles are to be resampled before forecasting	FORECAST	RESAMPLE
Specifies the particle filter algorithm to use in the smoothing process	SMOOTH	ALGORITHM=
Specifies the number of particles to use in the smoothing process	SMOOTH	NPARTICLE=

Table 27.1 *continued*

Description	Statement	Option
Specifies the smoothing method	SMOOTH	METHOD=
Specifies the type of parameter values from the posterior distribution to use in the smoothing process	SMOOTH	PARM=

PROC SMC Statement

PROC SMC *options* ;

The PROC SMC statement invokes the SMC procedure. You must specify the following *option*:

DATA=*libref.data-table*

names the input data table for PROC SMC to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1903.

data-table specifies the name of the input data table.

You can specify the following *options*:

NPARTICLE=*number*

specifies the number of particles to use in the EVALUATE, FILTER, FORECAST, LEARN, and SMOOTH statements if their own corresponding option is not specified. By default, NPARTICLE=1000.

SEED=*number*

specifies a nonnegative integer to use as the seed for generating random number sequences. You can use this option to replicate results from different runs. By default, SEED=1.

AUXMODEL Statement

AUXMODEL *dependent-variable* $\sim$ *distribution* ;

The AUXMODEL statement specifies the auxiliary conditional distribution of the dependent variable given the parameters, the lagged state variables, the lagged dependent variables, and the independent variables; that is, $\hat{p}_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$, $s \geq 0$, $r \geq 1$, $p \geq 1$. For more information about the auxiliary conditional distribution in the state space model, see the section “[Auxiliary Particle Filter](#)” on page 1966.

The AUXMODEL statement is ignored if you specify the SUBMITMODEL statement. The AUXMODEL statement has the following requirements if you omit the SUBMITMODEL statement:

- If you specify the ALGORITHM=APF option in the EVALUATE, FILTER, FORECAST, LEARN, or SMOOTH statement, each dependent variable must have a corresponding AUXMODEL statement.
- Multiple AUXMODEL statements are permitted.

The AUXMODEL statement must have the following components in the order shown:

1. *Dependent variable*: A single dependent variable from the input data table.
2. *Tilde*: A tilde ($\sim$) symbol between the dependent variable and the distribution.
3. *Distribution*: A distribution that is specified in the form *distribution-name*(*distribution-arguments*). The available *distribution-name* and *distribution-arguments* in the AUXMODEL statements are displayed in [Table 27.2](#). For the description of each distribution, see the section “[Standard Distributions](#)” on page 1991.

Table 27.2 Distribution Names and Arguments

Distribution Name	Distribution Arguments	Definitions
BETA	a, b	Beta distribution with arguments shape parameter a and shape parameter b
CAUCHY	l, s	Cauchy distribution with arguments location l and scale s
GAMMA	a, b	Gamma distribution with arguments shape a and scale b
LOGN	m, s	Log-normal distribution with arguments mean m and standard deviation s
NORMAL N	m, s	Normal distribution with arguments mean m and standard deviation s
POISSON	λ	Poisson distribution with argument mean λ
T	m, s, ν	Student's t distribution with arguments mean m , standard deviation s , and degrees of freedom ν
UNIFORM	l, u	Uniform distribution with arguments min l and scale u

The *distribution-arguments* are specified by expressions that are enclosed in parentheses and separated by commas. Such expressions must follow any SAS programming statements. To avoid the possible computation errors, you should make some adjustments to the expressions. For more information about the adjustments, see the section “[Distribution Argument Restriction](#)” on page 1990. The number of expressions is equal to the number of arguments of the given distribution. For the AUXMODEL statement, the variables that can appear in the expressions must be selected from the following list:

- the model parameters
- the lagged state variables
- the lagged dependent variables
- the current and lagged independent variables
- the actual time index, `_time`

For more information about the actual time index, see the section “[Time Index](#)” on page 1995. For more information about the parameters, state variables, and dependent/independent variables, see the sections “[PARAMETERS Statement](#)” on page 1937, “[STATEVAR Statement](#)” on page 1945, and “[VAR Statement](#)” on page 1960, respectively.

Moreover, the following conditions must be satisfied in the AUXMODEL statement:

- The maximum lag order of all the state variables in the *distribution-arguments* of all AUXMODEL statements must be less than or equal to r , where r is defined in the section “[Method 1: Maximum Lag Order of State Variables, \$r\$](#) ” on page 1986.
- The maximum lag order of all the dependent variables in the *distribution-arguments* of all AUXMODEL statements must be less than or equal to p , where p is defined in the section “[Method 1: Maximum Lag Order of Dependent Variables, \$p\$](#) ” on page 1986.
- The maximum lag order of all the independent variables in the *distribution-arguments* of all AUXMODEL statements must be less than or equal to s , where s is defined in the section “[Method 1: Maximum Lag Order of Independent Variables, \$s\$](#) ” on page 1986.
- The dependent variables in all AUXMODEL statements must not have duplicates; that is, each dependent variable should appear only once as the *dependent-variable* in all AUXMODEL statements.

For example, suppose that the auxiliary distributions of the dependent variables y_1 and y_2 are described in the following equations:

$$y_{1,t} = \exp\left(\phi \frac{x_{1,t-1}}{2}\right) \xi_t, \quad \xi_t \stackrel{\text{iid}}{\sim} N(0, 1)$$

$$y_{2,t} = \beta x_{2,t-1} + \gamma \eta_t \quad \eta_t \stackrel{\text{iid}}{\sim} N(0, 1), \quad \xi_t \perp \eta_t$$

Then the following statements define these auxiliary distributions:

```
auxmodel y1 ~ normal(0, exp(phi * x1.l1/2));
auxmodel y2 ~ normal(beta*x2.l1, gamma);
```

where ϕ , β , and γ are model parameters; y_1 and y_2 are current dependent variables; and $x1.l1$ and $x2.l1$ are the first-order lagged state variables, which are constructed by appending the suffix “.11” to the state variables “ $x1$ ” and “ $x2$ ” (see the section “[Method 1: Lagged Variables](#)” on page 1985).

ENDSUBMIT Statement

ENDSUBMIT ;

The ENDSUBMIT statement is required in order to terminate the user-defined program statements in the PROC SMC statement block whenever you specify the SUBMITMODEL statement without the FILE= option. Do not specify the ENDSUBMIT statement if you specify the FILE= option in the SUBMITMODEL statement. If you omit the FILE= option and you do not include an ENDSUBMIT statement to accompany your SUBMITMODEL statement, then the PROC SMC step ends with an error.

EVALUATE Statement

EVALUATE < options > ;

The EVALUATE statement specifies options that are related to the evaluation, which is the log-likelihood calculation (see the section “[SMC Evaluating](#)” on page 1972). If you specify the LEARN statement, the model parameters in the requested evaluation process are set to the posterior mean of the corresponding model parameters; otherwise, the parameters with initial values that you specify in the PARAMETERS statement are used in the evaluation. You can specify the following *options*:

ALGORITHM=*algorithm*

ALG=*algorithm*

specifies the particle filter algorithm to use in the evaluation. You can specify the following *algorithms*:

APF<(ADP)> specifies the auxiliary particle filter (APF) algorithm. The ADP suboption specifies the fully adapted auxiliary particle filter algorithm.

BF specifies the bootstrap filter (BF) algorithm.

SIR<(TH | THRESH | THRESHOLD=*number*)> specifies the sequential importance resampling (SIR) algorithm. The TH= option specifies the threshold (a value between 0 and 1) of the resampling step in the SIR algorithm. The resampling step occurs when the effective sample size is less than the product of the threshold and the number of particles. By default, TH=0.8.

By default, ALGORITHM=SIR.

NPARTICLE=*number*

specifies the number of particles (samples) to use in the evaluation. If you omit this option, the evaluation uses the value of the NPARTICLE= option in the PROC SMC statement.

OUT=*libref.data-table*

writes the evaluation estimates to an output data set. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1903.

PARM=MEAN | MEDIAN

specifies which parameter values from the posterior distribution to use for the evaluation problem when you also specify the LEARN statement. If you omit the LEARN statement, this option is ignored. You can specify one of the following values:

MEAN	solves the evaluation problem by using the estimated posterior means of the parameters.
MEDIAN	solves the evaluation problem by using the estimated posterior medians of the parameters.

By default, PARM=MEAN.

FILTER Statement

FILTER < options > ;

The FILTER statement specifies options that are related to the filtering problem, which is to find the probability distribution of the current state conditional on the observations up to now (see the section “[SMC Filtering](#)” on page 1964). If you specify the LEARN statement, the model parameters in the requested filtering process are set to the posterior mean or median of the corresponding model parameters; otherwise, the parameters whose initial values are specified in the PARAMETERS statement are used in the filtering process. You can specify the following *options*:

ALGORITHM=*algorithm*

ALG=*algorithm*

specifies the particle filter algorithm to use for the filtering problem. You can specify the following *algorithms*:

APF<(ADP)> specifies the auxiliary particle filter (APF) algorithm. The ADP suboption specifies the fully adapted auxiliary particle filter algorithm.

BF specifies the bootstrap filter (BF) algorithm.

SIR<(TH | THRESH | THRESHOLD=*number*)> specifies the sequential importance resampling (SIR) algorithm. The TH= option specifies the threshold (a number between 0 and 1) of the resampling step in the SIR algorithm. The resampling step occurs when the effective sample size is less than the product of the threshold and the number of particles. By default, TH=0.8.

By default, ALGORITHM=SIR.

NPARTICLE=*number*

specifies the number of particles to use for the filtering problem. If you omit this option, the filtering process uses the value of the NPARTICLE= option in the PROC SMC statement.

OUT<(PERCENTILES=*numeric-list*)>=*libref.data-table*

OUT<(PERCENTILE=*numeric-list*)>=*libref.data-table*

OUT<(PERCENT=*numeric-list*)>=*libref.data-table*

OUT<(PERC=*numeric-list*)>=*libref.data-table*

OUT<(PCT=*numeric-list*)>=*libref.data-table*

writes the summary of the filtering estimates to the specified output data table. The weighted average, weighted standard deviation, minimum value, first quantile, second quantile, third quantile, maximum value, and percentile points of each state variable at each time step are displayed. The optional PERCENTILES=*numeric-list* specifies the percentile points in the summary table of the filtering estimates, where each *value* in the *numeric-list* must be between 0 and 100 and values are separated by commas or spaces. By default, PERCENTILES=2.5, 97.5, which yields the 2.5th and 97.5th percentile points, respectively. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1903.

OUTSIM=*libref.data-table*

writes the filtering estimates to the specified output data table. The particle values with the corresponding normalized weights at each time step are displayed. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1903.

PARM=MEAN | MEDIAN

specifies which parameter values from the posterior distribution to use for the filtering distribution when you also specify the LEARN statement. If you omit the LEARN statement, this option is ignored. You can specify one of the following values:

MEAN	constructs the filtering distribution by using the estimated posterior means of the parameters.
MEDIAN	constructs the filtering distribution by using the estimated posterior medians of the parameters.

By default, PARM=MEAN.

FORECAST Statement

FORECAST < options > ;

The FORECAST statement specifies options that are related to the forecasting problem, which is to find the probability distribution of the state and the dependent variables. If you specify the LEARN statement, the model parameters in the requested forecasting process are set to the posterior mean of the corresponding model parameters; otherwise, the parameters whose initial values are specified in the PARAMETERS statement are used in the forecasting process. You can specify the following *options*:

ALGORITHM=*algorithm*

ALG=*algorithm*

specifies the particle filter algorithm to use for the forecasting problem. You can specify the following *algorithms*:

APF<(ADP)> specifies the auxiliary particle filter (APF) algorithm. The ADP suboption specifies the fully adapted auxiliary particle filter algorithm.

BF specifies the bootstrap filter (BF) algorithm.

SIR<(TH | THRESH | THRESHOLD=*value*)> specifies the sequential importance resampling (SIR) algorithm. The optional TH= option specifies the threshold (a value between 0 and 1) of the resampling step in the SIR algorithm. The resampling step occurs when the effective sample size is less than the product of the threshold and the number of particles. By default, TH=0.8.

By default, ALGORITHM=SIR.

LEAD=*number*

specifies the number of multi-step forecast values to compute, where *number* must be a nonnegative integer. By default, LEAD=1, which means that a one-step-ahead forecast is performed.

NPARTICLE=*number*

specifies the number of particles to use in the forecasting process. If you omit this option, the forecasting process uses the value of the NPARTICLE= option in the PROC SMC statement.

PARM=MEAN | MEDIAN

specifies which parameter values from the posterior distribution to use for the forecasting distribution when you also specify the LEARN statement. If you omit the LEARN statement, this option is ignored. You can specify one of the following values:

MEAN constructs the forecasting distribution by using the estimated posterior means of the parameters.

MEDIAN constructs the forecasting distribution by using the estimated posterior medians of the parameters.

By default, PARM=MEAN.

OUT<(PERCENTILES=*numeric-list*)>=*libref.data-table*

OUT<(PERCENTILE=*numeric-list*)>=*libref.data-table*

OUT<(PERCENT=*numeric-list*)>=*libref.data-table*

OUT<(PERC=*numeric-list*)>=*libref.data-table*

OUT<(PCT=*numeric-list*)>=*libref.data-table*

writes the summary of the forecasting estimates to the specified output data table. The weighted average, weighted standard deviation, minimum value, first quantile, second quantile, third quantile, maximum value, and percentile points of each state variable at each time step are displayed. The optional PERCENTILES=*numeric-list* specifies the percentile points in the summary table of the forecasting estimates, where each *value* in the *numeric-list* must be between 0 and 100 and values are separated by commas or spaces. By default, PERCENTILES=2.5, 97.5, which yields the 2.5th and

97.5th percentile points, respectively. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1903.

OUTSIM=*libref.data-table*

writes the forecasting estimates to the specified output data table. The particle values with the corresponding normalized weights at each time step are displayed. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1903.

RESAMPLE

specifies that particles are to be resampled before the forecasting process.

ID Statement

ID < TIME= *>variable* ;

The ID statement identifies observations in the input data set by specifying a variable for the time series data. The values of the *variable* must not contain duplicated values.

INITSTATE Statement

INITSTATE *state-variable* $\sim$ *distribution* *< / option >* ;

The INITSTATE statement specifies the initialization distribution of the state variable given the parameters; that is, $\mathbf{X}_t \sim \mu_{\theta}(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$. For more information about the initialization distribution such as the definitions of τ , p , and s , see the section “[Sequential Monte Carlo Methods and State Space Models](#)” on page 1961. The INITSTATE statement is ignored if you specify the SUBMITMODEL statement. The INITSTATE statement has the following requirements if you omit the SUBMITMODEL statement:

- The INITSTATE statement is required. Each state variable must have a corresponding INITSTATE statement.
- Multiple INITSTATE statements are permitted. When you specify multiple INITSTATE statements, be careful about the order if there is any dependency between state variables.

The INITSTATE statement must have the following components in the order shown:

1. *variable-name*: A non-lagged or lagged format of a single state variable that you specify in the [STATEVAR](#) statement.
2. *Tilde*: A tilde ($\sim$) symbol between the state variable and the distribution.
3. *Distribution*: A distribution that is specified in the form *distribution-name(distribution-arguments)*. The available *distribution-name* and *distribution-arguments* in the INITSTATE statements are displayed in [Table 27.3](#). For the description of each distribution, see the section “[Standard Distributions](#)” on page 1991.

Table 27.3 Distribution Names and Arguments

Distribution Name	Distribution Arguments	Definitions
BETA	a, b	Beta distribution with arguments shape parameter a and shape parameter b
CAUCHY	l, s	Cauchy distribution with arguments location l and scale s
DEGENERATE	c	Degenerate distribution with argument c
GAMMA	a, b	Gamma distribution with arguments shape a and scale b
LOGN	m, s	Log-normal distribution with arguments mean m and standard deviation s
NORMAL N	m, s	Normal distribution with arguments mean m and standard deviation s
POISSON	λ	Poisson distribution with argument mean λ
T	m, s, ν	Student's t distribution with arguments mean m , standard deviation s , and degrees of freedom ν
UNIFORM	l, u	Uniform distribution with arguments min l and scale u

The *distribution-arguments* are specified by expressions that are enclosed in parentheses and separated by commas. Such expressions must follow any SAS programming statements. To avoid the possible computation errors, you should make some adjustments to the expressions. For more information about the adjustments, see the section “[Distribution Argument Restriction](#)” on page 1990. The number of expressions is equal to the number of arguments of the given distribution. For the INITSTATE statement, the variables that can appear in the expressions must be selected from the following list:

- the model parameters
- the lagged dependent variables
- the current and lagged independent variables
- the current and lagged state variables that you specify in the preceding INITSTATE statements
- the actual time index, `_time`

For more information about the actual time index, see the section “[Time Index](#)” on page 1995. For more information about the parameters, state variables, and dependent/independent variables, see the sections “[PARAMETERS Statement](#)” on page 1937, “[STATEVAR Statement](#)” on page 1945, and “[VAR Statement](#)” on page 1960, respectively.

You can specify the following *option* after a forward slash (/):

PROPOSAL=*proposal-dist*

IMPORTANCE=*proposal-dist*

specifies the proposal (importance) initialization distribution given the parameters, the dependent and independent variables at the first time step; that is, $\mathbf{X}_1 \sim q_{\theta}^1(\mathbf{X}_1|\mathbf{Y}_1, \mathbf{Z}_1)$. For more information about the proposal initialization equation in the state space model, see the section “[Sequential Monte Carlo Methods and State Space Models](#)” on page 1961. If you omit this *option*, then the corresponding state variable uses the original initialization distribution as its proposal initialization distribution. In the SIR or APF algorithm, each state variable is sampled from its proposal initialization distributions at the first time step. Subsequently, the joint density of both the original and proposal initialization distributions of all the state variables is evaluated within the importance weight calculation. In the BF algorithm, the proposal initialization distribution of each state variable is always same as its original initialization distribution. For more information about the proposal initialization distribution, see the sections “[Sequential Importance Resampling Algorithm](#)” on page 1964 and “[Auxiliary Particle Filter](#)” on page 1966. The format of the *proposal-dist* is *proposal-dist-name(proposal-dist-arguments)*. The available *proposal-dist-name* and *proposal-dist-arguments* in the INITSTATE statements are also displayed in [Table 27.3](#).

The expressions for *proposal-dist-arguments* are enclosed in parentheses and separated by commas. Such expressions must follow any SAS programming statements. To avoid the possible computation errors, you should make some adjustments to the expressions. For more information about the adjustments, see the section “[Distribution Argument Restriction](#)” on page 1990. The number of expressions is equal to the number of arguments of the given distribution type. The variables that are used in the expressions for *proposal-dist-arguments* must be selected from the following list:

- the model parameters
- the current and lagged dependent variables
- the current and lagged independent variables
- the current and lagged state variables that have been specified in the preceding INITSTATE statements
- the actual time index, *_time*

Moreover, the INITSTATE statements must satisfy the following conditions:

- The maximum lag order of all the dependent variables in the *distribution-arguments* and the PROPOSAL= option of all INITSTATE statements must be less than or equal to *p*, where *p* is defined in the section “[Method 1: Maximum Lag Order of Dependent Variables, p](#)” on page 1986.
- The maximum lag order of all the independent variables in the *distribution-arguments* and the PROPOSAL= option of all INITSTATE statements must be less than or equal to *s*, where *s* is defined in the section “[Method 1: Maximum Lag Order of Independent Variables, s](#)” on page 1986.

- The variables that appear as the *variable-name* in all INITSTATE statements must not have duplicates; that is, each state variable or lagged state variable must appear only once as the *variable-name* in all INITSTATE statements.
- Each state variable must appear once as the *variable-name* in one INITSTATE statement.
- The maximum lag order of all the state variables in the *variable-name* of all the INITSTATE statements must be less than or equal to r , where r is defined in the section “[Method 1: Maximum Lag Order of State Variables, \$r\$](#) ” on page 1986
- Suppose there are d_x state variables. For $i = 1, \dots, d_x$, if $\tau_i = \max(r_{1i}, r_{2i} - 1) > 0$, where r_{1i} and r_{2i} are defined in the section “[Method 1: Maximum Lag Order of State Variables, \$r\$](#) ” on page 1986, then *statev.L1*, ..., *statev.L τ_i* must appear once as the *variable-name* in one INITSTATE statement.

For example, suppose the initialization equations are

$$\begin{aligned} x_{1,t} &= \gamma \xi_t, & \xi_t &\stackrel{\text{iid}}{\sim} N(0, 1) \\ x_{2,t} &= \phi x_{1,t} + \eta_t & \eta_t &\stackrel{\text{iid}}{\sim} N(0, 1), \quad \xi_t \perp \eta_t \end{aligned}$$

where $t = 1$. Then the following statements define these initialization equations:

```
initstate x1 ~ normal(0, gamma);
initstate x2 ~ normal(phi*x1, 1);
```

where phi and gamma are parameters and x1 and x2 are current state variables.

If $x_{1,1}$ has the proposal initialization equation

$$x_{1,t} = y_t + \gamma \epsilon_t, \epsilon_t \stackrel{\text{iid}}{\sim} N(0, \gamma), t = 1$$

then the corresponding initialization distributions can be defined by

```
initstate x1 ~ normal(0, gamma) / proposal = normal(y, gamma);
initstate x2 ~ normal(phi*x1, 1);
```

where the state variable x1 has a proposal initialization distribution that incorporates the current dependent variable y, and the state variable x2 uses the original initialization as its proposal initialization distribution.

LEARN Statement

LEARN <options> ;

The LEARN statement estimates the parameters. You can specify the following *options*:

ALGORITHM=*algorithm*

ALG=*algorithm*

specifies the particle filter algorithm. You can specify one of the following *algorithms*:

- APF<(ADP)>** specifies the auxiliary particle filter (APF) algorithm. The ADP suboption specifies the fully adapted auxiliary particle filter algorithm.
- BF** specifies the bootstrap filter (BF) algorithm.
- SIR<(TH | THRESH | THRESHOLD=*number*)>** specifies the sequential importance resampling (SIR) algorithm. The optional TH= option specifies the threshold (a value between 0 and 1) of the resampling step in the SIR algorithm. The resampling step occurs when the effective sample size is less than the product of the threshold and the number of particles. By default, TH=0.8.

By default, ALGORITHM=SIR.

NPARTICLE=*number*

specifies the number of particles to use in the parameter estimation. If you omit this option, the parameter estimation uses the value of the NPARTICLE= option in the PROC SMC statement.

METHOD=PMCMC <(pmcmc-options)>

specifies the particle Markov chain Monte Carlo method and customizes its behavior with the *pmcmc-options*. By default, METHOD=PMCMC.

You can specify the following *pmcmc-options* in parentheses:

DIAGNOSTICS <(diag-options)>=ALL | NONE | keyword-list | (keyword-list)

DIAG <(diag-options)>=ALL | NONE | keyword-list | (keyword-list)

specifies which diagnostics to produce. If the value of the NCHAIN= option is greater than 1, then instead of the diagnostic tables being produced for each Markov chain, by default the results of the diagnostics for all the chains are appropriately combined and summarized. For more information, see the section “[Using Multiple Chains](#)” on page 37 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or both of the following *diag-options*:

BYCHAIN

produces diagnostic tables for each Markov chain. If NCHAIN=1, then the BYCHAIN option does nothing.

SETTINGS

produces a table of all MCMC diagnostic settings.

If you specify both the BYCHAIN and SETTINGS options, separate them by using a space or a comma. To produce all diagnostics, specify DIAGNOSTICS=ALL. To produce no diagnostics, specify DIAGNOSTICS=NONE. To produce some but not all diagnostics, or to change certain settings of these diagnostics, specify one or more of the following keywords. If you specify multiple keywords, separate them by using spaces.

AUTOCORR <(LAGS=*numeric-list*)>

AC <(LAGS=*numeric-list*)>

computes the autocorrelations at lags that you specify in the *numeric-list*. For more information, see the section “[Autocorrelations](#)” on page 34 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” Elements in the *numeric-list* must be positive integers, and values can be separated either by spaces or by commas. Repeated values are removed. If you omit the LAGS= option, autocorrelations of lags 1, 5, 10, and 50 are computed.

GEWEKE <(geweke-options)>

GEW <(geweke-options)>

computes the Geweke spectral density diagnostics. For more information, see the section “[Geweke Diagnostics](#)” on page 28 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *geweke-options*:

FRAC1=value

F1=value

specifies the fraction f_1 for the first window. By default, FRAC1=0.1.

FRAC2=value

F2=value

specifies the fraction f_2 for the second window. By default, FRAC2=0.5.

HEIDELBERGER <(heidel-options)>

HEIDEL <(heidel-options)>

HB <(heidel-options)>

computes the Heidelberg-Welch diagnostic for each variable. For more information, see the section “[Heidelberg-Welch Diagnostics](#)” on page 30 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or more of the following *heidel-options*:

EPSILON=value

EPS=value

specifies a positive number ϵ such that if the halfwidth is less than ϵ times the sample mean of the retained iterates, the halfwidth test is passed. By default, EPSILON=0.1.

HALFWIDTH_ALPHA=value

HALPHA=value

specifies the α level ($0 < \alpha < 1$) for the halfwidth test. By default, HALFWIDTH_ALPHA=0.05.

STATIONARY_ALPHA=value

SALPHA=value

specifies the α level ($0 < \alpha < 1$) for the stationarity test. By default, STATIONARY_ALPHA=0.05.

RAFTERY <(raftery-options)>

RL <(raftery-options)>

computes the Raftery-Lewis diagnostics. For more information, see the section “[Raftery-Lewis Diagnostics](#)” on page 31 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify one or more of the following *raftery-options*:

ACCURACY=value

ACC=value

R=value

specifies a small positive number r as the margin of error for measuring the accuracy of estimation of the quantile. By default, ACCURACY=0.005.

EPSILON=*value*

EPS=*value*

specifies the tolerance level (ϵ , a small positive number) for the stationary test. By default, EPSILON=0.001.

PROBABILITY=*value*

PROB=*value*

S=*value*

specifies the probability s of attaining the accuracy of the estimation of the quantile. By default, PROBABILITY=0.95.

QUANTILE=*value*

QUANT=*value*

Q=*value*

specifies the order (q , a value between 0 and 1) of the quantile of interest. By default, QUANTILE=0.025.

SUMMARIES < (**MAXLAG**=*value*) >

SUMMARY < (**MAXLAG**=*value*) >

SUM < (**MAXLAG**=*value*) >

computes a summary of MCMC diagnostics for each parameter in the Markov chain, including effective sample size (ESS), Monte Carlo standard error (MCSE), and related diagnostics. For more information about ESS, see the section “[Effective Sample Size](#)” on page 34 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#)”; for more information about MCSE, see the section “[Standard Error of the Mean Estimate](#)” on page 36 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” MAXLAG=*value* specifies the maximum autocorrelation lag to use for computing ESS, which is used to compute MCSE, where *value* must be a positive integer. By default, MAXLAG=250.

By default, DIAGNOSTICS=SUMMARIES.

NBURNIN=*number*

BURNIN=*number*

NBI=*number*

specifies the number of burn-in iterations to perform in each Markov chain before the chains are saved. For more information, see the section “[Burn-In, Thinning, and Markov Chain Samples](#)” on page 27 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, NBURNIN=0.

NCHAIN=*number*

NCHAINS=*number*

CHAINS=*number*

specifies the number of Markov chains to run. By default, NCHAIN=1.

NSAMPLE=number**NSAM=number****NMC=number**

specifies the number of iterations after tuning and burn-in to save in each Markov chain. For more information, see the section “[Burn-In, Thinning, and Markov Chain Samples](#)” on page 27 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, NSAMPLE=1000.

OUTPOST<(outpost-option)>=libref.data-table

names the SAS data table to contain the posterior sample. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1903. You can specify the following *outpost-option*:

DETAILED

saves any relevant additional variables, such as auxiliary variables, in the SAS data table. The random walk Metropolis (RWM) sampler without the **TRANSFORM** option has no additional variables to save. With the **TRANSFORM** option, the SAS data table also includes the transformed version of each parameter and the log posterior of the transformed parameters.

SAMPLER=RWM <(rwm-options)>**SAMPLING=RWM <(rwm-options)>**

samples from the posterior distribution by using the random walk Metropolis scheme. You can specify the following *rwm-options*:

ACCTOL=value**ATOL=value**

specifies the tolerance ϵ for the acceptance rate during the tuning process of the random walk Metropolis (RWM) sampler. For more information, see the section “[Scale Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, ACCTOL=0.075.

INITSCALE=value**SCALE=value**

specifies the initial scale parameter(s) c^{cur} of the RWM proposal distribution. For more information, see the section “[Scale Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, INITSCALE=2.38.

NTUNE=number**NTU=number**

specifies the number of iterations, n_{tune} , to perform for each tuning stage of the random walk Metropolis sampler. For more information, see the section “[Tuning Random Walk Metropolis Algorithms](#)” on page 19 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, NTUNE=500.

NTUNESTAGE=number

specifies the number of tuning stages, n_{stage} . For more information, see the section “[Tuning Random Walk Metropolis Algorithms](#)” on page 19 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, NTUNESTAGE=24.

TARGETRATE=value**RATE=value****DELTA=value**

specifies the target acceptance rate δ of the random walk Metropolis sampler. For more information, see the section “[Tuning Random Walk Metropolis Algorithms](#)” on page 19 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, TARGETRATE=0.234.

TRANSFORM**TRANS**

transforms all parameters to an unconstrained space and implements the random walk Metropolis sampler on the unconstrained space. For more information, see the section “[Parameter Transformation](#)” on page 25 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).”

TUNEWEIGHT=value**WEIGHT=value****WT=value**

specifies the tuning weight w to use during the tuning phase of the random walk Metropolis sampler to tune the proposal covariance matrix. For more information, see the section “[Covariance Tuning](#)” on page 20 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, TUNEWEIGHT=0.75.

SAMPLERSETTINGS**SAMSETTINGS****SAMSET**

produces a summary of the MCMC sampler settings to be used to sample from the posterior distribution, including tuning settings.

SAMPLERSUMMARY**SAMSUMMARY****SAMSUM**

produces a summary of the MCMC sampler to be used to sample from the posterior distribution, including, for example, Metropolis acceptance rates.

SEED=number

specifies an integer seed in the range 1 to $2^{31} - 1$ to use for the random number generator in the simulation. Specifying a seed enables you to reproduce identical Markov chains for the same specification. If you omit this option, or if you specify a non-positive seed, the seed specified in the PROC SMC statement is used.

STATISTICS < (*stat-option*) >=ALL | NONE | keyword | (*keyword-list*)

STATISTIC < (*stat-option*) >=ALL | NONE | keyword | (*keyword-list*)

STATS < (*stat-option*) >=ALL | NONE | keyword | (*keyword-list*)

STAT < (*stat-option*) >=ALL | NONE | keyword | (*keyword-list*)

specifies which posterior summary statistics to produce. If the value of the NCHAIN= option is greater than 1, then instead of the posterior summary statistics being computed separately for each Markov chain, by default the posterior summary statistics are computed by using all chains. For more information, see the section “[Using Multiple Chains](#)” on page 37 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *stat-option*:

BYCHAIN

produces posterior summary statistics tables for each Markov chain. If NCHAIN=1, then the BYCHAIN option does nothing.

To produce all statistics, specify STATISTICS=ALL. To produce no statistics, specify STATISTICS=NONE. To produce some but not all statistics, or to change certain settings of these statistics, specify one or more of the following keywords. If you specify multiple keywords, separate them by using spaces.

CORR

COR

produces the posterior correlation matrix.

COV

produces the posterior covariance matrix.

INTERVAL < (*interval-option*) >

INTERVALS < (*interval-option*) >

INT < (*interval-option*) >

produces equal-tail posterior credible intervals and highest posterior density (HPD) credible intervals. For more information, see the sections “[Interval Estimation](#)” on page 10 and “[Summary Statistics](#)” on page 35 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *interval-option*:

ALPHA=*numeric-list*

ALPHAS=*numeric-list*

computes posterior credible intervals at the levels that you specify in the *numeric-list*. Values in the *numeric-list* must be between 0 and 1, and they can be separated either by spaces or by commas. Repeated values are removed. Each value in the *numeric-list* produces a pair of 100(1-*value*)% equal-tail and HPD intervals for each parameter. By default, ALPHA=0.05, which yields the 95% credible intervals for each parameter.

SUMMARIES < (*summary-option*) >

SUMMARY < (*summary-option*) >

SUM < (*summary-option*) >

produces the means, standard deviations, and percentiles for the posterior sample. For more information, see the section “[Summary Statistics](#)” on page 35 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” You can specify the following *summary-option*:

PERCENTILES=*numeric-list*

PERCENTILE=*numeric-list*

PERCENT=*numeric-list*

PERC=*numeric-list*

PCT=*numeric-list*

computes the percentiles of the posterior sample that you specify in the *numeric-list*. Values in the *numeric-list* must be between 0 and 100, and they can be separated either by spaces or by commas. Repeated values are removed. By default, PERCENTILES=(2.5 25 50 75 97.5), which yields the 2.5th, 25th, 50th, 75th, and 97.5th percentiles, respectively, for each parameter.

The posterior mean and median are two point estimates that you can use in the [EVALUATE](#), [FILTER](#), [FORECAST](#), and [SMOOTH](#) statements (see the PARM= option for each of those statements). Therefore, the 50th percentile of the posterior sample is always computed whether it is specified in the *numeric-list* or not.

By default, STATISTICS=SUMMARIES.

THIN=*k*

THINNING=*k*

thins the Markov chain by using only one of every *k* iterations. For more information, see the section “[Burn-In, Thinning, and Markov Chain Samples](#)” on page 27 in Chapter 2, “[Introduction to Bayesian Analysis Procedures](#).” By default, THIN=1.

MODEL Statement

MODEL *dependent-variable* ~ *distribution* ;

The MODEL statement specifies the measurement distribution of the dependent variable given the parameters, the state variables, and the independent variables; that is, $\mathbf{Y}_t \sim g_{\theta}(\cdot | \mathbf{X}_{t-r:t}, \mathbf{Z}_{t-s:t}, \mathbf{Y}_{t-p:t-1})$, $s \geq 0$, $r \geq 0$, $p \geq 1$. For more information about the measurement distribution in the state space model, see the section “[Sequential Monte Carlo Methods and State Space Models](#)” on page 1961. The MODEL statement is ignored if you specify the SUBMITMODEL statement. The MODEL statement has the following requirements if you omit the SUBMITMODEL statement:

- The MODEL statement is required. Each dependent variable must have a corresponding AUXMODEL statement.
- Multiple MODEL statements are permitted. When you specify multiple MODEL statements, be careful about the order of the MODEL statements if there exists dependency between dependent variables.

The MODEL statement must have the following components in the order shown:

1. *Dependent variable*: A single dependent variable from the input data table.
2. *Tilde*: A tilde (~) symbol between the dependent variable and the distribution.

3. *Distribution*: A distribution that is specified in the form *distribution-name(distribution-arguments)*. The available *distribution-name* and *distribution-arguments* in the MODEL statements are displayed in Table 27.4. For the description of each distribution, see the section “[Standard Distributions](#)” on page 1991.

Table 27.4 Distribution Names and Arguments

Distribution Name	Distribution Arguments	Definitions
BETA	a, b	Beta distribution with arguments shape parameter a and shape parameter b
CAUCHY	l, s	Cauchy distribution with arguments location l and scale s
GAMMA	a, b	Gamma distribution with arguments shape a and scale b
LOGN	m, s	Log-normal distribution with arguments mean m and standard deviation s
NORMAL N	m, s	Normal distribution with arguments mean m and standard deviation s
POISSON	λ	Poisson distribution with argument mean λ
T	m, s, ν	Student's t distribution with arguments mean m , standard deviation s , and degrees of freedom ν
UNIFORM	l, u	Uniform distribution with arguments min l and scale u

The *distribution-arguments* are specified by expressions that are enclosed in parentheses and separated by commas. Such expressions must follow any SAS programming statements. To avoid the possible computation errors, you should make some adjustments to the expressions. For more information about the adjustments, see the section “[Distribution Argument Restriction](#)” on page 1990. The number of expressions is equal to the number of arguments of the given distribution. For the MODEL statement, the variables that can appear in the expressions must be selected from the following list:

- the model parameters
- the current and lagged state variables
- the current and lagged independent variables
- the lagged dependent variables
- the current dependent variables whose MODEL statements are specified in the preceding MODEL statements
- the actual time index, `_time`

For more information about the actual time index, see the section “[Time Index](#)” on page 1995. For more information about the parameters, the state variables, and the dependent/independent variables, see the sections “[PARAMETERS Statement](#)” on page 1937, “[STATEVAR Statement](#)” on page 1945, and “[VAR Statement](#)” on page 1960, respectively.

Moreover, the MODEL statements must satisfy the following conditions:

- The maximum lag order of all the state variables in the *distribution-arguments* of all MODEL statements must be less than or equal to r , where r is defined in the section “[Method 1: Maximum Lag Order of State Variables, \$r\$](#) ” on page 1986.
- The maximum lag order of all the dependent variables in the *distribution-arguments* of all MODEL statements must be less than or equal to p , where p is defined in the section “[Method 1: Maximum Lag Order of Dependent Variables, \$p\$](#) ” on page 1986.
- The maximum lag order of all the independent variables in the *distribution-arguments* of all MODEL statements must be less than or equal to s , where s is defined in the section “[Method 1: Maximum Lag Order of Independent Variables, \$s\$](#) ” on page 1986.
- The dependent variables in all MODEL statements must not have duplicates; that is, each dependent variable should appear only once as the *dependent variable* in all MODEL statements.

For example, the following statement defines the measurement distribution $y_t = \beta \exp(\frac{x_t}{2})\epsilon_t$, $\epsilon_t \stackrel{\text{iid}}{\sim} N(0, 1)$:

```
model y ~ normal(0, beta*exp(x/2));
```

where y is the current dependent variable, β is the parameter, and x is the current state variable.

The second example supposes that the measurement distributions of the state-space model are

$$y_{1,t} = \alpha y_{1,t-1} + \exp(\frac{x_t}{2})\xi_t, \quad \xi_t \stackrel{\text{iid}}{\sim} N(0, 1)$$

$$y_{2,t} = \phi y_{1,t} + \gamma \exp(\frac{x_{t-1}}{2})\eta_t, \quad \eta_t \stackrel{\text{iid}}{\sim} N(0, 1), \quad \xi_t \perp \eta_t$$

Then the following statements define these measurement distributions:

```
model y1 ~ normal(alpha*y1.l1, exp(x1/2));
model y2 ~ normal(phi*y1, gamma*exp(x1.l1/2));
```

where α , ϕ , and γ are parameters and $y1$ and $y2$ are current dependent variables. The first-order lagged state variable $x.l1$ is constructed by appending the suffix “.l1” to the current state variable x . The first-order lagged dependent variables $y1.l1$ is constructed by appending the suffix “.l1” to the current dependent variable $y1$ (see the section “[Method 1: Lagged Variables](#)” on page 1985).

PARAMETERS Statement

PARAMETERS *parameter-specification* < , . . . , *parameter-specification* > < / *options* > ;

PARMS *parameter-specification* < , . . . , *parameter-specification* > < / *options* > ;

The PARAMETERS (or PARMS) statement lists the names of the model parameters and specifies the initial value of each parameter. Whether the PARAMETERS statement is required depends on whether the SUBMITMODEL statement is specified: If you specify the SUBMITMODEL statement, the PARAMETERS statement is ignored; if you omit the SUBMITMODEL statement, the PARAMETERS statement is required.

A *parameter-specification* has the following general form, where *name* identifies the parameter and *value* provides a single initial value for the parameter:

name = value

Multiple *parameter-specifications* must be separated by commas or spaces. You can specify the following *options* after a forward slash (/):

LOWER=(*number1 number2 ...*)

LOWER=(*number*)

specifies the lower bounds of the specified parameters. The list can contain exactly one number, which is taken to be the lower bound of all the listed parameters in the statement, or it must contain as many values as the number of parameters specified. A missing value, denoted by a period (.), is a permissible value, which signifies that the parameter has no lower bound.

UPPER=(*number1 number2 ...*)

UPPER=(*number*)

specifies the upper bounds of the specified parameters. The list can contain exactly one number, which is taken to be the upper bound of all the listed parameters in the statement, or it must contain as many values as the number of parameters specified. A missing value, denoted by a period (.), is a permissible value, which signifies that the parameter has no upper bound.

The parameter names are case-insensitive, and the following conditions must be satisfied:

- The parameter names must not have duplicates.
- The parameter names must not contain the dot (“.”) symbol.
- “_t” or “_time” is a reserved name in PROC SMC. “_t” or “_time” (case-insensitive) must not be used as a parameter name.
- “dft_” is a reserved prefix in PROC SMC. The variable and parameter names must not start with “dft_” (case-insensitive).

For example, the following PARMS statement specifies three parameters and sets their initial values:

```
parms phi=0.9 beta=1 gamma=0.4;
```

PRIOR Statement

PRIOR *parameter-list* ~ *distribution* ;

The PRIOR statement specifies the prior distribution and range of the model parameters. You must specify a single parameter or a list of parameters, a tilde (~), and then a distribution with its hyperparameters. Multiple PRIOR statements are permitted. The PRIOR statement is ignored if you specify the SUBMITMODEL statement.

With respect to each PRIOR statement, the syntax must have the following components in the order shown:

1. *List of parameters*: A list of parameters that have the same prior distribution.
2. *Tilde*: A tilde (~) symbol between the list of parameters and the distribution.
3. *Distribution with hyperparameters*: A distribution that is specified in the form *distribution-name*(*distribution-arguments*). The available *distribution-name* and *distribution-arguments* in the prior statements are displayed in [Table 27.5](#).

Table 27.5 Distribution Names and Arguments

Distribution Name	Distribution Arguments	Definitions
BETA	a, b	Beta distribution with arguments shape parameter a and shape parameter b
CAUCHY	l, s	Cauchy distribution with arguments location l and scale s
GAMMA	a, b	Gamma distribution with arguments shape a and scale b
IGAMMA	a, b	Inverse gamma distribution with arguments shape a and scale b
LOGN	m, s	Log-normal distribution with arguments mean m and standard deviation s
NORMAL N	m, s	Normal distribution with arguments mean m and standard deviation s
T	m, s, ν	Student's t distribution with arguments mean m , standard deviation s , and degrees of freedom ν
UNIFORM	l, u	Uniform distribution with arguments min l and scale u

The *distribution-arguments* are specified by numbers that are enclosed in parentheses and separated by commas.

The constant hyperparameters of a prior distribution are specified by constant numbers in parentheses. For example, the following statements specify the prior distributions of ϕ , var_1 , and var_2 , where the latter two parameters have the same prior distribution:

```
prior phi ~ uniform(-1, 1);
prior var_1 var_2 ~ igamma(0.01, 0.01);
```

The range of each model parameter defines its upper and lower bounds when you draw them from the random walk proposal in the PMCMC method. If any model parameter is not specified in any PRIOR statement, then the contribution of that parameter to the logarithmic prior density function, $\ln p(\theta)$, is zero (see the section “[Particle Marginal Metropolis-Hastings Sampler](#)” on page 1974), and the range of that parameter is defined by LOWER and UPPER options in the PARAMETERS statement (see the section “[PARAMETERS Statement](#)” on page 1937). For any model parameter that you specify in the PRIOR statement, its range is defined by the intersection of the range specified in the PARAMETERS statement and the default range of the prior distribution. In addition, the range of that parameter defined in the PARAMETERS statement must be a subset of the default range of its prior distribution. For example, the following statements cause an error because $0 < 0.2$:

```
parameter phi = 0.5 / lower = 0 upper = 1;
prior phi ~ uniform(0.2, 1);
```

SCORE Statement

SCORE *options* ;

The SCORE statement specifies options that are related to scoring data. You can specify the following *options*:

INMODEL=*libref.data-table*

specifies the input data table that contains model information. The data table must be an output data table that is produced by the OUTMODEL= option in a previous call of the SMC procedure. When you specify this option, the ID, MODEL, and LEARN statements are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1903.

OUTMODEL=*libref.data-table*

writes the model information and the learning results to the specified output data table. The output data table consists of binary large object columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 1903.

SMOOTH Statement

SMOOTH <*options*> ;

The SMOOTH statement specifies options that are related to the smoothing problem, which is to find the probability distribution of the state, conditional on all observations (see the section “[SMC Smoothing](#)” on page 1969). If the PMCMC method is specified in the LEARN statement, the model parameters in the requested smoothing method are set to the posterior mean of the corresponding model parameters. If the LEARN statement is omitted, the parameters with initial values specified in the PARAMETERS statement are used for the smoothing problem. You can specify the following *options*:

ALGORITHM=*algorithm*

ALG=*algorithm*

specifies the particle filter algorithm to use in the smoothing process. You can specify the following *algorithms*:

APF<(ADP)> specifies the auxiliary particle filter (APF) algorithm. The ADP suboption specifies the fully adapted auxiliary particle filter algorithm.

BF specifies the bootstrap filter (BF) algorithm.

SIR<(TH | THRESH | THRESHOLD=*number*)> specifies the sequential importance resampling (SIR) algorithm. The optional TH= option specifies the threshold (a value between 0 and 1) of the resampling step in the SIR algorithm. The resampling step

occurs when the effective sample size is less than the product of the threshold and the number of particles. By default, TH=0.8.

By default, ALGORITHM=SIR.

METHOD=FIXEDINTERVAL | FIXEDLAG<(number)>

specifies the type of smoothing method. You can specify the following *values*:

FIXEDINTERVAL specifies the fixed-interval smoothing method.

FIXEDLAG<(number)> specifies the fixed-lag smoothing method. The optional *number* in parentheses specifies the number of lags to use in the fixed-lag smoothing method. By default, *number*=25.

By default, METHOD=FIXEDLAG<(number)>.

NPARTICLE=number

specifies the number of particles to use for the smoothing problem. If this option is omitted, the smoothing method uses the value of the NPARTICLE= option in the PROC SMC statement.

OUT<(PERCENTILES=numeric-list)>=libref.data-table

OUT<(PERCENTILE=numeric-list)>=libref.data-table

OUT<(PERCENT=numeric-list)>=libref.data-table

OUT<(PERC=numeric-list)>=libref.data-table

OUT<(PCT=numeric-list)>=libref.data-table

writes the summary of the smoothing estimates to the specified output data table. The weighted average, weighted standard deviation, minimum value, first quantile, second quantile, third quantile, maximum value, and percentile points of each state variable at each time step are displayed. The optional PERCENTILES=*numeric-list* specifies the percentile points in the summary table of the smoothing estimates, where each *value* in the *numeric-list* must be between 0 and 100 and values are separated by commas or spaces. By default, PERCENTILES=2.5, 97.5, which yields the 2.5th and 97.5th percentile points, respectively. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1903.

OUTSIM=libref.data-table

writes the smoothing estimates to the specified output data table. The particle values with the corresponding normalized weights at each time step are displayed. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 1903.

PARM=MEAN | MEDIAN

specifies which parameter values from the posterior distribution to use for the smoothing distribution when you also specify the LEARN statement. If the LEARN statement is omitted, this option is ignored. You can specify one of the following values:

MEAN	constructs the smoothing distribution by using the estimated posterior means of the parameters.
MEDIAN	constructs the smoothing distribution by using the estimated posterior medians of the parameters.

By default, PARM=MEAN.

STATE Statement

STATE *state-variable* $\sim$ *distribution* *</ options>* ;

The STATE statement specifies the transition distributions of the state variable given the parameters, the lagged state variables, the lagged and current independent variables; that is, $\mathbf{X}_t \sim f_{\theta}(\cdot | \mathbf{X}_{t-r:t-1}, \mathbf{Z}_{t-s:t}, \mathbf{Y}_{t-p:t-1})$, $t > 1, s \geq 0, r \geq 1, p \geq 1$. For more information about the transition equation in the state space model, see the section “[Sequential Monte Carlo Methods and State Space Models](#)” on page 1961. The STATE statement is ignored if you specify the SUBMITMODEL statement. The STATE statement has the following requirements if you omit the SUBMITMODEL statement:

- The STATE statement is required. Each state variable must have a corresponding STATE statement.
- Multiple STATE statements are permitted. When you specify multiple STATE statements, be careful about the order if there is any dependency between state variables.

The STATE statement must have the following components in the order shown:

1. *State variable*: A single state variable that is specified in the [STATEVAR](#) statement.
2. *Tilde*: A tilde ($\sim$) symbol between the state variable and the distribution.
3. *Distribution*: A distribution that is specified in the form *distribution-name*(*distribution-arguments*). The available *distribution-name* and *distribution-arguments* in the STATE statements are displayed in [Table 27.6](#). For the description of each distribution, see the section “[Standard Distributions](#)” on page 1991.

Table 27.6 Distribution Names and Arguments

Distribution Name	Distribution Arguments	Definitions
BETA	a, b	Beta distribution with arguments shape parameter a and shape parameter b
CAUCHY	l, s	Cauchy distribution with arguments location l and scale s
DEGENERATE	c	Degenerate distribution with argument c
GAMMA	a, b	Gamma distribution with arguments shape a and scale b
LOGN	m, s	Log-normal distribution with arguments mean m and standard deviation s

Table 27.6 continued

Distribution Name	Distribution Arguments	Definitions
NORMAL N	m, s	Normal distribution with arguments mean m and standard deviation s
POISSON	λ	Poisson distribution with argument mean λ
T	m, s, ν	Student's t distribution with arguments mean m , standard deviation s , and degrees of freedom ν
UNIFORM	l, u	Uniform distribution with arguments min l and scale u

The *distribution-arguments* are specified by expressions that are enclosed in parentheses and separated by commas. Such expressions must follow any SAS programming statements. To avoid the possible computation errors, you should make some adjustments to the expressions. For more information about the adjustments, see the section “[Distribution Argument Restriction](#)” on page 1990. The number of expressions is equal to the number of arguments of the given distribution. For the STATE statement, the variables that can appear in the expressions must be selected from the following list:

- the model parameters
- the lagged state variables
- the lagged dependent variables
- the lagged and current independent variables
- the state variables that have been specified in the preceding STATE statements
- the actual time index, `_time`

For more information about the actual time index, see the section “[Time Index](#)” on page 1995. For more information about the parameters, the state variables, and the dependent/independent variables, see the sections “[PARAMETERS Statement](#)” on page 1937, “[STATEVAR Statement](#)” on page 1945, and “[VAR Statement](#)” on page 1960, respectively.

You can specify the following *option* after a forward slash (/):

PROPOSAL=*proposal-dist*

IMPORTANCE=*proposal-dist*

specifies the proposal (importance) transition distribution given the parameters, the lagged state variables, and the current dependent and independent variables for $t > 1$; that is, $\mathbf{X}_t \sim q_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t})$, $t > 1$, $s \geq 1$, $r \geq 1$, $p \geq 0$. If this option is not specified, then the corresponding state variable uses the original transition distribution as its proposal transition distribution. In the SIR or APF algorithm, each state variable is sampled from its proposal transition distributions. Subsequently, the joint densities of both the original and proposal transition distributions of all the state variables are evaluated within the importance weight calculation. In the BF algorithm, the proposal transition distribution of each state variable is always the same as its original transition distribution. For more information about the proposal distribution, see the sections “[Sequential Importance Resampling Algorithm](#)” on page 1964 and “[Auxiliary Particle Filter](#)” on page 1966. The format

of the *proposal-dist* is *proposal-dist-name(proposal-arguments)*. The available *proposal-dist-name* and *proposal-arguments* in the STATE statements are also displayed in [Table 27.6](#).

The expressions for *proposal-arguments* are enclosed in parentheses and separated by commas. Such expressions must follow any SAS programming statements. To avoid the possible computation errors, you should make some adjustments to the expressions. For more information about the adjustments, see the section “[Distribution Argument Restriction](#)” on page 1990. The number of expressions is equal to the number of arguments of the given distribution type. The variables used in the expressions for *proposal-arguments* must be selected from the following list:

- the model parameters
- the lagged state variables
- the lagged and current dependent variables
- the lagged and current independent variables
- the state variables that have been specified in the preceding STATE statements
- the actual time index, *_time*

Moreover, the STATE statements must satisfy the following conditions:

- The maximum lag order of all the state variables in the *distribution-arguments* of all STATE statements must be less than or equal to *r*, where *r* is defined in the section “[Method 1: Maximum Lag Order of State Variables, *r*](#)” on page 1986.
- The maximum lag order of all the dependent variables in the *distribution-arguments* of all the STATE statements, including the PROPOSAL= option, must be less than or equal to *p*, where *p* is defined in the section “[Method 1: Maximum Lag Order of Dependent Variables, *p*](#)” on page 1986.
- The maximum lag order of all the independent variables in the *distribution-arguments* of all the STATE statements, including the PROPOSAL= option, must be less than or equal to *s*, where *s* is defined in the section “[Method 1: Maximum Lag Order of Independent Variables, *s*](#)” on page 1986.
- The state variables appear as the *variable-name* in all STATE statements must not have duplicates; that is, each state variable should appear only once as the *variable-name* in all STATE statements.

For example, suppose the transition equations are

$$x_{1,t} = x_{1,t-1} + \gamma \xi_t, \quad \xi_t \stackrel{\text{iid}}{\sim} N(0, 1),$$

$$x_{2,t} = \phi x_{2,t-1} + \beta \eta_t, \quad \eta_t \stackrel{\text{iid}}{\sim} N(0, 1), \quad \xi_t \perp \eta_t$$

Then the following statements define these initialization equations:

```
state x1 ~ normal(x1.l1, gamma);
state x2 ~ normal(phi*x2.l1, beta);
```

where phi, gamma, and beta are parameters and x1 and x2 are current state variables. The first-order lagged state variables x1.l1 and x2.l1 are constructed by appending the suffix “.l1” to the current state variables x1 and x2 (see the section “[Method 1: Lagged Variables](#)” on page 1985).

If $x_{1,t}$ has a proposal transition equation $x_{1,1} = y + \gamma \epsilon_t, \epsilon_t \stackrel{\text{iid}}{\sim} N(0, \gamma)$, the transition equations can be defined by

```
state x1 ~ normal(x1.1, gamma) / proposal = normal(y, gamma);
state x2 ~ normal(phi*x2.1, beta);
```

where the state variable x1 has a proposal transition distribution that incorporates the current dependent variable y and the state variable x2 uses the original transition as its proposal transition distribution.

STATEVAR Statement

STATEVAR *variable-list* ;

The STATEVAR statement specifies the names of the state variables in the SMC procedure. Whether the STATEVAR statement is required depends on whether the SUBMITMODEL statement is specified: If you specify the SUBMITMODEL statement, the STATEVAR statement is ignored; if you omit the SUBMITMODEL statement, the STATEVAR statement is required. You can specify one or more state variables in a space-separated *variable-list*. Each state variable must have corresponding INITSTATE and STATE statements.

The state variable names are case-insensitive, and the following conditions must be satisfied:

- The state variable names must not have duplicates.
- The state variable names must not contain the dot (“.”) symbol.
- “_t” or “_time” is a reserved name in PROC SMC. “_t” or “_time” (case-insensitive) must not be used as any state variable name.
- “dft_” is a reserved prefix in PROC SMC. The state variable names must not start with “dft_” (case-insensitive).

For example, the following statement specifies the state variables x1 and x2:

```
statevar x1 x2;
```

SUBMITMODEL Statement

SUBMITMODEL *dependent-variables* <=*independent-variables*> < / *options* > ;

The SUBMITMODEL statement indicates the start of your user-defined program statements between the SUBMITMODEL and ENDSUBMIT statements or specifies a file that contains your user-defined program statements in the FILE= option. When you specify the SUBMITMODEL statement, the AUXMODEL, INITSTATE, MODEL, PARAMETERS, PRIOR, STATE, STATEVAR, and VAR statements are ignored.

PROC SMC loads the predefined functions or subroutines from the current CMPLIB session and combines them with the functions or subroutines defined in the SUBMITMODEL statement. For more information about the combination of the CMPLIB session and SUBMITMODEL statement, see the section “[Method 2](#)” on page 1987.

The SUBMITMODEL defines the user-defined functions or subroutines by using either the FILE= option or the user-defined program statements that appear between the SUBMITMODEL and ENDSUBMIT

statements. If you specify the FILE= option, you cannot add lines of code for the user-defined program in the SUBMITMODEL statement. If you omit the FILE= option in the SUBMITMODEL statement, then PROC SMC executes all lines of code that are specified between the SUBMITMODEL and ENDSUBMIT statements. For more information, see the section “[ENDSUBMIT Statement](#)” on page 1921.

To define a state space model, you define FCMP functions and subroutines in the user-defined program. For more information, see the “FCMP Procedure” chapter in the *Base SAS Procedures Guide*. Each function or subroutine must be named using the format `<model-name>_<keyword>`, where *model-name* must be identical to the name specified in the NAME= option and *keyword* identifies the function or subroutine. The total length of the name should not exceed 32 characters. Each function or subroutine must have a specific signature, which consists of the number of arguments, sequence and types of arguments, and return value type. The summary of all the recognized function and subroutine names and their expected behavior is presented in [Table 27.7](#). Moreover, the type, definition, and size of each argument in the functions and subroutines are summarized in [Table 27.8](#). Consider the following points when you define a distribution model:

- When you define a function or subroutine, the name, type, and order of those arguments must follow the corresponding signature syntax and semantics. For more information about the signature syntax and semantics, see the section “[Signature Syntax and Semantics of User-Defined Functions and Subroutines](#)” on page 1950.
- When the SMC procedure invokes any function or subroutine, it provides the necessary input values according to the specified signature, and it expects the function or subroutine to prepare the output and return it according to the specification of the return values in the signature.
- You can use most of the SAS programming statements and SAS functions that you can use in a DATA step to define the FCMP functions and subroutines. However, there are a few differences in the capabilities of the DATA step and the FCMP procedure. To learn more, see the documentation of the FCMP procedure in the *Base SAS Procedures Guide*.
- The dimensions of the state variable and parameter array must match the values specified in the NSTATE= and NPARAM= options in the SUBMITMODEL statement. The dimensions of the dependent and independent variable array must match the number of variables specified in the *dependent-variables* and *independent-variables* in the SUBMITMODEL statement. The dimensions of the lagged variables must match the XLAG=, YLAG=, and ZLAG= option values. For more information, see [Table 27.8](#).
- Not all the functions and subroutines are required. Whether you need to define a particular function or subroutine depends on the SMC algorithms and other statements (see the section “[Signature Syntax and Semantics of User-Defined Functions and Subroutines](#)” on page 1950).
- The results of the subroutine should be returned in the parameter arguments of the subroutine.

You can specify the following options in the SUBMITMODEL statement after a forward slash (/):

FILE=SAS-file-ref | 'File-path'

specifies a file that contains your user-defined program. If you specify this option, you cannot specify the ENDSUBMIT statement.

You can specify the file either by including a *SAS-file-ref* that is defined in a prior SAS FILENAME statement or by including a quoted string (a 'File-path') that identifies the host-specific file path.

The following rules apply when you specify the FILE= option:

- The text file that you specify must contain only the program statements that you want to submit and none of the PROC SMC statements. You cannot include an ENDSUBMIT statement in the file.
- The text file that you specify must not contain any SAS macro statements or macro variable references. The file content is submitted as specified.

NAME=*symbol-name*

specifies the name of the model of interest. The model should be defined using the user-defined program (see the section “[Method 2](#)” on page 1987).

NPARM=*number*

specifies the number of parameters. The *number* should be a nonnegative integer.

NSTATE<(*numeric-list*)>=*number*

specifies the number of state variables. The *number* should be a nonnegative integer. The optional *numeric-list* in parentheses specifies the number of levels of all category variables. Each value in the *numeric-list* must be a positive integer, and values can be separated by commas or spaces. The number of values in the *numeric-list* must be less than or equal to the number of state variables. For each categorical state variable, its level values start from 0. For example, if a categorical state variable has three levels, its value is 0, 1, or 2.

In the underlying SMC algorithms of the SMC procedure, the category state variables are positioned before the noncategorical state variables in the state variable vector. For example, suppose that there are d_x state variables and d_x is defined by the NSTATE= option. If there are d_c ($0 \leq d_c \leq d_x$) values in the *numeric-list*, then the first d_c state variables are category variables and the remaining $d_x - d_c$ state variables are noncategorical variables.

XLAG=(*xlag-options*)

specifies the lag orders of the state variables in the distribution functions. For the XLAG= option in the SUBMITMODEL statement, the value of INIT= suboption must be greater than or equal to $\max(r_1, r_2 - 1, r_3 - 1)$, where r_1, r_2, r_3 are specified by the MEAS=, STATE=, and AUX= suboptions, respectively.

You can specify one or more of the following *xlag-options*:

AUX=*number***AUXILIARY=***number*

specifies the maximum lag order of the state variables in the auxiliary distribution function (*name_EVALNEXTMEAS*). If the APF algorithm is not specified in one or more of the EVALUATE, FILTER, FORECAST, LEARN, or LEARN statements, the AUX= suboption is ignored. By default, AUX=1.

INIT=*number***INITIALIZATION=***number*

specifies the maximum lag order of the state variables in the initialization distribution functions (*name_SAMPINIT* or *name_EVALINIT*) and proposal initialization distribution functions (*name_SAMPINITP* or *name_EVALINITP*). By default, INIT=0.

MEAS=number

MEASUREMENT=number

MODEL=number

specifies the maximum lag order of the state variables in the measurement distribution functions (*name_EVALMEAS* or *name_SAMPMEAS*). By default, MEAS=0.

STATE=number

TRAN=number

TRANSITION=number

specifies the maximum lag order of the state variables in the transition distribution functions (*name_SAMPTRAN* or *name_EVALTRAN*) and proposal transition distribution functions (*name_SAMPTRANP* or *name_EVALTRANP*). By default, STATE=1.

YLAG=(ylag-options)

specifies the lag orders of the dependent in the distribution functions. You can specify one or more of the following *ylag-options*:

AUX=number

AUXILIARY=number

specifies the maximum lag order of the dependent variables in the auxiliary distribution function (*name_EVALNEXTMEAS*). If the APF algorithm is not specified in one or more of the EVALUATE, FILTER, FORECAST, LEARN, or LEARN statements, the AUX= suboption is ignored. By default, AUX=0.

INIT=number

INITIALIZATION=number

specifies the maximum lag order of the dependent variables in the initialization distribution functions (*name_SAMPINIT* or *name_EVALINIT*) and proposal initialization distribution functions (*name_SAMPINITP* or *name_EVALINITP*). By default, INIT=0.

MEAS=number

MEASUREMENT=number

MODEL=number

specifies the maximum lag order of the dependent variables in the measurement distribution functions (*name_EVALMEAS* or *name_SAMPMEAS*). By default, MEAS=0.

STATE=number

TRAN=number

TRANSITION=number

specifies the maximum lag order of the dependent variables in the transition distribution functions (*name_SAMPTRAN* or *name_EVALTRAN*) and proposal transition distribution functions (*name_SAMPTRANP* or *name_EVALTRANP*). By default, STATE=0.

ZLAG=(*zlag-options*)

specifies the lag orders of the independent in the distribution functions. You can specify one or more of the following *zlag-options*:

AUX=number**AUXILIARY=number**

specifies the maximum lag order of the independent variables in the auxiliary distribution function (*name_EVALNEXTMEAS*). If the APF algorithm is not specified in one or more of the EVALUATE, FILTER, FORECAST, LEARN, or LEARN statements, the AUX= suboption is ignored. By default, AUX=0.

INIT=number**INITIALIZATION=number**

specifies the maximum lag order of the independent variables in the initialization distribution functions (*name_SAMPINIT* or *name_EVALINIT*) and proposal initialization distribution functions (*name_SAMPINITP* or *name_EVALINITP*). By default, INIT=0.

MEAS=number**MEASUREMENT=number****MODEL=number**

specifies the maximum lag order of the independent variables in the measurement distribution functions (*name_EVALMEAS* or *name_SAMPMEAS*). By default, MEAS=0.

STATE=number**TRAN=number****TRANSITION=number**

specifies the maximum lag order of the independent variables in the transition distribution functions (*name_SAMPTRAN* or *name_EVALTRAN*) and proposal transition distribution functions (*name_SAMPTRANP* or *name_EVALTRANP*). By default, STATE=0.

Table 27.7 shows functions and subroutines that define a state space model, and the subsections after the table provide more details.

Table 27.7 Functions and Subroutines That Define a State Space Model

Name	Type	Expected to Return
<i>name</i> _EVALINIT	Function	LOGPDF of the initialization distribution
<i>name</i> _EVALINITP	Function	LOGPDF of the proposal initialization distribution
<i>name</i> _EVALMEAS	Function	LOGPDF of the measurement distribution
<i>name</i> _EVALNEXTMEAS	Function	LOGPDF of the auxiliary distribution
<i>name</i> _EVALPRIO	Function	LOGPDF of the prior distribution
<i>name</i> _EVALTRAN	Function	LOGPDF of the transition distribution
<i>name</i> _EVALTRANP	Function	LOGPDF of the proposal transition distribution
<i>name</i> _LOWERBOUNDS	Subroutine	Lower bounds of the parameters
<i>name</i> _SAMPINIT	Subroutine	A sample from the initialization distribution
<i>name</i> _SAMPINITP	Subroutine	A sample from the proposal initialization distribution
<i>name</i> _SAMPMEAS	Subroutine	A sample from the measurement distribution
<i>name</i> _SAMPPRIO	Subroutine	Initial values for the parameters
<i>name</i> _SAMPTRAN	Subroutine	A sample from the transition distribution
<i>name</i> _SAMPTRANP	Subroutine	A sample from the proposal transition distribution
<i>name</i> _UPPERBOUNDS	Subroutine	Upper bounds of the parameters

Signature Syntax and Semantics of User-Defined Functions and Subroutines

Each function or subroutine has a fixed sequence and type of arguments. The name of each argument is also fixed. All the arguments of the functions and subroutines are listed in Table 27.8, where n is specified by the NPARM= option, d_x is defined by the NSTATE= option, d_y is the number of dependent variables, d_z is defined by the number of independent variables, r is specified by the maximum value in the XLAG=(*xlag-options*) option, p is specified by the maximum value in the YLAG=(*ylag-options*) option, and s is specified by the maximum value in the ZLAG=(*zlag-options*) option. Moreover, *_time* refers to the actual time index of the observations (see the section “Time Index” on page 1995).

For an array argument array-argument[*] that is one of theta[*], x[*], y[*], and z[*], the i th ($i \geq 1$) element, array-argument[i], refers to the i th element of the corresponding variable or parameter vector. For a lagged variable array argument lag-array-argument[*] that is one of xlag[*], ylag[*], and zlag[*], lag-array-argument[($j - 1$) * $d + i$] ($i, j \geq 1$) refers to the j th order of lag for the i th element of the corresponding variable whose dimension is denoted by d . For more information about the array or lagged array arguments for the parameter, state variables, dependent variables, and independent variables, see the sections “Method 2: Parameters” on page 1988, “Method 2: State Variables” on page 1988, “Method 2: Dependent Variables” on page 1988, “Method 2: Independent Variables” on page 1989, and “Method 2: Lagged Variables” on page 1988.

Table 27.8 List of Arguments in Functions and Subroutines

Name	Type	Definition	Size
theta[*]	Input/Output array	Parameters	n
x[*]	Input/Output array	Current state variables	d_x
xlag[*]	Input/Output array	Lagged state variables	$d_x * r$
y[*]	Input/Output array	Current dependent variables	d_y
ylag[*]	Input array	Lagged dependent variables	$d_y * p$
z[*]	Input array	Current independent variables	d_z
zlag[*]	Input array	Lagged independent variables	$d_z * s$
_time	Input scalar	Actual time index	1

The details of the argument, signature syntax, and semantics of each function or subroutine are listed as follows:

*name_***EVALINIT**

defines a function that returns the log of the probability density function (LOGPDF) of the initialization distribution, $\mu_{\theta}(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$.

- *Type*: FUNCTION
- *Requirement*: Required only when the APF or SIR algorithm is specified and the *name_SAMPINITP* function is specified
- *Number of arguments*: 7
- *Sequence and type of arguments*:

x[*]	Input numeric array of the current state variables at the initial time step
xlag[*]	Input numeric array of the lagged state variables before the initial time step
ylag[*]	Input numeric array of the lagged dependent variables before the initial time step
z[*]	Input numeric array of the current independent variables at the initial time step
zlag[*]	Input numeric array of the lagged independent variables before the initial time step
theta[*]	Input numeric array of the parameters
_time	Input numeric value of the current actual time index

Here is a sample structure of this function for a model named ‘TOY’:

```
function TOY_EVALINIT(x[*],xlag[*],ylag[*],z[*],zlag[*],theta[*],_time);
/*Code to compute LOGPDF based on initialization distribution*/
LOGPDF = <computed logpdf>;
return (LOGPDF);
endsub;
```

name_EVALINITP

defines a function that returns the LOGPDF of the proposal initialization distribution, $q_{\theta}^1(\mathbf{X}_{1-\tau:1}|\mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})$.

- *Type*: FUNCTION
- *Requirement*: Required only when the APF or SIR algorithm is specified and the *name_SAMPINITP* function is specified
- *Number of arguments*: 8
- *Sequence and type of arguments*:

<code>x[*]</code>	Input numeric array of the current state variables at the initial time step
<code>xlag[*]</code>	Input numeric array of the lagged state variables before the initial time step
<code>y[*]</code>	Input numeric array of the current dependent variables at the initial time step
<code>ylag[*]</code>	Input numeric array of the lagged dependent variables before the initial time step
<code>z[*]</code>	Input numeric array of the current independent variables at the initial time step
<code>zlag[*]</code>	Input numeric array of the lagged independent variables before the initial time step
<code>theta[*]</code>	Input numeric array of the parameters
<code>_time</code>	Input numeric value of the current actual time index

Here is a sample structure of this function for a model named ‘TOY’:

```
function TOY_EVALINITP(x[*],xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);
/*Code to compute LOGPDF based on proposal initialization distribution*/
LOGPDF = <computed logpdf>;
return (LOGPDF);
endsub;
```

name_EVALMEAS

defines a function that returns the LOGPDF of the measurement distribution, $g_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$.

- *Type*: FUNCTION
- *Requirement*: Always required
- *Number of arguments*: 8
- *Sequence and type of arguments*:

<code>x[*]</code>	Input numeric array of the current state variables
<code>xlag[*]</code>	Input numeric array of the lagged state variables
<code>y[*]</code>	Input numeric array of the current dependent variables
<code>ylag[*]</code>	Input numeric array of the lagged dependent variables
<code>z[*]</code>	Input numeric array of the current independent variables
<code>zlag[*]</code>	Input numeric array of the lagged independent variables

theta[*] Input numeric array of the parameters
 _time Input numeric value of the current actual time index

Here is a sample structure of this function for a model named ‘TOY’:

```
function TOY_EVALMEAS(x[*],xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);
  /*Code to compute LOGPDF based on measurement distribution*/
  LOGPDF = <computed logpdf>;
  return (LOGPDF);
endsub;
```

*name_***EVALNEXTMEAS**

defines a function that returns the LOGPDF of the auxiliary distribution, $\hat{p}_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$.

- *Type:* FUNCTION
- *Required:* Required only when the APF algorithm is specified in one or more of the EVALUATE, FILTER, FORECAST, LEARN, and SMOOTH statements.
- *Number of arguments:* 7
- *Sequence and type of arguments:*

xlag[*] Input numeric array of the lagged state variables
 y[*] Input numeric array of the current dependent variables
 ylag[*] Input numeric array of the lagged dependent variables
 z[*] Input numeric array of the current independent variables
 zlag[*] Input numeric array of the lagged independent variables
 theta[*] Input numeric array of the parameters
 _time Input numeric value of the current actual time index

Here is a sample structure of this function for a model named ‘TOY’:

```
function TOY_EVALNEXTMEAS(xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);
  /*Code to compute LOGPDF based on the approximated density function*/
  LOGPDF = <computed logpdf>;
  return (LOGPDF);
endsub;
```

*name_***EVALPRIO**

defines a function that defines the LOGPDF of the prior distribution, $p(\theta)$.

- *Type:* FUNCTION
- *Required:* Required only when the PMCMC method is specified in the LEARN statement
- *Number of arguments:* 1
- *Sequence and type of arguments:*

theta[*] Input numeric array of the parameters

Here is a sample structure of this function for a model named ‘TOY’:

```
function TOY_EVALPRIO(theta[*]);
/*Code to compute LOGPDF based on the prior distribution*/
LOGPDF = <computed logpdf>;
return (LOGPDF);
endsub;
```

*name_***EVALTRAN**

defines a function that returns LOGPDF of the transition distribution, $f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$.

- *Type*: FUNCTION
- *Requirement*: Required only when the APF or SIR algorithm is specified and the *name_SAMPTRANP* subroutine is specified
- *Number of arguments*: 7
- *Sequence and type of arguments*:

x[*] Input numeric array of the current state variables
xlag[*] Input numeric array of the lagged state variables
ylag[*] Input numeric array of the lagged dependent variables
z[*] Input numeric array of the current independent variables
zlag[*] Input numeric array of the lagged independent variables
theta[*] Input numeric array of the parameters
_time Input numeric value of the current actual time index

Here is a sample structure of this function for a model named ‘TOY’:

```
function TOY_EVALTRAN(x[*],xlag[*],ylag[*],z[*],zlag[*],theta[*],_time);
/*Code to compute LOGPDF based on the transition distribution*/
LOGPDF = <computed logpdf>;
return (LOGPDF);
endsub;
```

*name_***EVALTRANP**

defines a function that returns the LOGPDF of the proposal transition distribution, $q_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t})$.

- *Type*: FUNCTION
- *Requirement*: Required only when the APF or SIR algorithm is specified and the *name_SAMPTRANP* subroutine is specified
- *Number of arguments*: 8
- *Sequence and type of arguments*:

<code>x[*]</code>	Input numeric array of the current state variables
<code>xlag[*]</code>	Input numeric array of the lagged state variables
<code>y[*]</code>	Input numeric array of the current dependent variables
<code>ylag[*]</code>	Input numeric array of the lagged dependent variables
<code>z[*]</code>	Input numeric array of the current independent variables
<code>zlag[*]</code>	Input numeric array of the lagged independent variables
<code>theta[*]</code>	Input numeric array of the parameters
<code>_time</code>	Input numeric value of the current actual time index

Here is a sample structure of this function for a model named ‘TOY’:

```
function TOY_EVALTRANP(x[*],xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);
  /* Code to compute LOGCDF based on the proposal transition distribution*/
  LOGPDF = <computed logpdf>;
  return (LOGPDF);
endsub;
```

*name_***LOWERBOUNDS**

defines a subroutine that returns lower bounds for the parameters. To indicate the output numeric array, the line “outargs theta;” must appear after the line “subroutine *name_*LOWERBOUNDS(theta[*]);”.

- *Type*: SUBROUTINE
- *Requirement*: Not always required
- *Number of arguments*: 1
- *Sequence and type of arguments*:

<code>theta[*]</code>	Output numeric array of the parameters. <code>theta[i]</code> assigns the lower bound for the <i>i</i> th parameter.
-----------------------	--

If this subroutine is not defined, then the SMC procedure assumes a lower bound of $-\infty$ for each parameter. If a lower bound of l_i is returned for a parameter p_i , then PROC SMC assumes that $l_i < p_i$ (strict inequality). If a missing value or no value is returned for one parameter, then PROC SMC assumes that there is no lower bound for that parameter (this is equivalent to a lower bound of $-\infty$).

Here is a sample structure of this subroutine for a model named ‘TOY’ that has three parameters:

```
subroutine TOY_LOWERBOUNDS(theta[*]);
  outargs theta;
  theta[1] = <lower bound for the first parameter>;
  /*the second parameter is missing */
  theta[3] = <lower bound for the third parameter>;
endsub;
```

name_SAMPINIT

defines a subroutine that returns the initial state variables drawn from the initialization distribution, $\mu_{\theta}(\mathbf{X}_{1-\tau:1}|\mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$. To indicate the output numeric arrays, the line “outargs x, xlag;” must appear after the line “subroutine *name_SAMPINIT*(x[*],xlag[*],ylag[*],z[*],zlag[*],theta[*],_time);”.

- *Type*: SUBROUTINE
- *Requirement*: Required only when either of the following conditions is satisfied:
 - The BF algorithm is specified.
 - The APF or SIR algorithm is specified and the *name_SAMPINITP* subroutine is not specified.
- *Number of arguments*: 7
- *Sequence and type of arguments*:

x[*]	Output numeric array of the current state variables at the initial time step. x[i] assigns the initial value of the <i>i</i> th state variable.
xlag[*]	Output numeric array of the lagged state variables before the initial time step xlag[(<i>j</i> – 1) * <i>d_x</i> + <i>i</i>] (<i>i, j</i> ≥ 1) assigns the initial value of the <i>j</i> th order of lag for the <i>i</i> th state variable.
ylag[*]	Input numeric array of the lagged dependent variables before the initial time step
z[*]	Input numeric array of the current independent variables at the initial time step
zlag[*]	Input numeric array of the lagged independent variables before the initial time step
theta[*]	Input numeric array of the parameters
_time	Input numeric value of the current actual time index

Here is a sample structure of this subroutine for a model named ‘TOY’ that has three state variables:

```
subroutine TOY_SAMPINIT(x[*],xlag[*],ylag[*],z[*],zlag[*],theta[*],_time);
  outargs x, xlag;
  /*Code to generate x and xlag based on the sampling initial distribution*/
  x[1] = <generated first state variable>;
  x[2] = <generated second state variable>;
  x[3] = <generated third state variable>;
endsub;
```

name_SAMPINITP

defines a subroutine that returns the initial state variables drawn from the proposal initialization distribution, $q_{\theta}^1(\mathbf{X}_{1-\tau:1}|\mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})$. To indicate the output numeric arrays, the line “outargs x, xlag;” must appear after the line “subroutine *name_SAMPINITP*(x[*],xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);”.

- *Type*: SUBROUTINE
- *Required*: Required only when the APF or SIR algorithm is specified and the proposal initialization distribution, $q_{\theta}^1(\mathbf{X}_{1-\tau:1}|\mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})$, is different from the initialization distribution, $\mu_{\theta}(\mathbf{X}_{1-\tau:1}|\mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$
- *Number of arguments*: 8
- *Sequence and type of arguments*:

<code>x[*]</code>	Output numeric array of the current state variables at the initial time step. <code>x[i]</code> assigns the initial value of the i th state variable.
<code>xlag[*]</code>	Output numeric array of the lagged state variables before the initial time step. <code>xlag[($j - 1$) * d_x + i]</code> ($i, j \geq 1$) assigns the initial value of the j th order of lag for the i th state variable.
<code>y[*]</code>	Input numeric array of the lagged dependent variables at the initial time step
<code>ylag[*]</code>	Input numeric array of the lagged dependent variables before the initial time step
<code>z[*]</code>	Input numeric array of the current independent variables at the initial time step
<code>zlag[*]</code>	Input numeric array of the lagged independent variables before the initial time step
<code>theta[*]</code>	Input numeric array of the parameters
<code>_time</code>	Input numeric value of the current actual time index

Here is a sample structure of this subroutine for a model named ‘TOY’ that has three state variables:

```
subroutine TOY_SAMPINITP(x[*],xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);
  outargs x, xlag;
  /*Code to generate x and xlag based on the sampling initial distribution*/
  x[1] = <generated first state variable>;
  x[2] = <generated second state variable>;
  x[3] = <generated third state variable>;
endsub;
```

name_ **SAMPMEAS**

defines a subroutine that returns the current dependent variables drawn from the measurement distribution, $g_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$. To indicate the output numeric array, the line “outargs y;” must appear after the line “subroutine *name_*SAMPMEAS(`x[*]`,`xlag[*]`,`y[*]`,`ylag[*]`,`z[*]`,`zlag[*]`,`theta[*]`,`_time`);”.

- *Type*: SUBROUTINE
- *Requirement*: Required when the FORECAST statement is specified
- *Number of arguments*: 8
- *Sequence and type of arguments*:

<code>x[*]</code>	Input numeric array of the current state variables
<code>xlag[*]</code>	Input numeric array of the lagged state variables
<code>y[*]</code>	Output numeric array of the current dependent variables. <code>y[i]</code> assigns the value of the i th current dependent variable.
<code>ylag[*]</code>	Input numeric array of the lagged dependent variables
<code>z[*]</code>	Input numeric array of the current independent variables
<code>zlag[*]</code>	Input numeric array of the lagged independent variables
<code>theta[*]</code>	Input numeric array of the parameters
<code>_time</code>	Input numeric value of the current actual time index

Here is a sample structure of this subroutine for a model named ‘TOY’ that has one dependent variable:

```

subroutine TOY_SAMPMEAS(x[*],xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);
  outargs y;
  /*Code to generate x based on the sampling observation distribution*/
  y[1] = <the first generated dependent variable>;
endsub;

```

name_SAMPPRIO

defines a subroutine that returns the initial values of the parameters. To indicate the output numeric array, the line “outargs theta;” must appear after the line “subroutine *name_SAMPPRIO*(theta[*]);”.

- *Type*: Subroutine
- *Requirement*: Always required
- *Number of arguments*: 1
- *Sequence and type of arguments*:

theta[*] Output numeric array of the parameters. theta[i] assigns the initial value of the *i*th parameter.

Here is a sample structure of this subroutine for a model named ‘TOY’ that has three parameters:

```

subroutine TOY_SAMPPRIO(theta[*]);
  outargs theta;
  /*Code to initialize parameters*/
  theta[1] = <initial value of the first parameter>;
  theta[2] = <initial value of the second parameter>;
  theta[3] = <initial value of the third parameter>;
endsub;

```

name_SAMPTRAN

defines a subroutine that returns the current state variables drawn from the transition distribution, $f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$. To indicate the output numeric array, the line “outargs x;” must appear after the line “subroutine *name_SAMPTRAN*(x[*],xlag[*],ylag[*],z[*],zlag[*],theta[*],_time);”.

- *Type*: SUBROUTINE
- *Requirement*: Required only when one of the following conditions is satisfied:
 - The BF algorithm is specified.
 - The APF or SIR algorithm is specified and the *name_SAMPTRANP* subroutine is not specified.
 - The FORECAST statement is specified.
- *Number of arguments*: 7
- *Sequence and type of arguments*:

<code>x[*]</code>	Output numeric array of the current state variables. <code>x[i]</code> assigns the value of the <i>i</i> th current state variable.
<code>xlag[*]</code>	Input numeric array of the lagged state variables
<code>ylag[*]</code>	Input numeric array of the lagged dependent variables
<code>z[*]</code>	Input numeric array of the current independent variables
<code>zlag[*]</code>	Input numeric array of the lagged independent variables
<code>theta[*]</code>	Input numeric array of the parameters
<code>_time</code>	Input numeric value of the current actual time index

Here is a sample structure of this subroutine for a model named ‘TOY’ that has three state variables:

```

subroutine TOY_SAMPTRAN(x[*],xlag[*],ylag[*],z[*],zlag[*],theta[*],_time);
  outargs x;
  /*Code to generate x based on the sampling transition distribution*/
  x[1] = <the first generated state variable>;
  x[2] = <the second generated state variable>;
  x[3] = <the third generated state variable>;
endsub;

```

name_SAMPTRANP

defines a subroutine that returns the current state variables drawn from the proposal transition distribution, $q_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t})$. To indicate the output numeric array, the line “outargs x;” must appear after the line “subroutine *name_SAMPTRANP*(`x[*]`,`xlag[*]`,`y[*]`,`ylag[*]`,`z[*]`,`zlag[*]`,`theta[*]`,`_time`);”.

- *Type*: SUBROUTINE
- *Requirement*: Required only when the APF or SIR algorithm is specified and the proposal transition distribution, $q_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t})$, is different from the transition distribution, $f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$.
- *Number of arguments*: 9
- *Sequence and type of arguments*:

<code>x[*]</code>	Output numeric array of the current state variables. <code>x[i]</code> assigns the value of the <i>i</i> th current state variable.
<code>xlag[*]</code>	Input numeric array of the lagged state variables
<code>y[*]</code>	Input numeric array of the current dependent variables
<code>ylag[*]</code>	Input numeric array of the lagged dependent variables
<code>z[*]</code>	Input numeric array of the current independent variables
<code>zlag[*]</code>	Input numeric array of the lagged independent variables
<code>theta[*]</code>	Input numeric array of the parameters
<code>_time</code>	Input numeric value of the current actual time index

Here is a sample structure of this subroutine for a model named ‘TOY’ that has three state variables:

```

subroutine TOY_SAMPTRANP (x[*],xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);
  outargs x;
  /*Code to generate x based on the sampling transition distribution*/
  x[1] = <the first generated state variable>;
  x[2] = <the second generated state variable>;
  x[3] = <the third generated state variable>;
endsub;

```

*name_*UPPERBOUNDS

defines a subroutine that returns upper bounds for the parameters. To indicate the output numeric array, the line “outargs theta;” must appear after the line “subroutine *name_*UPPERBOUNDS(theta[*]);”.

- *Type*: Subroutine
- *Requirement*: Not always required
- *Number of arguments*: 1
- *Sequence and type of arguments*:

theta[*] Output numeric array of the parameters. theta[i] assigns the upper bound for the *i*th parameter.

Here is a sample structure of this subroutine for a model named ‘TOY’ that has three parameters:

```

subroutine TOY_UPPERBOUNDS (theta[*]);
  outargs theta;
  theta[1] = <upper bound for the first parameter>;
  theta[2] = <upper bound for the second parameter>;
  theta[3] = <upper bound for the third parameter>;
endsub;

```

If this subroutine is not defined, then the SMC procedure assumes that there is no upper bound for any of the parameters. If an upper bound of u_i is returned for a parameter p_i , then PROC SMC assumes that $p_i < u_i$ (strict inequality). If a missing value or no value is returned for one parameter, then PROC SMC assumes that there is no upper bound for that parameter (this is equivalent to an upper bound of ∞).

VAR Statement

VAR *variable-list* ;

The VAR statement specifies the names of the dependent variables and the independent (exogenous) variables in the SMC procedure. One or more variables can be specified in a space-separated *variable-list*, and the order does not matter. The variables must be selected from the input data table (see the **DATA=** option in the **PROC SMC** statement). The requirement of the VAR statement depends on whether the SUBMITMODEL statement is specified: If the SUBMITMODEL is specified, the VAR statement is ignored; if the SUBMITMODEL is not specified, the VAR statement is required.

The variables in the VAR statement must contain all dependent variables that are specified before the tilde (~) symbol in the MODEL statements, and the remaining variables in the VAR statement are viewed as independent variables in the SMC procedure.

The dependent and independent variable names are case-insensitive, and the following conditions must be satisfied:

- The dependent and independent variable names must not have duplicates.
- The dependent and independent variable names must not contain the dot (“.”) symbol.
- “_t” and “_time” is a reserved name in PROC SMC. “_t” or “_time” (case-insensitive) must not be used as any dependent and independent variable name.
- “dft_” is a reserved prefix in PROC SMC. The dependent and independent variable names must not start with “dft_” (case-insensitive).

For example, the following statements specify the variables y1, y2, and z1:

```
var y1 y2 z1;
model y1 ~ normal(0, exp(x1/2));
model y2 ~ normal(phi*x2, gamma);
```

where y1 and y2 are dependent variables and z1 is the independent variable.

Details: The SMC Procedure

Sequential Monte Carlo Methods and State Space Models

This section is a brief review of the sequential Monte Carlo (SMC) methods (also known as particle filtering methods). These methods provide simulation-based approximate solutions to the state space model (SSM), which is also known as the hidden Markov model (HMM), especially for the nonlinear or non-Gaussian SSM. In some literature, the HMM is referred to as the discrete-state SSM (that is, the state can take only discrete values), whereas the SSM is referred to as the continuous-state SSM (that is, the state can take only continuous values). In this chapter, the HMM is equivalent to the SSM; that is, the state can take discrete or continuous values or even a mix of discrete and continuous values. For more information, see Doucet and Johansen (2011); Doucet, Godsill, and Andrieu (2000); Cappé, Godsill, and Moulines (2007).

The SSM consists of a discrete time process $\{\mathbf{X}_t, \mathbf{Y}_t, \mathbf{Z}_t\}$, where $\{\mathbf{X}_t\}$ is an unobserved Markov process; conditional on $\{\mathbf{X}_t\}$, the observable process $\{\mathbf{Y}_t\}$ is a sequence of independent random variables. $\mathbf{X}_t$ are called the state variables, and $\mathbf{Y}_t$ are called the dependent variables. In many cases, the SSM also has independent variables that are denoted by $\mathbf{Z}_t$. The independent variables, also called the exogenous variables, are determined outside the SSM. Hereafter, θ denotes the parameters, any discrete-time sequence $\mathbf{S}_i, \mathbf{S}_{i+1}, \dots, \mathbf{S}_j (i \leq j)$ is denoted by $\mathbf{S}_{i:j}$, and T denotes the time step of the last observation. There is no restriction on the dimensionality of $\mathbf{X}_t$, $\mathbf{Y}_t$, and $\mathbf{Z}_t$; that is, they can be scalars, vectors, or matrices.

In general, the SSM can be expressed in three distribution functions:

- initialization distribution function
- transition distribution function

- measurement distribution function

In the distribution functions, a sequence that is denoted by $\mathbf{S}_{i:j}$ with $i > j$ might sometimes appear as an argument, and that sequence should be ignored in the function. For ease of representation, the lag orders of the three kinds of variables $\{\mathbf{X}_t, \mathbf{Y}_t, \mathbf{Z}_t\}$ in the distribution functions are constantly set to r , p , and s , respectively. However, in practice, the lag orders for each kind of variable can be different in different distribution functions. Therefore, in all the necessary distribution functions, r can be seen as the maximum lag order of all the state variables, p can be seen as the maximum lag order of all the dependent variables, s can be seen as the maximum lag order of the independent variables, and τ can be seen as the maximum lag order of the state variables that must be generated in the initialization (or proposal initialization) distribution. The maximum lag order of all the state variables for the measurement distribution is r_1 and the counterpart for the transition (or proposal transition) distribution is r_2 , then $\tau = \max(r_1, r_2 - 1)$. For each of the three kinds of variables of any distribution function, only the corresponding lagged variables are necessary, even though the maximum lag order of those variables is used in the function listed later. For example, suppose that $r_1 = 3$ and $r_2 = 5$. Then, r is set to 5. $\mathbf{X}_{t-3:t}$, $t \geq 1$, are used in the measurement distribution function, and $\mathbf{X}_{t-5:t-1}$, $t > 1$, are used in the transition distribution function. Moreover, $\tau = 4$; that is, only $\mathbf{X}_{-3:1}$ are necessarily drawn from the initialization distribution function because $\mathbf{X}_{-4}$ is not used in all the possible distribution functions in any SMC algorithm.

Besides those three distribution functions, some other distribution functions might be required in a specific SMC algorithm. All the distribution functions that might be necessary in the SMC algorithms are summarized in the following list:

- Initialization distribution function, $\mu_{\theta}(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$

This is the density function for the state variables at time step 1 and the lagged state variables with lag order up to τ that are denoted by $\mathbf{X}_{1-\tau:1}$, conditional on the lagged dependent variables with lag order up to p that are denoted by $\mathbf{Y}_{1-p:0}$, and the independent variables at time step 1 and the lagged independent variables with lag order up to s that are denoted by $\mathbf{Z}_{1-s:1}$.

- Transition distribution function, $f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}), t > 1$

This is the density function for the current state variables that are denoted by $\mathbf{X}_t$, conditional on the lagged state variables with lag order up to r that are denoted by $\mathbf{X}_{t-r:t-1}$, p lagged dependent variables that are denoted by $\mathbf{Y}_{t-p:t-1}$, and the current and lagged independent variables with lag order up to s that are denoted by $\mathbf{Z}_{t-s:t}$.

- Measurement distribution function, $g_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}), t \geq 1$

This is the density function for the current dependent variables that are denoted by $\mathbf{Y}_t$, conditional on the current and lagged state variables with lag order up to r that are denoted by $\mathbf{X}_{t-r:t}$, the lagged dependent variables with lag order up to p that are denoted by $\mathbf{Y}_{t-p:t-1}$, and the current and lagged independent variables with lag order up to s that are denoted by $\mathbf{Z}_{t-s:t}$.

- Proposal initialization distribution function, $q_{\theta}^1(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})$

This is the proposal density function for the initial state variables at time step 1 and the lagged state variables with lag order up to τ that are denoted by $\mathbf{X}_{1-\tau:1}$, conditional on the current and lagged dependent variables with lag order up to p that are denoted by $\mathbf{Y}_{1-p:1}$, and the current and lagged independent variables with lag order up to s that are denoted by $\mathbf{Z}_{1-s:1}$.

- Proposal transition distribution function, $q_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t}), t > 1$

This is the proposal density function for the current state variables that are denoted by $\mathbf{X}_t$, conditional on the lagged state variables with lag order up to r that are denoted by $\mathbf{X}_{t-r:t-1}$, the current and lagged dependent variables with lag order up to p that are denoted by $\mathbf{Y}_{t-p:t}$, and the current and lagged independent variables with lag order up to s that are denoted by $\mathbf{Z}_{t-s:t}$.

- Auxiliary distribution function, $\hat{p}_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}), t > 1$

This is the auxiliary density function for the current dependent variables that are denoted by $\mathbf{Y}_t$, conditional on the lagged state variables with lag order up to r that are denoted by $\mathbf{X}_{t-r:t-1}$, the lagged dependent variables with lag order up to p that are denoted by $\mathbf{Y}_{t-p:t-1}$, and the current and lagged independent variables with lag order up to s that are denoted by $\mathbf{Z}_{t-s:t}$.

- Prior distribution function, $p(\theta)$

This is the density function for the prior distribution of the parameters.

An SSM has five common problems:

- Filtering problem: What is $p_{\theta}(\mathbf{X}_t | \mathbf{Y}_{1:t}, \mathbf{Z}_{1:t})$, the probability distribution of state $\mathbf{X}_t$ given dependent variables $\mathbf{Y}_{1:t}$ and independent variables $\mathbf{Z}_{1:t}$?
- Smoothing problem: What is $p_{\theta}(\mathbf{X}_t | \mathbf{Y}_{1:T}, \mathbf{Z}_{1:T}), 1 \leq t \leq T$, the probability distribution of state $\mathbf{X}_t$ given dependent variables $\mathbf{Y}_{1:T}$ and independent variables $\mathbf{Z}_{1:T}$?
- Forecasting problem: What is $p_{\theta}(\mathbf{X}_{T+L} | \mathbf{Y}_{1:T}, \mathbf{Z}_{1:T}), L > 0$, the probability distribution of state $\mathbf{X}_{T+L}$ given dependent variables $\mathbf{Y}_{1:T}$ and independent variables $\mathbf{Z}_{1:T}$?
- Evaluating problem: What is $p_{\theta}(\mathbf{Y}_{1:t} | \mathbf{Z}_{1:t})$, the probability (or the likelihood) distribution of dependent variables $\mathbf{Y}_{1:t}$ given independent variables $\mathbf{Z}_{1:t}$?
- Learning problem: How do you estimate the unknown parameters θ given all observed variables $\mathbf{Y}_{1:T}$ and $\mathbf{Z}_{1:T}$?

The following sections demonstrate how to use the SMC methods to solve these problems. First, the SMC filtering, smoothing, forecasting, and evaluating algorithms are discussed using the simple SSM, in which $p = 0$ and no independent variables are incorporated. Moreover, the maximum lag order of all the state variables for the measurement distribution is 0 ($r_1 = 0$), and the counterpart for the transition (or proposal transition) distribution is 1 ($r_2 = 1$). This simple SSM is used in most of the literature and textbooks. The corresponding SMC algorithms for the general SSM that considers $p \geq 0, r_1 \geq 0, r_2 \geq 0$, the existence of independent variables, and $s \geq 0$ is discussed in the section “[SMC Algorithms for the General Case of SSMs](#)” on page 1974.

SMC Filtering

The SMC filtering algorithms recursively provide numerical estimates of the sequence of joint posterior densities $p_{\theta}(\mathbf{X}_{1:t}|\mathbf{Y}_{1:t})$ over time t for the given parameters θ based on the weighted particles $\{\mathbf{X}_{1:t}^{(k)}, \tilde{w}_t^{(k)}\}_{k=1}^N$, where N denotes the number of particles, $\mathbf{X}_{1:t}^{(k)}$ denotes the state variable values of the k th particle sequence, and $\tilde{w}_t^{(k)}$ denotes the normalized weight of the k th particle sequence.

The numerical approximations of the joint posterior densities $p_{\theta}(\mathbf{X}_{1:t}|\mathbf{Y}_{1:t})$ over time are given by

$$\hat{p}_{\theta}(\mathbf{X}_{1:t}|\mathbf{Y}_{1:t}) = \sum_{k=1}^N \tilde{w}_t^{(k)} \delta_{\mathbf{X}_{1:t}^{(k)}}(\mathbf{X}_{1:t}), \quad t = 1, \dots, T$$

where $\delta_{\hat{x}}(x)$ denotes the Dirac measure for $\hat{x}$ such that

$$\delta_{\hat{x}}(x) = \begin{cases} 1 & \text{if } \hat{x} = x \\ 0 & \text{otherwise} \end{cases}$$

The conditional expected value of an arbitrary function $h(\mathbf{X}_{1:t})$ given $\mathbf{Y}_{1:t}$ is approximated by

$$\mathbb{E}_{\theta}[h(\mathbf{X}_{1:t})|\mathbf{Y}_{1:t}] = \sum_{k=1}^N \tilde{w}_t^{(k)} h(\mathbf{X}_{1:t}^{(k)}), \quad t = 1, \dots, T$$

It is also straightforward to approximate the marginal densities by

$$\hat{p}_{\theta}(\mathbf{X}_t|\mathbf{Y}_{1:t}) = \sum_{k=1}^N \tilde{w}_t^{(k)} \delta_{\mathbf{X}_t^{(k)}}(\mathbf{X}_t), \quad t = 1, \dots, T$$

Sequential Importance Resampling Algorithm

The sequential importance sampling (SIS) algorithm offers a solution to the filtering problem. In the SIS algorithm, random particles (samples) are propagated by the importance sampling mechanism, and the target posterior densities are then estimated recursively. However, the variance of the importance weights in the SIS algorithm increases exponentially over time. This issue is described as the degeneracy problem in some literature, and it results in poor approximations of the filtering distribution $p_{\theta}(\mathbf{X}_{1:t}|\mathbf{Y}_{1:t})$ when t is large. In practice, after a few iterations of the algorithm, most of the normalized importance weights are sufficiently close to zero, and their contribution to the final estimate is almost zero. The introduction of a resampling procedure at certain time steps is thus necessary in order for the SIS algorithm to discard the particles with negligible weights and multiply the promising ones. The resulting algorithm is known as the sequential importance resampling (SIR) algorithm or sequential importance sampling and resampling (Doucet, de Freitas, and Gordon 2001; Ristic, Arulampalam, and Gordon 2004; Rubin 1988).

The SIS and SIR algorithms rely on weighted particles $\{\mathbf{X}_{1:t}^{(k)}, \tilde{w}_t^{(k)}\}_{k=1}^N$. The particles $\{\mathbf{X}_{1:t}^{(k)}\}_{k=1}^N$ are propagated from a proposal distribution, and their normalized importance weights $\{\tilde{w}_t^{(k)}\}_{k=1}^N$ are updated such that they approximate the densities $p_{\theta}(\mathbf{X}_{1:t}|\mathbf{Y}_{1:t})$. In practice, the resampling step in the SIR algorithm is not necessary at each time step. Only when the variance of importance weights is greater than a prespecified threshold should the particles be resampled. The effective sample size (ESS) (or effective number of particles)

introduced by Liu and Chen (1995) is a widely used measure of that variance. The ESS at the t th time step is estimated as

$$\widehat{N}_{\text{eff}}(t) = \frac{1}{\sum_{k=1}^N (\tilde{w}_t^{(k)})^2}$$

Resampling is implemented when $\widehat{N}_{\text{eff}}$ is below a threshold N_{thres} that is significantly less than N (such as $\frac{N}{10}$). As for the resampling schemes, there are four popular algorithms: stratified resampling, systematic resampling, residual resampling, and multinomial resampling. Stratified resampling, proposed by Kitagawa (1996), is used in the SMC procedure.

The SIR algorithm proceeds as follows:

1. At time $t = 1$:

- For $k = 1, \dots, N$, sample $\mathbf{X}_1^{(k)}$ from the proposal initialization distribution $q_{\theta}^1(\mathbf{X}_1|\mathbf{Y}_1)$.
- For $k = 1, \dots, N$, calculate the importance weights:

$$w_1(\mathbf{X}_1^{(k)}) = \frac{\mu_{\theta}(\mathbf{X}_1^{(k)})g_{\theta}(\mathbf{Y}_1|\mathbf{X}_1)}{q_{\theta}^1(\mathbf{X}_1|\mathbf{Y}_1)}$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_1^{(k)} = \frac{w_1(\mathbf{X}_1^{(k)})}{\sum_{i=1}^N w_1(\mathbf{X}_1^{(i)})}$$

2. At times $t = 2, \dots, T$:

- Calculate $\widehat{N}_{\text{eff}}(t-1)$, and compare it with N_{thres} .
 - If $\widehat{N}_{\text{eff}} \geq N_{\text{thres}}$: for $k = 1, \dots, N$, set $A_{t-1}^{(k)} = k$, and set $V_{t-1}^{(k)} = \tilde{w}_{t-1}^{(k)}$.
 - Otherwise: for $k = 1, \dots, N$, sample index $A_{t-1}^{(k)}$ based on $\{\tilde{w}_{t-1}^{(k)}\}_{k=1}^N$ using a resampling scheme, and set $V_{t-1}^{(k)} = \frac{1}{N}$.
- For $k = 1, \dots, N$, sample $\mathbf{X}_t^{(k)}$ from the proposal distribution $q_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_t)$, and set $\mathbf{X}_{1:t}^{(k)} = (\mathbf{X}_{1:t-1}^{(A_{t-1}^{(k)})}, \mathbf{X}_t^{(k)})$.
- For $k = 1, \dots, N$, calculate the importance weights:¹

$$w_t(\mathbf{X}_{1:t}^{(k)}) = V_{t-1}^{(k)} \frac{f_{\theta}(\mathbf{X}_t^{(k)}|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})})g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t^{(k)})}{q_{\theta}(\mathbf{X}_t^{(k)}|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_t)}$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_t^{(k)} = \frac{w_t(\mathbf{X}_{1:t}^{(k)})}{\sum_{i=1}^N w_t(\mathbf{X}_{1:t}^{(i)})}$$

¹In some literature, the scaling factor $V_{t-1}^{(k)}$ is replaced by the normalized or non-normalized weights at the previous time step that are adjusted if the resampling step is implemented. $V_{t-1}^{(k)}$ is used here so that the original weights can be restored for the following approximations regardless of the possible resampling step.

Bootstrap Filter

The bootstrap filter (BF) algorithm proposed by Gordon, Salmond, and Smith (1993) is an extreme case of the SIR algorithm. It sets $q_{\theta}^1(\mathbf{X}_1|\mathbf{Y}_1) = \mu_{\theta}(\mathbf{X}_1)$ and $q_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1}, \mathbf{Y}_t) = f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})$. The resampling step is carried out at every time step.

The BF algorithm proceeds as follows:

1. At time $t = 1$:

- For $k = 1, \dots, N$, sample $\mathbf{X}_1^{(k)}$ from the initial distribution $\mu_{\theta}(\mathbf{X}_1)$.
- For $k = 1, \dots, N$, calculate the weights:

$$w_1(\mathbf{X}_1^{(k)}) = g_{\theta}(\mathbf{Y}_1|\mathbf{X}_1^{(k)})$$

- For $k = 1, \dots, N$, normalize the weights:

$$\tilde{w}_1^{(k)} = \frac{w_1(\mathbf{X}_1^{(k)})}{\sum_{i=1}^N w_1(\mathbf{X}_1^{(i)})}$$

2. At times $t = 2, \dots, T$:

- For $k = 1, \dots, N$, sample index $A_{t-1}^{(k)}$ based on $\{\tilde{w}_{t-1}^{(k)}\}_{k=1}^N$ using a resampling scheme.
- For $k = 1, \dots, N$, sample $\mathbf{X}_t^{(k)}$ from the transition distribution $f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})})$, and set $\mathbf{X}_{1:t}^{(k)} = (\mathbf{X}_{1:t-1}^{(A_{t-1}^{(k)})}, \mathbf{X}_t^{(k)})$.
- For $k = 1, \dots, N$, calculate the weights:

$$w_t(\mathbf{X}_{1:t}^{(k)}) = g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t^{(k)})$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_t^{(k)} = \frac{w_t(\mathbf{X}_{1:t}^{(k)})}{\sum_{i=1}^N w_t(\mathbf{X}_{1:t}^{(i)})}$$

Auxiliary Particle Filter

The auxiliary particle filter (APF), proposed by Pitt and Shephard (1999), is a popular alternative to the SIR algorithm. At time step t , the APF first resamples weighted particles $\{\mathbf{X}_{t-1}^{(k)}, \tilde{w}_{t-1}^{(k)}\}_{k=1}^N$ to obtain only the ones that are likely to survive after considering the coming observation $\mathbf{Y}_t$. That resampling step is implemented by sampling auxiliary index variables according to an auxiliary distribution that weights particles in terms of their compatibility with the coming observation. Those weights are defined by

$$v_{t-1}^{(k)} = \tilde{w}_{t-1}^{(k)} p_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1}^{(k)}), \quad k = 1, \dots, N$$

where the auxiliary density function $p_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1})$ can be expanded as follows:

$$p_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1}) = \int g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t) f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1}) d\mathbf{X}_t$$

The auxiliary density function is usually replaced by an approximation $\hat{p}_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1})$ because it is not analytically available in most cases. It is suggested in the literature that you set $\hat{p}_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1}) = g_{\theta}(\mathbf{Y}_t|m(\mathbf{X}_{t-1}))$, where $m(\mathbf{X}_{t-1})$ is the mean, the median, a draw, or any other meaningful value associated with the transition density $f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})$. The resampled particles are then used to sample new particles at time step t with the proposal distribution.

The APF algorithm proceeds as follows:²

1. At time $t = 1$:

- For $k = 1, \dots, N$, sample $\mathbf{X}_1^{(k)}$ from the proposal initialization distribution $q_{\theta}^1(\mathbf{X}_1|\mathbf{Y}_1)$.
- For $k = 1, \dots, N$, calculate the importance weights:

$$w_1(\mathbf{X}_1^{(k)}) = \frac{\mu_{\theta}(\mathbf{X}_1^{(k)})g_{\theta}(\mathbf{Y}_1|\mathbf{X}_1)}{q_{\theta}^1(\mathbf{X}_1|\mathbf{Y}_1)}$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_1^{(k)} = \frac{w_1(\mathbf{X}_1^{(k)})}{\sum_{i=1}^N w_1(\mathbf{X}_1^{(i)})}$$

2. At times $t = 2, \dots, T$:

- For $k = 1, \dots, N$, calculate the weights:

$$v_{t-1}^{(k)} = \tilde{w}_{t-1}^{(k)} \hat{p}_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1}^{(k)})$$

- For $k = 1, \dots, N$, normalize the weights:

$$\tilde{v}_{t-1}^{(k)} = \frac{v_{t-1}^{(k)}}{\sum_{i=1}^N v_{t-1}^{(i)}}$$

- For $k = 1, \dots, N$, sample index $A_{t-1}^{(k)}$ with weights $\{\tilde{v}_{t-1}^{(k)}\}_{k=1}^N$ using a resampling scheme.
- For $k = 1, \dots, N$, sample $\mathbf{X}_t^{(k)}$ from the proposal distribution $q_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_t)$, and set $\mathbf{X}_{1:t}^{(k)} = (\mathbf{X}_{1:t-1}^{(A_{t-1}^{(k)})}, \mathbf{X}_t^{(k)})$.
- For $k = 1, \dots, N$, calculate the importance weights:

$$w_t(\mathbf{X}_{1:t}^{(k)}) = \frac{f_{\theta}(\mathbf{X}_t^{(k)}|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})})g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t^{(k)})}{\hat{p}_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})})q_{\theta}(\mathbf{X}_t^{(k)}|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_t)}$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_t^{(k)} = \frac{w_t(\mathbf{X}_{1:t}^{(k)})}{\sum_{i=1}^N w_t(\mathbf{X}_{1:t}^{(i)})}$$

²In some literature, a second resampling step is implemented at the end of each iteration. In practice, this setting is outperformed by the APF with the single resampling scheme that is listed here.

Fully Adapted Auxiliary Particle Filter

The fully adapted auxiliary particle filter (APF(ADP)), introduced by Pitt and Shephard (1999) and Pitt et al. (2012), is a special case of the APF algorithm. Full adaptation is possible when the transition distribution $f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})$ is conjugate in $\mathbf{X}_t$ to the measurement distribution $g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t)$. That is,

$$\frac{g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t)f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})}{\int g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t)f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})d\mathbf{X}_t}$$

is in the same probability distribution family as the transition distribution $f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})$.

When conjugacy holds, the auxiliary density function $p_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1}) = \int g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t)f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})d\mathbf{X}_t$ is analytically available. In addition, the proposal transition distribution that is chosen as

$$q_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1}, \mathbf{Y}_t) = \frac{g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t)f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})}{\int g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t)f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1})d\mathbf{X}_t}$$

has a closed-form expression and can be sampled from easily.

As a result of conjugacy, the importance weights simplify to 1:

$$w_t(\mathbf{X}_{1:t}^{(k)}) = \frac{f_{\theta}(\mathbf{X}_t^{(k)}|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})})g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t^{(k)})}{p_{\theta}(\mathbf{Y}_t|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})})q_{\theta}(\mathbf{X}_t^{(k)}|\mathbf{X}_{t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_t)} = 1$$

The fully adapted auxiliary particle filter has several advantages:

- Because the importance weights simplify to 1, the steps that compute those weights are eliminated from the APF algorithm. This considerably reduces the computing time, especially when the number of particles and/or the number of time points is large.
- Following the guideline for choosing the optimal number of particles provided by Pitt et al. (2012), APF(ADP) requires many fewer particles than BF and APF.
- Pitt et al. (2012) also showed that the estimated log likelihood obtained by APF(ADP) has a much smaller variance than that obtained by BF. This can prevent slow mixing of the Markov chain in the LEARN statement.
- The Markov chain that is estimated in the LEARN statement has a smaller absolute autocorrelation for APF(ADP) than for APF and BF.

Table 27.9 lists examples of conjugate distributions that PROC SMC supports.

Table 27.9 Conjugate Distributions

Measurement Distribution	Transition Distribution
$y_t \sim \text{normal}(\beta_t x_t, \lambda_t^2)$	$x_t \sim \text{normal}(\mu(x_{t-1}), \sigma^2(x_{t-1}))$
$y_t \sim \text{normal}(\mu_t, \frac{1}{x_t})$	$x_t \sim \text{gamma}(a_t x_{t-1}, \frac{\sigma^2}{x_{t-1}})$
$y_t \sim \text{lognormal}(x_t, s_t)$	$x_t \sim \text{normal}(\mu(x_{t-1}), \sigma^2(x_{t-1}))$
$y_t \sim \text{lognormal}(\mu_t, \frac{1}{x_t})$	$x_t \sim \text{gamma}(a_t x_{t-1}, \frac{\sigma^2}{x_{t-1}})$
$y_t \sim \text{gamma}(\alpha_t, x_t)$	$x_t \sim \text{gamma}(a_t x_{t-1}, \frac{\sigma^2}{x_{t-1}})$
$y_t \sim \text{poisson}(a_t x_t)$	$x_t \sim \text{gamma}(b_t x_{t-1}, \frac{\sigma^2}{x_{t-1}})$

SMC Smoothing

The smoothing problem, $p(\mathbf{X}_t | \mathbf{Y}_{1:T})$, $1 \leq t \leq T$, can be solved by marginalizing $p_\theta(\mathbf{X}_{1:T} | \mathbf{Y}_{1:T})$, which is approximated by the SMC filtering algorithms. However, this approximation is poor when $T - t$ is large because the resampling steps cause $p(\mathbf{X}_t | \mathbf{Y}_{1:T})$ to be approximated by only a few different particles. The following sections introduce the alternative algorithms that solve the smoothing problem.

Fixed-Lag Smoothing

Suppose that the number of lags in the fixed-lag smoothing algorithm is λ . This algorithm, proposed by Kitagawa and Sato (2001), relies on the assumption that observations collected at any time step Λ ($\Lambda > t + \lambda$) provide no additional information to $\mathbf{X}_{1:t}$. Therefore, the following approximation is given by implementing an SMC filtering algorithm until time step t' ($t' = \min(t + \lambda, T)$):

$$\hat{p}_\theta(\mathbf{X}_{1:t} | \mathbf{Y}_{1:T}) \approx \hat{p}_\theta(\mathbf{X}_{1:t} | \mathbf{Y}_{1:t'})$$

To clearly describe the smoothing algorithm, $\{\mathbf{S}_t^{(k)}\}_{k=1}^N$ denote the particles for the smoothing estimation at time t and $\{\tilde{w}_t^{(k)}\}_{k=1}^N$ denote the corresponding normalized weights for the particles. As previously mentioned, $\tilde{\mathbf{X}}_{i:j}$ ($i \leq j$), which is equivalent to $\{\tilde{\mathbf{X}}_t\}_{t=i}^j$, denotes a discrete-time sequence that ranges from time step i to time step j . Therefore, given the sequence $\tilde{\mathbf{X}}_{i:j}$, $\tilde{\mathbf{X}}_s$ ($i \leq s \leq j$) denotes the random variable at time step s .

Based on a given SMC filtering algorithm, the fixed-lag smoothing algorithm proceeds as follows:

1. At time $t' = 1$:

- For $k = 1, \dots, N$, sample $\mathbf{X}_{t'}^{(k)}$ and calculate its importance weights $w_{t'}(\mathbf{X}_{t'}^{(k)})$ based on the specific filtering algorithm.
- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_{t'}^{(k)} = \frac{w_{t'}(\mathbf{X}_{t'}^{(k)})}{\sum_{i=1}^N w_{t'}(\mathbf{X}_{t'}^{(i)})}$$

2. At times $t' = 2, \dots, T$:

- For $k = 1, \dots, N$, get the sample index $A_{t'-1}^{(k)}$ based on the specific filtering algorithm.
- For $k = 1, \dots, N$, sample $\mathbf{X}_{t'}^{(k)}$ and set $\tilde{\mathbf{X}}_{1:t'}^{(k)} = (\tilde{\mathbf{X}}_{1:t'-1}^{(A_{t'-1}^{(k)})}, \mathbf{X}_{t'}^{(k)})$, and calculate the importance weights $w_{t'}(\mathbf{X}_{1:t'}^{(k)})$ based on the specific filtering algorithm.
- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_{t'}^{(k)} = \frac{w_{t'}(\mathbf{X}_{1:t'}^{(k)})}{\sum_{i=1}^N w_{t'}(\mathbf{X}_{1:t'}^{(i)})}$$

- If $t' > \lambda$ and $t' < T$, get $t = t' - \lambda$.
 - for $k = 1, \dots, N$, set $\mathbf{S}_t^{(k)} = \tilde{\mathbf{X}}_t^{(k)}$ and $u_t^{(k)} = \tilde{w}_{t'}^{(k)}$
- If $t' = T$, get $t_l = \max(t' - \lambda, 1)$. For $t = t_l, \dots, T$ and $k = 1, \dots, N$,
 - $k = 1, \dots, N$, set $\mathbf{S}_t^{(k)} = \tilde{\mathbf{X}}_t^{(k)}$ and $u_t^{(k)} = \tilde{w}_{t'}^{(k)}$

With the weighted particles $\{\mathbf{S}_t^{(k)}, \tilde{u}_t^{(k)}\}_{k=1}^N$ at each time step t , the smoothing problem is solved by

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{X}_t | \mathbf{Y}_{1:T}) = \sum_{k=1}^N \tilde{u}_t^{(k)} \delta_{\mathbf{S}_t^{(k)}}(\mathbf{X}_t), 1 \leq t \leq T$$

Moreover, the expected value of an arbitrary function $h(\mathbf{X}_t)$ given $\mathbf{Y}_{1:T}$ is approximated by

$$\mathbb{E}_{\boldsymbol{\theta}}[h(\mathbf{X}_t) | \mathbf{Y}_{1:T}] = \sum_{k=1}^N \tilde{u}_t^{(k)} h(\mathbf{S}_t^{(k)})$$

Fixed-Interval Smoothing

The fixed-interval smoothing algorithm, proposed by Doucet, Godsill, and Andrieu (2000), solves the marginal smoothing problem by backward recursion on the basis of the following formula:

$$p_{\boldsymbol{\theta}}(\mathbf{X}_t | \mathbf{Y}_{1:T}) = p_{\boldsymbol{\theta}}(\mathbf{X}_t | \mathbf{Y}_{1:t}) \int \frac{f_{\boldsymbol{\theta}}(\mathbf{X}_{t+1} | \mathbf{X}_t) p_{\boldsymbol{\theta}}(\mathbf{X}_{t+1} | \mathbf{Y}_{1:T})}{p_{\boldsymbol{\theta}}(\mathbf{X}_{t+1} | \mathbf{Y}_{1:t})} d\mathbf{X}_{t+1}, \quad t = 1, \dots, T$$

Given the weighted particles $\{\mathbf{X}_t^{(k)}, \tilde{w}_t^{(k)}\}_{k=1}^N$ obtained by a filtering algorithm until time step T , the weights in the backward recursions can be calculated as follows:

1. At time $t = T$:
 - For $k = 1, \dots, N$, initialize $\tilde{w}_{T|T}^{(k)} = \tilde{w}_T^{(k)}$
2. At times $t = T - 1, \dots, 1$:
 - For $k = 1, \dots, N$, evaluate the importance weights

$$\tilde{w}_{t|T}^{(k)} = \sum_{j=1}^N \tilde{w}_{t+1|T}^{(j)} \frac{\tilde{w}_t^{(k)} f_{\boldsymbol{\theta}}(\mathbf{X}_{t+1}^{(j)} | \mathbf{X}_t^{(k)})}{\sum_{i=1}^N \tilde{w}_t^{(i)} f_{\boldsymbol{\theta}}(\mathbf{X}_{t+1}^{(j)} | \mathbf{X}_t^{(i)})}$$

The reweighted particles $\{\mathbf{X}_t^{(k)}, \tilde{w}_{t|T}^{(k)}\}_{k=1}^N$ after the backward recursion approximates the smoothing distribution $p_{\theta}(\mathbf{X}_t|\mathbf{Y}_{1:T})$ as

$$\hat{p}_{\theta}(\mathbf{X}_t|\mathbf{Y}_{1:T}) = \sum_{k=1}^N \tilde{w}_{t|T}^{(k)} \delta_{\mathbf{X}_t^{(k)}}(\mathbf{X}_t), 1 \leq t \leq T$$

Moreover, the expected value of an arbitrary function $h(\mathbf{X}_t)$ given $\mathbf{Y}_{1:T}$ is approximated by

$$\mathbb{E}_{\theta}[h(\mathbf{X}_t)|\mathbf{Y}_{1:T}] = \sum_{k=1}^N \tilde{w}_{t|T}^{(k)} h(\mathbf{X}_t^{(k)}), 1 \leq t \leq T$$

SMC Forecasting

At logical time step T , the L -step-ahead forecasting problem is solved by extending the weighted particles $\{\mathbf{X}_{1:T}^{(k)}, \tilde{w}_T^{(k)}\}_{k=1}^N$ obtained using an SMC filtering algorithm. The SMC filtering algorithms usually do not resample at the last time step. An extra resampling step might be implemented at time step T . If so, the normalized weights of all resampled particles denoted by $\{\tilde{w}_T^{(k)}\}_{k=1}^N$ are set to $\frac{1}{N}$. For ease of presentation, the same symbol for the weighted particles is used no matter whether the resampling step is implemented at time step T . In the forecasting stage, no resampling step is needed.

The SMC forecasting algorithm proceeds as follows:

1. For $k = 1, \dots, N$, set $\tilde{w}_{x,T}^{(k)} = \tilde{w}_T^{(k)}$, where $\tilde{w}_T^{(k)}$ is derived from the filtering process.
2. For $t = T + 1$ to $T + L$:
 - For $k = 1, \dots, N$, sample $\mathbf{X}_t^{(k)}$ from the transition distribution $f_{\theta}(\mathbf{X}_t|\mathbf{X}_{t-1}^{(k)})$.
 - For $k = 1, \dots, N$, set $\mathbf{X}_{1:t}^{(k)} = (\mathbf{X}_{1:t-1}^{(k)}, \mathbf{X}_t^{(k)})$, and sample $\mathbf{Y}_t^{(k)}$ from the measurement distribution $g_{\theta}(\mathbf{Y}_t|\mathbf{X}_t^{(k)})$.
 - For $k = 1, \dots, N$, calculate the normalized weights of the state and dependent variables:

$$\tilde{w}_{x,t}^{(k)} = \tilde{w}_{y,t}^{(k)} = \tilde{w}_{x,t-1}^{(k)}$$

The weighted particles for the state and dependent variables are obtained. Here the transition density for forecasting should not rely on the observations after time step T . The approximations of $p_{\theta}(\mathbf{X}_t|\mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T})$ and $p_{\theta}(\mathbf{Y}_t|\mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T})$ for $t = T + 1, \dots, T + L$ are given by

$$\begin{aligned} \hat{p}_{\theta}(\mathbf{X}_t|\mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}) &= \sum_{k=1}^N \tilde{w}_{x,t}^{(k)} \delta_{\mathbf{X}_t^{(k)}}(\mathbf{X}_t) \\ \hat{p}_{\theta}(\mathbf{Y}_t|\mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}) &= \sum_{k=1}^N \tilde{w}_{y,t}^{(k)} \delta_{\mathbf{Y}_t^{(k)}}(\mathbf{Y}_t) \end{aligned}$$

Moreover, the conditional expected value of an arbitrary function $h(\mathbf{X}_t)$ or $h(\mathbf{Y}_t)$ given $\mathbf{Y}_{1:T}$ is approximated by

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\theta}}[h(\mathbf{X}_t)|\mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}] &= \sum_{k=1}^N \tilde{w}_{x,t}^{(k)} h(\mathbf{X}_t^{(k)}), \quad t = T+1, \dots, T+L \\ \mathbb{E}_{\boldsymbol{\theta}}[h(\mathbf{Y}_t)|\mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}] &= \sum_{k=1}^N \tilde{w}_{y,t}^{(k)} h(\mathbf{Y}_t^{(k)}), \quad t = T+1, \dots, T+L\end{aligned}$$

SMC Evaluating

The marginal likelihood is defined by

$$p_{\boldsymbol{\theta}}(\mathbf{Y}_{1:t}) = \int p_{\boldsymbol{\theta}}(\mathbf{Y}_{1:t}, \mathbf{X}_{1:t}) d\mathbf{X}_{1:t}$$

Monte Carlo methods can directly estimate this integral, but the approximation is usually poor in practice. Alternatively, the marginal likelihood ($t > 1$) is decomposed as

$$p_{\boldsymbol{\theta}}(\mathbf{Y}_{1:t}) = p_{\boldsymbol{\theta}}(\mathbf{Y}_1) \prod_{j=2}^t p_{\boldsymbol{\theta}}(\mathbf{Y}_j | \mathbf{Y}_{1:j-1})$$

For $j = 2, \dots, t$, it is straightforward to have

$$\begin{aligned}p_{\boldsymbol{\theta}}(\mathbf{Y}_j | \mathbf{Y}_{1:j-1}) &= \int p_{\boldsymbol{\theta}}(\mathbf{Y}_j | \mathbf{X}_{j-1}) p_{\boldsymbol{\theta}}(\mathbf{X}_{j-1} | \mathbf{Y}_{1:j-1}) d\mathbf{X}_{j-1} \\ &= \int \int g_{\boldsymbol{\theta}}(\mathbf{Y}_j | \mathbf{X}_j) f_{\boldsymbol{\theta}}(\mathbf{X}_j | \mathbf{X}_{j-1}) p_{\boldsymbol{\theta}}(\mathbf{X}_{j-1} | \mathbf{Y}_{1:j-1}) d\mathbf{X}_{j-1} d\mathbf{X}_j\end{aligned}$$

These terms can be estimated by the SMC algorithms with the weighted particles at time step $j-1$ as

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j | \mathbf{Y}_{1:j-1}) = \sum_{k=1}^N \tilde{w}_{j-1}^{(k)} \int g_{\boldsymbol{\theta}}(\mathbf{Y}_j | \mathbf{X}_j) f_{\boldsymbol{\theta}}(\mathbf{X}_j | \mathbf{X}_{j-1}) d\mathbf{X}_j, \quad j > 1$$

SIR Algorithm

In the SIR algorithm, this approximation can be further simplified by sampling $\mathbf{X}_j^{(k)}$ ($j > 1$) from $q_{\boldsymbol{\theta}}(\mathbf{X}_j | \mathbf{X}_{j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_j)$ for $k = 1, \dots, N$:

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j | \mathbf{Y}_{1:j-1}) = \sum_{k=1}^N V_{j-1}^{(k)} \frac{g_{\boldsymbol{\theta}}(\mathbf{Y}_j | \mathbf{X}_j^{(k)}) f_{\boldsymbol{\theta}}(\mathbf{X}_j^{(k)} | \mathbf{X}_{j-1}^{(A_{j-1}^{(k)})})}{q_{\boldsymbol{\theta}}(\mathbf{X}_j | \mathbf{X}_{j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_j)} = \sum_{k=1}^N w_j(\mathbf{X}_{1:j}^{(k)})$$

Moreover, $p_{\boldsymbol{\theta}}(\mathbf{Y}_1)$ can be written as the following integral with the importance sampling scheme:

$$p_{\boldsymbol{\theta}}(\mathbf{Y}_1) = \int \frac{\mu_{\boldsymbol{\theta}}(\mathbf{X}_1) g_{\boldsymbol{\theta}}(\mathbf{Y}_1 | \mathbf{X}_1)}{q_{\boldsymbol{\theta}}^1(\mathbf{X}_1 | \mathbf{Y}_1)} q_{\boldsymbol{\theta}}^1(\mathbf{X}_1 | \mathbf{Y}_1) d\mathbf{X}_1$$

In the SIR algorithm, $\{\mathbf{X}_1^{(k)}\}_{k=1}^N$ are sampled from $q_{\boldsymbol{\theta}}^1(\mathbf{X}_1|\mathbf{Y}_1)$. Therefore, the basic Monte Carlo approximation of $p_{\boldsymbol{\theta}}(\mathbf{Y}_1)$ is

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_1) = \frac{1}{N} \sum_{k=1}^N w_1(\mathbf{X}_1^{(k)})$$

BF Algorithm

In the BF algorithm, this approximation can be simplified by sampling $\mathbf{X}_j^{(k)}$ ($j > 1$) from $f_{\boldsymbol{\theta}}(\mathbf{X}_j^{(k)}|\mathbf{X}_{j-1}^{(A_{j-1}^{(k)})})$ for $k = 1, \dots, N$:

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{Y}_{1:j-1}) = \sum_{k=1}^N \frac{1}{N} \frac{g_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{X}_j^{(k)}) f_{\boldsymbol{\theta}}(\mathbf{X}_j^{(k)}|\mathbf{X}_{j-1}^{(A_{j-1}^{(k)})})}{f_{\boldsymbol{\theta}}(\mathbf{X}_j^{(k)}|\mathbf{X}_{j-1}^{(A_{j-1}^{(k)})})} = \frac{1}{N} \sum_{k=1}^N w_j(\mathbf{X}_{1:j}^{(k)})$$

Moreover, the $\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_1)$ in the BF algorithm is computed in the same way as the one in the SIR algorithm.

APF Algorithm

In the APF algorithm, because $v_{j-1}^{(k)} = \tilde{w}_{j-1}^{(k)} \hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{X}_{j-1}^{(k)})$, the approximation can be written as

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{Y}_{1:j-1}) = \sum_{k=1}^N \frac{v_{j-1}^{(k)} \sum_{i=1}^N v_{j-1}^{(i)}}{(\sum_{i=1}^N v_{j-1}^{(i)}) \hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{X}_{j-1}^{(k)})} \int g_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{X}_j) f_{\boldsymbol{\theta}}(\mathbf{X}_j|\mathbf{X}_{j-1}^{(k)}) d\mathbf{X}_j, \quad j > 1$$

The auxiliary index variables $\{A_{j-1}^{(k)}\}_{k=1}^N$ are sampled according to $\left\{ \frac{v_{j-1}^{(k)}}{\sum_{i=1}^N v_{j-1}^{(i)}} \right\}_{k=1}^N$, and $\mathbf{X}_j^{(k)}$ are sampled from $q_{\boldsymbol{\theta}}(\mathbf{X}_j|\mathbf{X}_{j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_j)$. The weight for each auxiliary index is $\frac{1}{N}$. The preceding approximation is thus simplified as

$$\begin{aligned} \hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{Y}_{1:j-1}) &= \sum_{k=1}^N \frac{1}{N} \left(\sum_{i=1}^N v_{j-1}^{(i)} \right) \frac{g_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{X}_j^{(k)}) f_{\boldsymbol{\theta}}(\mathbf{X}_j^{(k)}|\mathbf{X}_{j-1}^{(A_{j-1}^{(k)})})}{\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{X}_{j-1}^{(A_{j-1}^{(k)})}) q_{\boldsymbol{\theta}}(\mathbf{X}_j^{(k)}|\mathbf{X}_{j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_j)} \\ &= \frac{1}{N} \left(\sum_{i=1}^N v_{j-1}^{(i)} \right) \left(\sum_{k=1}^N w_j(\mathbf{X}_{1:j}^{(k)}) \right). \end{aligned}$$

It is further simplified to $\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_j|\mathbf{Y}_{1:j-1}) = \sum_{i=1}^N v_{j-1}^{(i)}$ if you use APF(ADP). Moreover, the $\hat{p}_{\boldsymbol{\theta}}(\mathbf{Y}_1)$ in the APF algorithm is computed in the same way as the one in the SIR algorithm.

SMC Learning

In most cases, the learning problem is the most difficult one. One of the SSM parameter estimation methods is the particle Markov chain Monte Carlo (PMCMC) algorithm, which combines the SMC and Markov chain Monte Carlo methods (Andrieu, Doucet, and Holenstein 2010).

Particle Marginal Metropolis-Hastings Sampler

The particle marginal Metropolis-Hastings algorithm directly targets the marginal density $p(\theta | \mathbf{Y}_{1:T})$. This algorithm proceeds as follows:

1. Initialization, $i = 0$:
 - Set $\theta[0]$ to the initial parameter values and run the SMC filtering algorithm with $\theta[i]$ until the time step T . Calculate $\hat{p}_{\theta[i]}(\mathbf{Y}_{1:T})$ using one SMC evaluating scheme.
2. For $i = 1, \dots, M$ (M is the number of samples):
 - Sample θ^* from the random walk Metropolis sampler based on $\theta[m-1]$, and run the SMC filtering algorithm with θ^* until T . Calculate $\hat{p}_{\theta^*}(\mathbf{Y}_{1:T})$ using the SMC evaluating scheme.
 - Calculate the acceptance probability $\alpha_m = \min\left(\frac{\hat{p}_{\theta^*}(\mathbf{Y}_{1:T})p(\theta^*)}{\hat{p}_{\theta[m-1]}(\mathbf{Y}_{1:T})p(\theta[m-1])} \frac{q(\theta[m-1]|\theta^*)}{q(\theta^*|\theta[m-1])}, 1\right)$.
 - Sample u_m uniformly over $[0, 1]$.
 - If $u_m \leq \alpha_m$, then set $\theta[m] = \theta^*$ and $\hat{p}_{\theta[m]}(\mathbf{Y}_{1:T}) = \hat{p}_{\theta^*}(\mathbf{Y}_{1:T})$. Otherwise, set $\theta[m] = \theta[m-1]$ and $\hat{p}_{\theta[m]}(\mathbf{Y}_{1:T}) = \hat{p}_{\theta[m-1]}(\mathbf{Y}_{1:T})$.

The random walk Metropolis sampler is used to draw parameters. Therefore, $q(\theta[m-1]|\theta^*) = q(\theta^*|\theta[m-1])$ holds in the algorithm.

SMC Algorithms for the General Case of SSMs

This section discusses the SMC algorithms for the general case of SSMs, where any of the following conditions might exist:

- There exist some independent variables.
- The maximum lag order of all the state variables for the measurement distribution is greater than or equal to 0.
- The maximum lag order of all the state variables for the transition (or proposal transition) distribution is greater than or equal to 0.
- The maximum lag order of the dependent variables in any distribution is greater than or equal to 0.

The notations τ , r , p , and s are defined in the section “[Sequential Monte Carlo Methods and State Space Models](#)” on page 1961. For convenience, the definition is copied here. The value r can be seen as the maximum lag order of all the state variables; p can be seen as the maximum lag order of all the dependent

variables; s can be seen as the maximum lag order of the independent variables; and τ can be seen as the maximum lag order of the state variables that must be generated in the initialization (or proposal initialization) distribution. For more information about the value of τ , see the sections “[Method 1: Maximum Lag Order of State Variables to Be Generated in Initialization Distribution, \$\tau\$](#) ” on page 1986 (Method 1) and “[Method 2: Maximum Lag Order of State Variables to Be Generated in Initialization Distribution, \$\tau\$](#) ” on page 1989 (Method 2).

SMC Filtering Algorithm

Sequential Importance Resampling

The sequential importance sampling (SIS) algorithm offers a solution to the filtering problem. In the SIS algorithm, random particles (samples) are propagated by the importance sampling mechanism, and the target posterior densities are then estimated recursively. However, the variance of the importance weights in the SIS algorithm increases exponentially over time. This issue is described as the degeneracy problem in some literature, and it results in poor approximations of the filtering distribution $p_{\theta}(\mathbf{X}_{1-\tau:t}|\mathbf{Y}_{1-p:t}, \mathbf{Z}_{1-s:t})$ when t is large. In practice, after a few iterations of the algorithm, most of the normalized importance weights are sufficiently close to zero, and their contribution to the final estimate is almost zero. The introduction of a resampling procedure at certain time steps is thus necessary in order for the SIS algorithm to discard the particles with negligible weights and multiply the promising ones. The resulting algorithm is known as the sequential importance resampling (SIR) algorithm or sequential importance sampling and resampling (Doucet, de Freitas, and Gordon 2001; Ristic, Arulampalam, and Gordon 2004; Rubin 1988).

The SIS and SIR algorithms rely on weighted particles $\{\mathbf{X}_{1-\tau:t}^{(k)}, \tilde{w}_t^{(k)}\}_{k=1}^N$. The particles $\{\mathbf{X}_{1:t}^{(k)}\}_{k=1}^N$ are propagated from a proposal distribution, and their normalized importance weights $\{\tilde{w}_t^{(k)}\}_{k=1}^N$ are updated such that they approximate the densities $p_{\theta}(\mathbf{X}_{1-\tau:t}|\mathbf{Y}_{1-p:t}, \mathbf{Z}_{1-s:t})$. In practice, the resampling step in the SIR algorithm is not necessary at each time step. Only when the variance of importance weights is greater than a pre-specified threshold should the particles be resampled. The effective sample size (ESS) (or effective number of particles) introduced by Liu and Chen (1995) is a widely used measure of that variance. The ESS at t th time step is estimated as

$$\widehat{N}_{\text{eff}}(t) = \frac{1}{\sum_{k=1}^N (\tilde{w}_t^{(k)})^2}$$

Resampling is implemented when $\widehat{N}_{\text{eff}}$ is below a threshold N_{thres} that is significantly less than N (such as $\frac{N}{10}$). As for the resampling schemes, there are four popular algorithms: stratified resampling, systematic resampling, residual resampling, and multinomial resampling. Stratified resampling, proposed by Kitagawa (1996), is used in the SMC procedure.

The SIR algorithm proceeds as follows:

1. At time $t = 1$:

- For $k = 1, \dots, N$, sample $\mathbf{X}_{1-\tau:1}^{(k)}$ from the proposal initialization distribution $q_{\theta}^1(\mathbf{X}_{1-\tau:1}|\mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})$.
- For $k = 1, \dots, N$, calculate the importance weights:

$$w_1(\mathbf{X}_{1-\tau:1}^{(k)}) = \frac{\mu_{\theta}(\mathbf{X}_{1-\tau:1}^{(k)}|\mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})g_{\theta}(\mathbf{Y}_1|\mathbf{X}_{1-\tau:1}, \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})}{q_{\theta}^1(\mathbf{X}_{1-\tau:1}|\mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})}$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_1^{(k)} = \frac{w_1(\mathbf{X}_{1-\tau:1}^{(k)})}{\sum_{i=1}^N w_1(\mathbf{X}_{1-\tau:1}^{(i)})}$$

2. At times $t = 2, \dots, T$:

- Calculate $\widehat{N}_{\text{eff}}(t-1)$, and compare it with N_{thres} .
 - If $\widehat{N}_{\text{eff}} \geq N_{\text{thres}}$: for $k = 1, \dots, N$, set $A_{t-1}^{(k)} = k$, and set $V_{t-1}^{(k)} = \tilde{w}_{t-1}^{(k)}$.
 - Otherwise: for $k = 1, \dots, N$, sample index $A_{t-1}^{(k)}$ based on $\{\tilde{w}_{t-1}^{(k)}\}_{k=1}^N$ using a resampling scheme, and set $V_{t-1}^{(k)} = \frac{1}{N}$.
- For $k = 1, \dots, N$, sample $\mathbf{X}_t^{(k)}$ from the proposal distribution $q_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$, and set $\mathbf{X}_{1-\tau:t}^{(k)} = (\mathbf{X}_{1-\tau:t-1}^{(A_{t-1}^{(k)})}, \mathbf{X}_t^{(k)})$.
- For $k = 1, \dots, N$, calculate the importance weights³:

$$w_t(\mathbf{X}_{1-\tau:t}^{(k)}) = V_{t-1}^{(k)} \frac{f_{\theta}(\mathbf{X}_t^{(k)} | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) g_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t}^{(k)}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})}{q_{\theta}(\mathbf{X}_t^{(k)} | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t})}$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_t^{(k)} = \frac{w_t(\mathbf{X}_{1-\tau:t}^{(k)})}{\sum_{i=1}^N w_t(\mathbf{X}_{1-\tau:t}^{(i)})}$$

Bootstrap Filter

The bootstrap filter (BF) algorithm proposed by Gordon, Salmond, and Smith (1993) is an extreme case of the SIR algorithm. It sets $q_{\theta}^1(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1}) = \mu_{\theta}(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$ and $q_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t}) = f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$. The resampling step is carried out at every time step.

The BF algorithm proceeds as follows:

1. At time $t = 1$:

- For $k = 1, \dots, N$, sample $\mathbf{X}_{1-\tau:1}^{(k)}$ from the initial distribution $\mu_{\theta}(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$.
- For $k = 1, \dots, N$, calculate the weights:

$$w_1(\mathbf{X}_{1-\tau:1}^{(k)}) = g_{\theta}(\mathbf{Y}_1 | \mathbf{X}_{1-\tau:1}^{(k)}, \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$$

- For $k = 1, \dots, N$, normalize the weights:

$$\tilde{w}_1^{(k)} = \frac{w_1(\mathbf{X}_{1-\tau:1}^{(k)})}{\sum_{i=1}^N w_1(\mathbf{X}_{1-\tau:1}^{(i)})}$$

³In some literature, the scaling factor $V_{t-1}^{(k)}$ is replaced by the normalized or non-normalized weights at the previous time step that are adjusted if the resampling step is implemented. $V_{t-1}^{(k)}$ is used here so that the original weights can be restored for the following approximations regardless of the possible resampling step.

2. At times $t = 2, \dots, T$:

- For $k = 1, \dots, N$, sample index $A_{t-1}^{(k)}$ based on $\{\tilde{w}_{t-1}^{(k)}\}_{k=1}^N$ using a resampling scheme.
- For $k = 1, \dots, N$, sample $\mathbf{X}_t^{(k)}$ from the transition distribution $f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$, and set $\mathbf{X}_{1-\tau:t}^{(k)} = (\mathbf{X}_{1-\tau:t-1}^{(A_{t-1}^{(k)})}, \mathbf{X}_t^{(k)})$.
- For $k = 1, \dots, N$, calculate the weights:

$$w_t(\mathbf{X}_{1-\tau:t}^{(k)}) = g_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t}^{(k)}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_t^{(k)} = \frac{w_t(\mathbf{X}_{1-\tau:t}^{(k)})}{\sum_{i=1}^N w_t(\mathbf{X}_{1-\tau:t}^{(i)})}$$

Auxiliary Particle Filter

The auxiliary particle filter (APF), proposed by Pitt and Shephard (1999), is a popular alternative to the SIR method. At time step t , the APF first resamples weighted particles $\{\mathbf{X}_{t-r:t-1}^{(k)}, \tilde{w}_{t-1}^{(k)}\}_{k=1}^N$ to obtain only the ones that are likely to survive after considering the coming observation $\mathbf{Y}_t$. That resampling step is implemented by sampling auxiliary index variables according to an auxiliary distribution that weights particles in terms of their compatibility with the coming observation. Those weights are defined by

$$v_{t-1}^{(k)} = \tilde{w}_{t-1}^{(k)} p_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}^{(k)}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}), \quad k = 1, \dots, N$$

where the auxiliary density function $p_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$ can be expanded as follows:

$$\begin{aligned} & p_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) \\ &= \int g_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) d\mathbf{X}_t \end{aligned}$$

The auxiliary density function is usually replaced by an approximation $\hat{p}_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$ because it is not analytically available in most cases. It is suggested in the literature that you set $\hat{p}_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) = g_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, m(\mathbf{X}_{t-r:t-1}), \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_t)$, where $m(\mathbf{X}_{t-r:t-1})$ is the mean, the median, a draw, or any other meaningful value associated with the transition density $f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$. The resampled particles are then used to sample new particles at time step t with the proposal distribution.

The APF algorithm proceeds as follows:⁴

1. At time $t = 1$:

- For $k = 1, \dots, N$, sample $\mathbf{X}_{1-\tau:1}^{(k)}$ from the proposal initialization distribution $q_{\theta}^1(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})$.
- For $k = 1, \dots, N$, calculate the importance weights:

$$w_1(\mathbf{X}_{1-\tau:1}^{(k)}) = \frac{\mu_{\theta}(\mathbf{X}_{1-\tau:1}^{(k)} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1}) g_{\theta}(\mathbf{Y}_1 | \mathbf{X}_{1-\tau:1}, \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})}{q_{\theta}^1(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})}$$

⁴In some literature, a second resampling step is implemented at the end of each iteration. In practice, this setting is outperformed by the APF with the single resampling scheme that is listed here.

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_1^{(k)} = \frac{w_1(\mathbf{X}_{1-\tau:1}^{(k)})}{\sum_{i=1}^N w_\tau(\mathbf{X}_{1-\tau:1}^{(i)})}$$

2. At times $t = 2, \dots, T$:

- For $k = 1, \dots, N$, calculate the weights:

$$v_{t-1}^{(k)} = \tilde{w}_{t-1}^{(k)} \hat{p}_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}^{(k)}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$$

- For $k = 1, \dots, N$, normalize the weights:

$$\tilde{v}_{t-1}^{(k)} = \frac{v_{t-1}^{(k)}}{\sum_{i=1}^N v_{t-1}^{(i)}}$$

- For $k = 1, \dots, N$, sample index $A_{t-1}^{(k)}$ with weights $\{\tilde{v}_{t-1}^{(k)}\}_{k=1}^N$ using a resampling scheme.
- For $k = 1, \dots, N$, sample $\mathbf{X}_t^{(k)}$ from the proposal distribution $q_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t})$, and set $\mathbf{X}_{1-\tau:t}^{(k)} = (\mathbf{X}_{1-\tau:t-1}^{(A_{t-1}^{(k)})}, \mathbf{X}_t^{(k)})$.
- For $k = 1, \dots, N$, calculate the importance weights:

$$w_t(\mathbf{X}_{1-\tau:t}^{(k)}) = \frac{f_\theta(\mathbf{X}_t^{(k)} | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) g_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})}{\hat{p}_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) q_\theta(\mathbf{X}_t^{(k)} | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t})}$$

- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_t^{(k)} = \frac{w_t(\mathbf{X}_{1-\tau:t}^{(k)})}{\sum_{i=1}^N w_t(\mathbf{X}_{1-\tau:t}^{(i)})}$$

Fully Adapted Auxiliary Particle Filter

The fully adapted auxiliary particle filter (APF(ADP)), introduced by Pitt and Shephard (1999) and Pitt et al. (2012), is special a case of the APF algorithm. Full adaptation is possible when the transition distribution $f_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$ is conjugate in $\mathbf{X}_t$ to the measurement distribution $g_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$. That is,

$$\frac{g_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) f_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})}{\int g_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) f_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) d\mathbf{X}_t}$$

is in the same probability distribution family as the transition distribution $f_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})$.

When conjugacy holds, the auxiliary density function $p_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) = \int g_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) f_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) d\mathbf{X}_t$ is analytically available. In addition, the proposal transition distribution that is chosen as

$$q_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t}) = \frac{g_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) f_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})}{\int g_\theta(\mathbf{Y}_t | \mathbf{X}_{t-r:t}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) f_\theta(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) d\mathbf{X}_t}$$

has a closed-form expression and can be sampled from easily.

As a result of conjugacy, the importance weights simplify to 1:

$$w_t(\mathbf{X}_{1-\tau:t}^{(k)}) = \frac{f_{\theta}(\mathbf{X}_t^{(k)} | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) g_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t}^{(k)}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})}{p_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) q_{\theta}(\mathbf{X}_t^{(k)} | \mathbf{X}_{t-r:t-1}^{(A_{t-1}^{(k)})}, \mathbf{Y}_{t-p:t}, \mathbf{Z}_{t-s:t})} = 1$$

The fully adapted auxiliary particle filter has several advantages:

- Because the importance weights simplify to 1, the steps that compute those weights are eliminated from the APF algorithm. This considerably reduces the computing time, especially when the number of particles and/or the number of time points is large.
- Following the guideline on choosing the optimal number of particles provided by Pitt et al. (2012), APF(ADP) requires many fewer particles than BF and APF.
- Pitt et al. (2012) also showed that the estimated log likelihood obtained by APF(ADP) has a much smaller variance than that obtained by BF. This can prevent slow mixing of the Markov chain in the LEARN statement.
- The Markov chain that is estimated in the LEARN statement has a smaller absolute autocorrelation for APF(ADP) than for APF and BF.

Fixed-Lag Smoothing

Suppose that the number of lags in the fixed-lag smoothing algorithm is λ . This algorithm, proposed by Kitagawa and Sato (2001), relies on the assumption that observations collected at any time step Λ ($\Lambda > t + \lambda$) provide no additional information to $\mathbf{X}_{1-\tau:t}$. Therefore, the following approximation is given by implementing an SMC filtering algorithm until time step t' ($t' = \min(t + \lambda, T)$):

$$\hat{p}_{\theta}(\mathbf{X}_{1-\tau:t} | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}) \approx \hat{p}_{\theta}(\mathbf{X}_{1-\tau:t} | \mathbf{Y}_{1-p:t'}, \mathbf{Z}_{1-s:t'})$$

To clearly describe the smoothing algorithm, $\{\mathbf{S}_t^{(k)}\}_{k=1}^N$ denote the particles for the smoothing estimation at time t and $\{\tilde{w}_t^{(k)}\}_{k=1}^N$ denote the corresponding normalized weights for the particles. As previously mentioned, $\tilde{\mathbf{X}}_{i:j}$ ($i \leq j$), which is equivalent to $\{\tilde{\mathbf{X}}_t\}_{t=i}^j$, denotes a discrete-time sequence that ranges from time step i to time step j . Therefore, given the sequence $\tilde{\mathbf{X}}_{i:j}$, $\tilde{\mathbf{X}}_s$ ($i \leq s \leq j$) denotes the random variable at time step s .

Based on a given SMC filtering algorithm, the fixed-lag smoothing algorithm proceeds as follows:

1. At time $t' = 1$:

- For $k = 1, \dots, N$, sample $\mathbf{X}_{1-\tau:t'}^{(k)}$ and calculate its importance weights $w_{t'}(\mathbf{X}_{1-\tau:t'}^{(k)})$ based on the specific filtering algorithm.
- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_{t'}^{(k)} = \frac{w_{t'}(\mathbf{X}_{1-\tau:t'}^{(k)})}{\sum_{i=1}^N w_{t'}(\mathbf{X}_{1-\tau:t'}^{(i)})}$$

2. At times $t' = 2, \dots, T$:

- For $k = 1, \dots, N$, get the sample index $A_{t'-1}^{(k)}$ based on the specific filtering algorithm.
- For $k = 1, \dots, N$, sample $\mathbf{X}_{t'}^{(k)}$ and set $\tilde{\mathbf{X}}_{1-\tau:t'}^{(k)} = (\tilde{\mathbf{X}}_{1-\tau:t'-1}^{(A_{t'-1}^{(k)})}, \mathbf{X}_{t'}^{(k)})$, and calculate the importance weights $w_{t'}(\mathbf{X}_{1-\tau:t'}^{(k)})$ based on the specific filtering algorithm.
- For $k = 1, \dots, N$, normalize the importance weights:

$$\tilde{w}_{t'}^{(k)} = \frac{w_{t'}(\mathbf{X}_{1-\tau:t'}^{(k)})}{\sum_{i=1}^N w_{t'}(\mathbf{X}_{1-\tau:t'}^{(i)})}$$

- If $t' > \lambda$ and $t' < T$, get $t = t' - \lambda$.
 - If $t = 1$: for $t_s = 1 - \tau, \dots, t$,
 - * for $k = 1, \dots, N$, set $\mathbf{S}_{t_s}^{(k)} = \tilde{\mathbf{X}}_{t_s}^{(k)}$ and $\tilde{u}_{t_s}^{(k)} = \tilde{w}_{t'}^{(k)}$.
 - Otherwise: for $k = 1, \dots, N$, set $\mathbf{S}_t^{(k)} = \tilde{\mathbf{X}}_t^{(k)}$ and $\tilde{u}_t^{(k)} = \tilde{w}_{t'}^{(k)}$.
- If $t' = T$, get $t_l = \max(t' - \lambda, 1)$. For $t = t_l, \dots, T$,
 - If $t = 1$: for $t_s = 1 - \tau, \dots, t$,
 - * for $k = 1, \dots, N$, set $\mathbf{S}_{t_s}^{(k)} = \tilde{\mathbf{X}}_{t_s}^{(k)}$ and $\tilde{u}_{t_s}^{(k)} = \tilde{w}_{t'}^{(k)}$.
 - Otherwise: for $k = 1, \dots, N$, set $\mathbf{S}_t^{(k)} = \tilde{\mathbf{X}}_t^{(k)}$ and set $\tilde{u}_t^{(k)} = \tilde{w}_{t'}^{(k)}$.

With the weighted particles $\{\mathbf{S}_t^{(k)}, \tilde{u}_t^{(k)}\}_{k=1}^N$ at each time step t , the smoothing problem is solved by

$$\hat{p}_{\boldsymbol{\theta}}(\mathbf{X}_t | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}) = \sum_{k=1}^N \tilde{u}_t^{(k)} \delta_{\mathbf{S}_t^{(k)}}(\mathbf{X}_t), 1 - \tau \leq t \leq T$$

Moreover, the expected value of an arbitrary function $h(\mathbf{X}_t)$ given $\mathbf{Y}_{1-p:T}$ and $\mathbf{Z}_{1-s:T}$ is approximated by

$$\mathbb{E}_{\boldsymbol{\theta}}[h(\mathbf{X}_t) | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}] = \sum_{k=1}^N \tilde{u}_t^{(k)} h(\mathbf{S}_t^{(k)})$$

Fixed-Interval Smoothing

The fixed-interval smoothing algorithm, proposed by Doucet, Godsill, and Andrieu (2000), solves the marginal smoothing problem by backward recursion on the basis of the following formula:

$$p_{\boldsymbol{\theta}}(\mathbf{X}_t | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}) = p_{\boldsymbol{\theta}}(\mathbf{X}_t | \mathbf{Y}_{1-p:t}, \mathbf{Z}_{1-s:t}) \int \frac{f_{\boldsymbol{\theta}}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t}) p_{\boldsymbol{\theta}}(\mathbf{X}_{t+1} | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T})}{p_{\boldsymbol{\theta}}(\mathbf{X}_{t+1} | \mathbf{Y}_{1-p:t}, \mathbf{Z}_{1-s:t})} d\mathbf{X}_{t+1}$$

Given the weighted particles $\{\mathbf{X}_t^{(k)}, \tilde{w}_t^{(k)}\}_{k=1}^N$ obtained by a filtering algorithm until time step T , the weights in the backward recursions can be calculated as follows:

1. At time $t = T$:

- For $k = 1, \dots, N$, initialize $\tilde{w}_{T|T}^{(k)} = \tilde{w}_T^{(k)}$.

2. At times $t = T - 1, \dots, 1$:

- For $k = 1, \dots, N$, evaluate the importance weights

$$\tilde{w}_{t|T}^{(k)} = \sum_{j=1}^N \tilde{w}_{t+1|T}^{(j)} \frac{\tilde{w}_t^{(k)} f_{\theta}(\mathbf{X}_{t+1}^{(j)} | \mathbf{X}_t^{(k)}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})}{\sum_{i=1}^N \tilde{w}_t^{(i)} f_{\theta}(\mathbf{X}_{t+1}^{(j)} | \mathbf{X}_t^{(i)}, \mathbf{Y}_{t-p:t-1}, \mathbf{Z}_{t-s:t})}$$

The reweighted particles $\{\mathbf{X}_t^{(k)}, \tilde{w}_{t|T}^{(k)}\}_{k=1}^N$ after the backward recursion approximates the smoothing distribution $p_{\theta}(\mathbf{X}_t | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T})$ as

$$\hat{p}_{\theta}(\mathbf{X}_t | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}) = \sum_{k=1}^N \tilde{w}_{t|T}^{(k)} \delta_{\mathbf{X}_t^{(k)}}(\mathbf{X}_t), \quad 1 - \tau \leq t \leq T$$

Moreover, the expected value of an arbitrary function $h(\mathbf{X}_t)$ given $\mathbf{Y}_{1-p:T}$ and $\mathbf{Z}_{1-s:T}$ is approximated by

$$\mathbb{E}_{\theta}[h(\mathbf{X}_t) | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T}] = \sum_{k=1}^N \tilde{w}_{t|T}^{(k)} h(\mathbf{X}_t^{(k)}), \quad 1 - \tau \leq t \leq T$$

SMC Forecasting

At time step T , the L -step-ahead forecasting problem is solved by extending the weighted particles $\{\mathbf{X}_{1-\tau:T}^{(k)}, \tilde{w}_T^{(k)}\}_{k=1}^N$ obtained using an SMC filtering algorithm. The SMC filtering algorithms usually do not resample at the last time step. An extra resampling step might be implemented at time step T . If so, the normalized weights of all resampled particles denoted by $\{\tilde{w}_T^{(k)}\}_{k=1}^N$ are set to $\frac{1}{N}$. For ease of presentation, the same symbol for the weighted particles is used no matter whether the resampling step is implemented at time step T . In the forecasting stage, no resampling step is needed.

The SMC forecasting algorithm proceeds as follows:

1. For $k = 1, \dots, N$, set $\tilde{w}_{x,T}^{(k)} = \tilde{w}_T^{(k)}$, where $\tilde{w}_T^{(k)}$ is derived from the filtering process, and set $\mathbf{Y}_{1-p:T}^{(k)} = \mathbf{Y}_{1-p:T}$.
2. For $t = T + 1$ to $T + L$:
 - For $k = 1, \dots, N$, sample $\mathbf{X}_t^{(k)}$ from the transition distribution $f_{\theta}(\mathbf{X}_t | \mathbf{X}_{t-r:t-1}^{(k)}, \mathbf{Y}_{t-p:t-1}^{(k)}, \mathbf{Z}_{t-s:t})$, and set $\mathbf{X}_{1-\tau:t}^{(k)} = (\mathbf{X}_{1-\tau:t-1}^{(k)}, \mathbf{X}_t^{(k)})$.
 - For $k = 1, \dots, N$, sample $\mathbf{Y}_t^{(k)}$ from the measurement distribution $g_{\theta}(\mathbf{Y}_t | \mathbf{X}_{t-r:t}^{(k)}, \mathbf{Y}_{t-p:t-1}^{(k)}, \mathbf{Z}_{t-s:t})$, and set $\mathbf{Y}_{1-p:t}^{(k)} = (\mathbf{Y}_{1-p:t-1}^{(k)}, \mathbf{Y}_t^{(k)})$.
 - For $k = 1, \dots, N$, calculate the normalized weights of the state and dependent variables:

$$\tilde{w}_{x,t}^{(k)} = \tilde{w}_{y,t}^{(k)} = \tilde{w}_{x,t-1}^{(k)}$$

With the weighted particles for the state variables, $\{\mathbf{X}_t^{(k)}, \tilde{w}_{x,t}^{(k)}\}_{t=T+1}^{T+L}$, and the weighted particles for the dependent variables, $\{\mathbf{Y}_t^{(k)}, \tilde{w}_{y,t}^{(k)}\}_{t=T+1}^{T+L}$, are obtained. The approximations of $p_{\theta}(\mathbf{X}_t | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:t})$ and

$p_{\theta}(\mathbf{Y}_t | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:T+L})$ for $t = T + 1, \dots, T + L$ are given by

$$\begin{aligned}\hat{p}_{\theta}(\mathbf{X}_t | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:t}) &= \sum_{k=1}^N \tilde{w}_{x,t}^{(k)} \delta_{\mathbf{X}_t^{(k)}}(\mathbf{X}_t) \\ \hat{p}_{\theta}(\mathbf{Y}_t | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:t}) &= \sum_{k=1}^N \tilde{w}_{y,t}^{(k)} \delta_{\mathbf{Y}_t^{(k)}}(\mathbf{Y}_t)\end{aligned}$$

Moreover, the conditional expected value of an arbitrary function $h(\mathbf{X}_t)$ or $h(\mathbf{Y}_t)$ given $\mathbf{Y}_{1-p:T}$ and $\mathbf{Z}_{1-s:t}$ is approximated by

$$\begin{aligned}\mathbb{E}_{\theta}[h(\mathbf{X}_t) | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:t}] &= \sum_{k=1}^N \tilde{w}_{x,t}^{(k)} h(\mathbf{X}_t^{(k)}), \quad t = T + 1, \dots, T + L \\ \mathbb{E}_{\theta}[h(\mathbf{Y}_t) | \mathbf{Y}_{1-p:T}, \mathbf{Z}_{1-s:t}] &= \sum_{k=1}^N \tilde{w}_{y,t}^{(k)} h(\mathbf{Y}_t^{(k)}), \quad t = T + 1, \dots, T + L\end{aligned}$$

SMC Evaluating

The marginal likelihood is defined by

$$p_{\theta}(\mathbf{Y}_{1:t} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:t}) = \int p_{\theta}(\mathbf{Y}_{1:t}, \mathbf{X}_{1-\tau:t} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:t}) d\mathbf{X}_{1-\tau:t}$$

Monte Carlo methods can directly estimate this integral, but the approximation is usually poor in practice. Alternatively, the marginal likelihood ($t > 1$) is decomposed as

$$p_{\theta}(\mathbf{Y}_{1:t} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:t}) = p_{\theta}(\mathbf{Y}_1 | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1}) \prod_{j=2}^t p_{\theta}(\mathbf{Y}_j | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j})$$

For $j = 2, \dots, t$, it is straightforward to have

$$\begin{aligned}& p_{\theta}(\mathbf{Y}_j | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) \\ &= \int p_{\theta}(\mathbf{Y}_j | \mathbf{X}_{j-r:j-1}, \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) p_{\theta}(\mathbf{X}_{j-r:j-1} | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) d\mathbf{X}_{j-r:j-1} \\ &= \int \int g_{\theta}(\mathbf{Y}_j | \mathbf{X}_{j-r:j}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) f_{\theta}(\mathbf{X}_j | \mathbf{X}_{j-r:j-1}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) \\ &\quad p_{\theta}(\mathbf{X}_{j-r:j-1} | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) d\mathbf{X}_{j-r:j-1} d\mathbf{X}_j\end{aligned}$$

These terms can be estimated by the SMC algorithms with the weighted particles at time step $j - 1$ ($j > 1$) as

$$\begin{aligned}& \hat{p}_{\theta}(\mathbf{Y}_j | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) \\ &= \sum_{k=1}^N \tilde{w}_{j-1}^{(k)} \int g_{\theta}(\mathbf{Y}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{X}_j, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) f_{\theta}(\mathbf{X}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) d\mathbf{X}_j\end{aligned}$$

SIR Algorithm

In the SIR algorithm, this approximation can be further simplified by sampling $\mathbf{X}_j^{(k)}$ ($j > 1$) from $q_{\theta}(\mathbf{X}_j | \mathbf{X}_{j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j}, \mathbf{Z}_{j-s:j})$ and setting $\mathbf{X}_{1-\tau:j}^{(k)} = (\mathbf{X}_{1-\tau:j-1}^{(A_{j-1}^{(k)})}, \mathbf{X}_j^{(k)})$ for $k = 1, \dots, N$:

$$\begin{aligned} & \hat{p}_{\theta}(\mathbf{Y}_j | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) \\ &= \sum_{k=1}^N V_{j-1}^{(k)} \frac{g_{\theta}(\mathbf{Y}_j | \mathbf{X}_{j-r:j}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) f_{\theta}(\mathbf{X}_j^{(k)} | \mathbf{X}_{j-r:j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j})}{q_{\theta}(\mathbf{X}_j | \mathbf{X}_{j-r:j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j}, \mathbf{Z}_{j-s:j})} \\ &= \sum_{k=1}^N w_j(\mathbf{X}_{1-\tau:j}^{(k)}) \end{aligned}$$

Moreover, $p_{\theta}(\mathbf{Y}_1 | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$ can be written as the following integral with the importance sampling scheme:

$$\begin{aligned} & p_{\theta}(\mathbf{Y}_1 | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1}) \\ &= \int \frac{\mu_{\theta}(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1}) g_{\theta}(\mathbf{Y}_1 | \mathbf{X}_{1-r:1}, \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})}{q_{\theta}^1(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})} q_{\theta}^1(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1}) d\mathbf{X}_{1-\tau:1} \end{aligned}$$

In the SIR algorithm, $\{\mathbf{X}_{1-\tau:1}^{(k)}\}_{k=1}^N$ are sampled from $q_{\theta}^1(\mathbf{X}_{1-\tau:1} | \mathbf{Y}_{1-p:1}, \mathbf{Z}_{1-s:1})$. Therefore, the basic Monte Carlo approximation of $p_{\theta}(\mathbf{Y}_1 | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$ is

$$\hat{p}_{\theta}(\mathbf{Y}_1 | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1}) = \frac{1}{N} \sum_{k=1}^N w_1(\mathbf{X}_{1-\tau:1}^{(k)})$$

BF Algorithm

In the BF algorithm, this approximation can be simplified by sampling $\mathbf{X}_j^{(k)}$ ($j > 1$) from $f_{\theta}(\mathbf{X}_j^{(k)} | \mathbf{X}_{j-r:j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j})$ and setting $\mathbf{X}_{1-\tau:j}^{(k)} = (\mathbf{X}_{1-\tau:j-1}^{(A_{j-1}^{(k)})}, \mathbf{X}_j^{(k)})$ for $k = 1, \dots, N$:

$$\begin{aligned} & \hat{p}_{\theta}(\mathbf{Y}_j | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) \\ &= \sum_{k=1}^N \frac{1}{N} \frac{g_{\theta}(\mathbf{Y}_j | \mathbf{X}_{j-r:j}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) f_{\theta}(\mathbf{X}_j^{(k)} | \mathbf{X}_{j-r:j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j})}{f_{\theta}(\mathbf{X}_j^{(k)} | \mathbf{X}_{j-r:j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j})} \\ &= \frac{1}{N} \sum_{k=1}^N w_j(\mathbf{X}_{1-\tau:j}^{(k)}) \end{aligned}$$

Moreover, the $\hat{p}_{\theta}(\mathbf{Y}_1 | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$ in the BF algorithm is computed in the same way as the one in the SIR algorithm.

APF Algorithm

In the APF algorithm, $v_{j-1}^{(k)} = \tilde{w}_{j-1}^{(k)} \hat{p}_{\theta}^A(\mathbf{Y}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j})$. Then the approximation can be written as

$$\begin{aligned} & \hat{p}_{\theta}(\mathbf{Y}_j | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}), \quad j > 1 \\ &= \sum_{k=1}^N \frac{v_{j-1}^{(k)} \int g_{\theta}(\mathbf{Y}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{X}_j, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) f_{\theta}(\mathbf{X}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) d\mathbf{X}_j}{\hat{p}_{\theta}^A(\mathbf{Y}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j})} \\ &= \sum_{k=1}^N \left(\frac{v_{j-1}^{(k)} \sum_{i=1}^N v_{j-1}^{(i)}}{(\sum_{i=1}^N v_{j-1}^{(i)}) \hat{p}_{\theta}^A(\mathbf{Y}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j})} \right. \\ & \quad \left. \int g_{\theta}(\mathbf{Y}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{X}_j, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) f_{\theta}(\mathbf{X}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) d\mathbf{X}_j \right) \end{aligned}$$

The auxiliary index variables $\{A_{j-1}^{(k)}\}_{k=1}^N$ are sampled according to $\left\{ \frac{v_{j-1}^{(k)}}{\sum_{i=1}^N v_{j-1}^{(i)}} \right\}_{k=1}^N$. After the sampling, the weight for each auxiliary index is $\frac{1}{N}$. The approximation can be further simplified by sampling $\mathbf{X}_j^{(k)}$ ($j > 1$) from $q_{\theta}(\mathbf{X}_j | \mathbf{X}_{j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j}, \mathbf{Z}_{j-s:j})$ and setting $\mathbf{X}_{1-\tau:j}^{(k)} = (\mathbf{X}_{1-\tau:j-1}^{(A_{j-1}^{(k)})}, \mathbf{X}_j^{(k)})$ for $k = 1, \dots, N$:

$$\begin{aligned} & \hat{p}_{\theta}(\mathbf{Y}_j | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) \\ &= \sum_{k=1}^N \frac{1}{N} \left(\sum_{i=1}^N v_{j-1}^{(i)} \right) \frac{g_{\theta}(\mathbf{Y}_j | \mathbf{X}_{j-r:j}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) f_{\theta}(\mathbf{X}_j^{(k)} | \mathbf{X}_{j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j})}{\hat{p}_{\theta}^A(\mathbf{Y}_j | \mathbf{X}_{j-r:j-1}^{(k)}, \mathbf{Y}_{j-p:j-1}, \mathbf{Z}_{j-s:j}) q_{\theta}(\mathbf{X}_j^{(k)} | \mathbf{X}_{j-r:j-1}^{(A_{j-1}^{(k)})}, \mathbf{Y}_{j-p:j}, \mathbf{Z}_{j-s:j})} \\ &= \frac{1}{N} \left(\sum_{i=1}^N v_{j-1}^{(i)} \right) \left(\sum_{k=1}^N w_j(\mathbf{X}_{1-\tau:j}^{(k)}) \right). \end{aligned}$$

It is further simplified to $\hat{p}_{\theta}(\mathbf{Y}_j | \mathbf{Y}_{1-p:j-1}, \mathbf{Z}_{1-s:j}) = \sum_{i=1}^N v_{j-1}^{(i)}$ if you use APF(ADP). Moreover, the $\hat{p}_{\theta}(\mathbf{Y}_1 | \mathbf{Y}_{1-p:0}, \mathbf{Z}_{1-s:1})$ in the APF is computed in the same way as the one in the SIR algorithm.

SMC Learning

The SMC learning algorithm of the general SSM is the same as the SMC learning algorithm of the simple SSM that is illustrated in the section “**SMC Learning**” on page 1974.

Defining a State Space Model

In order to define an SSM, you must provide parameters, variables, and multiple distribution functions (see the section “[Sequential Monte Carlo Methods and State Space Models](#)” on page 1961). The SMC procedure defines an SSM by using either of two methods: Method 1 and Method 2.

Method 1

In the SMC procedure, Method 1 defines an SSM by using the combinations of AUXMODEL, INITSTATE, MODEL, PARAMETER, PRIOR, STATE, STATEVAR, and VAR statements. The following sections define the components of an SSM by using Method 1.

Method 1: Parameters

The parameters are specified in the PARAMETER statement. Each name that you specify in the PARAMETER statement is a parameter.

- *Parameter value:* The initial parameter values are specified in the PARAMETER statement.
- *Parameter bound:* The parameter bounds, which might be needed in the parameter estimation, are specified in the PARAMETER statement and the PRIOR statements.

Method 1: Lagged Variables

The lagged state, dependent, and independent variables can appear in the *distribution-arguments* of the AUXMODEL, MODEL, INITSTATE, and STATE statements. Moreover, the lagged state variables can also appear as the *variable-name* in the INITSTATE statements. The lagged variables are constructed by concatenating the variable name, the suffix “.L” (or “.I”), and the string “i” for the order of lag i :

- $rv.Li$ or $rv.Ii$: the i th lag of the variable rv ($i \geq 1$)

Method 1: State Variables

Each variable that you specify in the STATEVAR statement is a state variable.

Method 1: Dependent Variables

Each variable that satisfies the following three conditions is a dependent variable:

- The variable is specified in the VAR statement.
- The variable appears before the tilde ($\sim$) symbol in the MODEL statement.
- The variable corresponds to one variable from the input data table.

Method 1: Independent Variables

Each variable that satisfies the following three conditions is an independent variable:

- The variable is specified in the VAR statement.
- The variable is not a dependent variable.
- The variable corresponds to one variable from the input data table.

Method 1: Maximum Lag Order of State Variables, r

Suppose the maximum lag order of the i th state variable in the *distribution-arguments* of all the MODEL statements is denoted by r_{1i} . Suppose the maximum lag order of the i th state variable in the *distribution-arguments* of all the STATE statements including the PROPOSAL= option is denoted by r_{2i} . Assuming that the number of state variables is d_x , the maximum lag order of the state variables, r , is equal to $\max(\{r_{1i}\}_{i=1}^{d_x}, \{r_{2i}\}_{i=1}^{d_x})$.

Method 1: Maximum Lag Order of State Variables to Be Generated in Initialization Distribution, τ

Suppose the maximum lag order of all the state variables for the measurement distribution is r_1 and the counterpart for the transition (or proposal transition) distribution is r_2 ; that is, $r_1 = \max(\{r_{1i}\}_{i=1}^{d_x})$ and $r_2 = \max(\{r_{2i}\}_{i=1}^{d_x})$, where d_x , r_{1i} , and r_{2i} are defined in the section “Method 1: Maximum Lag Order of State Variables, r ” on page 1986. Suppose the maximum lag order of the i th state variable that must be generated in the initialization (or proposal initialization) distribution is denoted by τ_i and $\tau_i = \max(r_{1i}, r_{2i} - 1)$. Then $\tau = \max(\{\tau_i\}_{i=1}^{d_x})$.

Method 1: Maximum Lag Order of Dependent Variables, p

The maximum lag order of the dependent variables, p , is equal to the maximum lag order of all the dependent variables in the *distribution-arguments* of all the MODEL statements and the STATE statements including the PROPOSAL= option.

Method 1: Maximum Lag Order of Independent Variables, s

The maximum lag order of the independent variables, s , is equal to the maximum lag order of all the independent variables in the *distribution-arguments* of all the MODEL statements and the STATE statements including the PROPOSAL= option.

Method 1: Number of Skipped Observations, κ

Suppose that the maximum lag order of all the dependent variables for the measurement distribution is denoted by p_1 , and the counterparts for the transition (or proposal transition) distribution and the initialization (or proposal initialization) distribution are denoted by p_2 and p_3 , respectively. Moreover, the corresponding maximum lag orders of all the independent variables for the measurement distribution, the transition (or proposal transition) distribution, and the initialization (or proposal initialization) distribution are denoted by s_1 , s_2 , and s_3 , respectively. The SMC algorithms skip some observations and begin at a certain time step in the data table such that there are enough lagged observations to be used. The number of skipped observations, κ , is equal to $\max(p_2 - 1, s_2 - 1, p_1, p_3, s_1, s_3)$. In other words, the $(\kappa + 1)$ th observation corresponds to the logical time index $_t = 1$ in the SMC algorithms.

Method 1: Distribution Functions

How to define each distribution function is summarized in [Table 27.10](#).

Table 27.10 Distribution Functions

Distribution Function	Statement	Option
Initialization	INITSTATE	
Proposal initialization	INITSTATE	PROPOSAL=
Transition	STATE	
Proposal transition	STATE	PROPOSAL=
Measurement	MODEL	
Auxiliary	AUXMODEL	
Prior	PRIOR	

Method 2

In the SMC procedure, Method 2 defines an SSM by using the options in the SUBMITMODEL statement and the user-defined program. Functions and subroutines are defined in the user-defined program that appears between the SUBMITMODEL and ENDSUBMIT statements or in the file that is specified in the FILE= option in the SUBMITMODEL statement. Moreover, functions or subroutines can also be loaded from a given CMPLIB session that stores the corresponding functions or subroutines. For example, you can use the following statements to update the CMPLIB sessions:

```
proc cas;
  action sessionProp.setSessOpt / cmplib="casuser._smc_";
  run;
quit;
```

Subsequently, the predefined functions and subroutines in the session `casuser._smc_` are loaded.

To add functions and subroutines to a CMP session, you can use the **addRoutines** action. For example, the following statements store a subroutine, `toy_SAMPPRIO`, in a table in the session `casuser._smc_`:

```
proc cas;
  sessionProp.setSessOpt / cmplib="casuser._smc_";
  fcompact.addRoutines / package = "pkg", saveTable = True,
    funcTable = {name="_smc_" caslib="CASUSER" replace=1},
    routineCode =
      ' subroutine toy_sampprio(theta[*]);
        outargs theta;
        theta[1] = 0.95;
        theta[2] = 0.43;
        theta[3] = 1;
      endsub; ';
quit;
```

When you define a function or subroutine, the name, type, and order of those arguments must follow the corresponding signature syntax and semantics. For more information about the signature syntax and semantics of each function or subroutine, see the section “[Signature Syntax and Semantics of User-Defined Functions and Subroutines](#)” on page 1950.

The subroutines and functions that are loaded from the CMPLIB session are combined with the ones defined between the SUBMITMODEL and ENDSUBMIT statements or the ones defined in the file that you specify in the FILE= option in the SUBMITMODEL statement. Moreover, the subroutines and functions defined between the SUBMITMODEL and ENDSUBMIT statements or in the file that you specify in the FILE= option in the SUBMITMODEL overwrite the identical ones stored in the CMPLIB session.

The following sections define the components of an SSM by using Method 2.

Method 2: Model Name

The model name is specified in the NAME= option in the SUBMITMODEL statement, and the name is used in the name_KEYWORD functions and subroutines defined in the user-defined program.

Method 2: Parameters

The number of parameters, n , is specified in the NPARAM= option in the SUBMITMODEL statement. The parameters appear as an array argument, theta[*], of the functions and subroutines in the user-defined program. The i th element of theta[*], theta[i], denotes the i th parameter. For example, theta[2] denotes the second parameter.

- *Parameter value:* The initial parameter values are specified in the name_SAMPPRIO subroutine.
- *Parameter bound:* The parameter bounds are specified in the name_UPPERBOUNDS and name_LOWERBOUNDS subroutines.

Method 2: Lagged Variables

The lagged state, dependent, and independent variables appear as array arguments of the functions and subroutines in the user-defined program. Specifically, the lagged state variables appear as xlag[*], the lagged dependent variables appear as ylag[*], and the lagged independent variables appear as zlag[*].

In the lagged variable array argument, variables that have a lower lag order have smaller indices. Specifically, $xlag[(j - 1) * d_x + i]$ ($i, j \geq 1$) refers to the j th order of lag for the i th state variable; $ylag[(j - 1) * d_y + i]$ ($i, j \geq 1$) refers to the j th order of lag for the i th dependent variable; and $zlag[(j - 1) * d_z + i]$ ($i, j \geq 1$) refers to the j th order of lag for the i th independent variable. For example, suppose that a model has three state variables ($d_x = 3$) and the maximum lag order of the state variables is 2 ($r = 2$). In the functions and subroutines, the three first-lag state variables are denoted by xlag[1], xlag[2], and xlag[3], respectively, and the three second-order lag state variables are denoted by xlag[4], xlag[5], and xlag[6], respectively.

Method 2: State Variables

The state variables appear in an array argument, x[*], of the functions and subroutines in the user-defined program. The i th element of x[*], x[i], denotes the i th state variable. For example, x[3] denotes the third parameter.

Method 2: Dependent Variables

The dependent variables are specified in the *dependent-variables* in the SUBMITMODEL statement. The specified dependent variables match the array argument, y[*], of the functions and subroutines in the user-defined program such that The i th element of y[*], y[i], denotes the i th dependent variable in the *dependent-variables*. For example, y[1] corresponds to the first variable in the *dependent-variables*.

Method 2: Independent Variables

The independent variables are specified in the optional *independent-variables* in the SUBMITMODEL statement. The specified independent variables match the array argument, $z[*]$, of the functions and subroutines in the user-defined program such that The i th element of $z[*]$, $z[i]$, denotes the i th variable in the *independent-variables*. For example, $z[2]$ corresponds to the second variable in the *independent-variables*.

Method 2: Maximum Lag Order of State Variables, r

The maximum lag order of the state variables, r , is specified by the maximum value in the $XLAG=(xlag-options)$ option in the SUBMITMODEL statement.

Method 2: Maximum Lag Order of State Variables to Be Generated in Initialization Distribution, τ

The maximum lag order of the state variables to be generated in the initialization (or proposal initialization) distribution, τ , is specified by the INIT= suboption in the $XLAG=(xlag-options)$ option of the SUBMITMODEL statement.

Method 2: Maximum Lag Order of Dependent Variables, p

The maximum lag order of the dependent variables, p , is specified by the maximum value in the $YLAG=(ylag-options)$ option in the SUBMITMODEL statement.

Method 2: Maximum Lag Order of Independent Variables, s

The maximum lag order of the independent variables, s , is specified by the maximum value in the $ZLAG=(zlag-options)$ option in the SUBMITMODEL statement.

Method 2: Number of Skipped Observations, κ

Suppose that the maximum lag orders of all the dependent variables for the measurement, transition (or proposal transition), initialization (or proposal initialization), and auxiliary distributions are denoted by p_1 , p_2 , p_3 , and p_4 , respectively. The values p_1 , p_2 , p_3 , and p_4 are specified, respectively, by the MEAS=, TRAN=, INIT=, and AUX= suboptions of the $YLAG=(ylag-options)$ option in the SUBMITMODEL statement. Moreover, the corresponding maximum lag orders of all the independent variables for the measurement distribution, transition (or proposal transition) distribution, initialization (or proposal initialization), and auxiliary distributions are denoted by s_1 , s_2 , s_3 , and s_4 , respectively. The values s_1 , s_2 , s_3 , and s_4 are specified, respectively, by the MEAS=, TRAN=, INIT=, and AUX= suboptions of the $ZLAG=(zlag-options)$ option in the SUBMITMODEL statement.

The SMC algorithms skip some observations and begin at a certain time step in the data table such that there are enough lagged observations to be used. The number of skipped observations, κ , is equal to $\max(p_2 - 1, s_2 - 1, p_1, p_3, p_4, s_1, s_3, s_4)$. In other words, the $(\kappa + 1)$ th observation corresponds to the time index $_t = 1$ in the SMC algorithms.

Method 2: Distribution Function

For Method 2, how to define each distribution function is summarized in Table 27.11.

Table 27.11 Method 2: Distribution Functions

Distribution Function	Function and Subroutine
Initialization	<i>name_EVALINIT</i> , <i>name_SAMPINIT</i>
Proposal initialization	<i>name_EVALINITP</i> , <i>name_SAMPINITP</i>
Transition	<i>name_EVALTRAN</i> , <i>name_SAMPTRAN</i>
Proposal transition	<i>name_EVALTRANP</i> , <i>name_SAMPTRANP</i>
Measurement	<i>name_EVALMEAS</i> , <i>name_SAMPMEAS</i>
Auxiliary	<i>name_EVALNEXTMEAS</i>
Prior	<i>name_EVALPRIO</i>

Method 2: Category State Variables

The category state variables are supported only by Method 2. For more information about the category state variables, see the section “*NSTATE*<(numeric-list)>=*number*” on page 1947.

Distribution Argument Restriction

In the AUXMODEL, MODEL, INITSTATE, and STATE statements, the *distribution-arguments* are specified by expressions that must follow any SAS programming statements. In practice, the evaluation of the *distribution-arguments* can have computation errors even though they are specified correctly. The reason why the computation errors exist is that the SMC methods are based on simulations and the state variables can have extreme values. In most cases, the errors result from two circumstances: the support restriction of a specific function and the parameter restriction of a specific distribution. For example, the following statements contain the expressions that can cause computation errors of the function support restriction:

```
state x1 ~ normal(x1.11, 1);
state x2 ~ normal(x2.11, 1);
state x3 ~ normal(log(x1), sqrt(x2));
```

where x1, x2, and x3 are state variables. The LOG and SQRT functions both require a positive argument. However, x1 or x2 is drawn from a normal distribution, so there is likely to be a negative x1 or x2 value in the simulation within the SMC method. To avoid the computation error, you can add the IFN function to the third statement as follows:

```
state x3 ~ normal(log(ifn(x1>0.01,x1,0.01)), sqrt(ifn(x2>0.01,x2,0.01)));
```

All the arguments to the IFN function are evaluated before the function is called, and the results of the evaluation are then passed to the function. The following statement can still have computation errors even though it appears to have the same logic as the previous statement:

```
state x3 ~ normal(ifn(x1>0.01,log(x1),log(0.01)), sqrt(ifn(x2>0.01,x2,0.01)));
```

The following statements contain the expressions that can cause computation errors of the parameter restriction:


```
state x1 ~ normal(x1.11, 1);
state x2 ~ lognormal(x1, 1);
state x3 ~ poisson(x1);
```

where x_1 , x_2 , and x_3 are state variables. The Poisson distribution asks for a nonnegative parameter, but a negative x_1 value can be drawn from a specific distribution. Moreover, an overflow error can occur when you draw a new sample from the lognormal distribution if the x_1 value is quite large. To avoid the computation error, you can add the IFN function as follows:

```
state x2 ~ lognormal(ifn(x1<500,x1,500), 1);
state x3 ~ poisson(ifn(x1<0,0,x1));
```

Whether the adjustment is necessary or what the format of the adjustment is must depend on the *distribution-arguments* of a specific distribution.

Standard Distributions

This section lists all distributions that PROC SMC recognizes.

Table 27.12 Cauchy Distribution

Distribution	cauchy (l, s)
Parameter	Location parameter: l ; scale parameter: s
Density	$\frac{1}{\pi} \left(\frac{s}{s^2 + (\theta - l)^2} \right)$
Parameter restriction	$s > 0$
Range	$\theta \in (-\infty, \infty)$
Mean	Not defined
Variance	Not defined
Mode	l

Table 27.13 Beta Distribution

Distribution	beta (a, b)
Density	$\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \theta^{a-1} (1-\theta)^{b-1}$
Parameter	Shape parameters: a and b
Parameter restriction	$a > 0, b > 0$
Range	$\begin{cases} [0, 1] & \text{when } a = 1, b = 1 \\ [0, 1) & \text{when } a = 1, b \neq 1 \\ (0, 1] & \text{when } a \neq 1, b = 1 \\ (0, 1) & \text{otherwise} \end{cases}$
Mean	$\frac{a}{a+b}$

Table 27.13 *continued*

Variance	$\frac{ab}{(a+b)^2(a+b+1)}$	
	$\frac{a-1}{a+b-2}$	$a > 1, b > 1$
	0 and 1	$a < 1, b < 1$
Mode	0	$\begin{cases} a < 1, b \geq 1 \\ a = 1, b > 1 \end{cases}$
	1	$\begin{cases} a \geq 1, b < 1 \\ a > 1, b = 1 \end{cases}$
	does not exist uniquely	$a = b = 1$

Table 27.14 **Gamma Distribution**

Distribution	gamma (a, b)
Parameter	Shape parameter: a ; scale parameter: b
Density	$\frac{1}{b^a \Gamma(a)} \theta^{a-1} e^{-\theta/b}$
Parameter restriction	$a > 0, b > 0$
Range	$\theta \in [0, \infty)$ if $a = 1$; $(0, \infty)$ otherwise
Mean	ab
Variance	ab^2
Mode	$(a-1)b$ if $a \geq 1$

Table 27.15 **Inverse Gamma Distribution**

Distribution	igamma (a, b)
Parameter	Shape parameter: a ; scale parameter: b
Density	$\frac{b^a}{\Gamma(a)} \theta^{-(a+1)} e^{-b/\theta}$
Parameter restriction	$a > 0, b > 0$
Range	$\theta \in (0, \infty)$
Mean	$\frac{b}{a-1}$ if $a > 1$
Variance	$\frac{b^2}{(a-1)^2(a-2)}$
Mode	$\frac{b}{a+1}$
Relationship to the gamma distribution	$\theta \sim \text{gamma}(a, 1/b) \Leftrightarrow 1/\theta \sim \text{igamma}(a, b)$

Table 27.16 Lognormal Distribution

Distribution	lognormal (m, s)
Parameter	Mean parameter: m ; standard deviation parameter: s
Density	$\frac{1}{\theta s \sqrt{2\pi}} \exp\left(-\frac{(\log \theta - m)^2}{2s^2}\right)$
Parameter restriction	$s > 0$
Range	$\theta \in (0, \infty)$
Mean	$\exp(m + s^2/2)$
Variance	$\exp(2(m + s^2)) - \exp(2m + s^2)$
Mode	$\exp(m - s^2)$

Table 27.17 Normal Distribution

Distribution	normal (m, s)
Parameter	Mean parameter: m ; standard deviation parameter: s
Density	$\frac{1}{s \sqrt{2\pi}} \exp\left(-\frac{(\theta - m)^2}{2s^2}\right)$
Parameter restriction	$s > 0$
Range	$\theta \in (-\infty, \infty)$
Mean	m
Variance	s^2
Mode	m

Table 27.18 Poisson Distribution

Distribution	poisson (λ)
Density	$\frac{\lambda^\theta}{\theta!} \exp(-\lambda)$
Parameter	Mean parameter: λ
Parameter restriction	$\lambda \geq 0$
Range	$\theta \in \begin{cases} \{0, 1, \dots\} & \text{if } \lambda > 0 \\ \{0\} & \text{if } \lambda = 0 \end{cases}$
Mean	λ
Variance	λ , if $\lambda > 0$
Mode	$\text{round}(\lambda)$

Table 27.19 *t* Distribution

Distribution	$\mathbf{t}(m, s, \nu)$
Density	$\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})s\sqrt{\nu\pi}}(1 + \frac{(\theta-m)^2}{\nu s^2})^{-\frac{\nu+1}{2}}$
Parameter	Mean parameter: m ; standard deviation parameter: s ; degrees of freedom parameter: ν
Parameter restriction	$s > 0, \nu > 0$
Range	$\theta \in (-\infty, \infty)$
Mean	m if $\nu > 1$
Variance	$\frac{\nu}{\nu-2}s^2$ if $\nu > 2$
Mode	m

Table 27.20 Uniform Distribution

Distribution	$\mathbf{uniform}(l, u)$
Density	$\frac{1}{u-l}$
Parameter	Left location parameter: l ; right location parameter: u
Parameter restriction	$u > l$
Range	$\theta \in [l, u]$
Mean	$\frac{l+u}{2}$
Variance	$\frac{(u-l)^2}{12}$
Mode	Does not exist

Table 27.21 Degenerate Distribution

Distribution	$\mathbf{degenerate}(c)$
Density	1 if $\theta = c$
Parameter	Constant value: c
Parameter restriction	Does not exist
Range	$\theta = c$
Mean	c
Variance	0
Mode	c

Missing Values

Missing values of the dependent and independent variables are treated as follows:

- The independent variables cannot have missing values.
- Missing values in the dependent variables are not supported. But at the end of the data table, for the purpose of forecasting, the dependent variables can be set to missing values when the independent variables are provided.

Time Index

There are two types of time index:

- *actual time index*
This is the time index of the observations (dependent or independent variables). The actual time index of the i th observation (dependent or independent variable) vector is i ($i \geq 1$). For example, if all observations of the same time step are stored in a row and there are 365 rows, then the actual time indices for the observations are 1, 2, ..., 365. The actual time index is used as a variable or an argument, denoted by `_time`, in the AUXMODEL, INITSTATE, MODEL, STATE, and SUBMITMODEL statements.
- *logical time index*
This is the time index used in the SMC algorithms. The first time step that is incorporated in the SMC algorithms is indexed as 1. This index is denoted by t in these algorithms. For more information about the SMC algorithms, see the sections “[SMC Evaluating](#)” on page 1972, “[SMC Filtering](#)” on page 1964, “[SMC Forecasting](#)” on page 1971, “[SMC Smoothing](#)” on page 1969, and “[SMC Algorithms for the General Case of SSMS](#)” on page 1974. The logical time index is output as `_t` in the data table that corresponds to the OUT= or OUTSIM= option in the EVALUATE, FILTER, FORECAST, and SMOOTH statements.

The actual time index and logical time index do not necessarily have the same value. For example, if the observations contain N_{ob} time steps and the first two observations must be skipped (see the sections “[Method 1: Number of Skipped Observations, \$\kappa\$](#) ” on page 1986 and “[Method 2: Number of Skipped Observations, \$\kappa\$](#) ” on page 1989), the actual time indices of all observations are 1, 2, 3, ..., N_{ob} and the logical time indices of all observations are $-1, 0, 1, \dots, N_{\text{ob}} - 2$.

Data Table Output

The SMC procedure can create the data tables that are specified in the OUT= and OUTSIM= options in the FILTER, FORECAST, and SMOOTH statements; the OUT= option in the EVALUATE statement; the OUTMODEL= option in the SCORE statement; and the OUTPOST= option in the METHOD=PMCMC(OUTPOST=) option in the LEARN statement. The information in each table is described in the following sections.

OUT= Data Table Generated from the EVALUATE Statement

The output data table contains the following variables:

Time ID Variable	value of the variable that is specified in the TIME= option in the ID statement
_t	value of the logical time index in the evaluation algorithms. For more information about the logical time index, see the section “ Time Index ” on page 1995.
LogLikelihood	log likelihood up to each time step

OUT= Data Table Generated from the FILTER Statement

The output data table contains the following variables:

Time ID Variable	value of the variable that is specified in the TIME= option in the ID statement
_t	value of the logical time index in the filtering algorithms. For more information about the logical time index, see the section “ Time Index ” on page 1995.
State ID	index of the state variable in the filtering algorithms at a given time step.
mean	weighted mean of the corresponding noncategorical state variable at a given time step in the filtering estimates. This variable does not exist if there is no noncategorical state variable.
stdDev	weighted sample standard deviation of the corresponding noncategorical state variable at a given time step in the filtering estimates. This variable does not exist if there is no noncategorical state variable.
min	minimum value of the corresponding noncategorical state variable at a given time step in the filtering estimates. This variable does not exist if there is no noncategorical state variable.
Q1	first quantile of the corresponding noncategorical state variable at a given time step in the filtering estimates. This variable does not exist if there is no noncategorical state variable.
Q2	second quantile of the corresponding noncategorical state variable at a given time step in the filtering estimates. This variable does not exist if there is no noncategorical state variable.
Q3	third quantile of the corresponding noncategorical state variable at a given time step in the filtering estimates. This variable does not exist if there is no noncategorical state variable.

max	maximum value of the corresponding noncategorical state variable at a given time step in the filtering estimates. This variable does not exist if there is no noncategorical state variable.
P _j	j th ($j = 1, \dots, J$) percentile points of the corresponding noncategorical state variable at a given time step. The percentile points are specified by the PERCENT= <i>number-list</i> suboption of the OUT= option in the FILTER statement, and J is the number of values in the <i>number-list</i> . These variables do not exist if there is no noncategorical state variable.
Cat _l	the relative frequency of the l th ($l = 1, \dots, L$) category for the corresponding category state variable at a given time step, where L is the largest value in the <i>number-list</i> suboption of the NSTATE= <i>number-list</i> option in the SUBMITMODEL statement. The relative frequency equals the probability that the category state variable is in the l th category. These variables do not exist if there is no category state variable.

OUTSIM= Data Table Generated from the FILTER Statement

The output data table contains the following variables:

Time ID Variable	value of the variable that is specified in the TIME= option in the ID statement
_t	value of the logical time index in the filtering algorithms. For more information about the logical time index, see the section “ Time Index ” on page 1995.
Particle ID	index of the particle (sample) at a given time step in the filtering algorithm
Weight	normalized weight of the corresponding particle (sample) at a given time step in the filtering algorithm
State _k	value of the k th state variable, $k = 1, \dots, d_x$, of the corresponding particle (sample), where d_x is the number of state variables that are specified in the STATEVAR statement that is specified by the NSTATE= option in the SUBMITMODEL statement or the number of state variables that are specified in the STATEVAR statement.

OUT= Data Table Generated from the SMOOTH Statement

The output data table contains the following variables:

Time ID Variable	value of the variable that is specified in the TIME= option in the ID statement
_t	value of the logical time index in the smoothing algorithms. For more information about the logical time index, see the section “ Time Index ” on page 1995.
State ID	index of the state variable in the smoothing algorithms at a given time step
mean	weighted mean of the corresponding noncategorical state variable at a given time step in the smoothing estimates. This variable does not exist if there is no noncategorical state variable.
stdDev	weighted sample standard deviation of the corresponding noncategorical state variable at a given time step in the smoothing estimates. This variable does not exist if there is no noncategorical state variable.

min	minimum value of the corresponding noncategorical state variable at a given time step in the smoothing estimates. This variable does not exist if there is no noncategorical state variable.
Q1	first quantile of the corresponding noncategorical state variable at a given time step in the smoothing estimates. This variable does not exist if there is no noncategorical state variable.
Q2	second quantile of the corresponding noncategorical state variable at a given time step in the smoothing estimates. This variable does not exist if there is no noncategorical state variable.
Q3	third quantile of the corresponding noncategorical state variable at a given time step in the smoothing estimates. This variable does not exist if there is no noncategorical state variable.
max	maximum value of the corresponding noncategorical state variable at a given time step in the smoothing estimates. This variable does not exist if there is no noncategorical state variable.
P _j	j th ($j = 1, \dots, J$) percentile points of the corresponding state variable at a given time step. The percentile points are specified by the PERCENT= <i>number-list</i> suboption of the OUT= option in the SMOOTH statement, and J is the number of values in the <i>number-list</i> . These variables do not exist if there is no noncategorical state variable.
Cat _l	the relative frequency of the l th ($l = 1, \dots, L$) category for the corresponding category state variable at a given time step, where L is the largest value in the <i>number-list</i> suboption of the NSTATE= <i>number-list</i> option in the SUBMITMODEL statement. The relative frequency equals the probability that the category state variable is in the l th category. These variables do not exist if there is no category state variable.

OUTSIM= Data Table Generated from the SMOOTH Statement

The output data table contains the following variables:

Time ID Variable	value of the variable that is specified in the TIME= option in the ID statement
_t	value of the logical time index in the smoothing algorithms. For more information about the logical time index, see the section “ Time Index ” on page 1995.
Particle ID	index of the particle (sample) in the smoothing algorithm
Weight	normalized weight of the corresponding particle (sample) at a given time step in the smoothing algorithm
State _k	value of the k th state variable, $k = 1, \dots, d_x$, of the corresponding particle (sample), where d_x is the number of state variables that are specified in the STATEVAR statement or the number of state variables that is specified by the NSTATE= option in the SUBMITMODEL statement.

OUT= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

Time ID Variable	value of the variable that is specified in the TIME= option in the ID statement for the time step before the forecasting process
_t	value of the logical time index in the forecasting algorithms. For more information about the logical time index, see the section “ Time Index ” on page 1995.
Step	value h for the h -step-ahead forecast
Variable	name of the state variable or the dependent variable
Variable ID	index of the state variable or the dependent variable
mean	weighted mean of the corresponding noncategorical state variable at a given time step in the forecasting estimates. This variable does not exist if there is no noncategorical state variable.
stdDev	weighted sample standard deviation of the corresponding noncategorical state variable at a given time step in the forecasting estimates. This variable does not exist if there is no noncategorical state variable.
min	minimum value of the corresponding noncategorical state variable at a given time step in the forecasting estimates. This variable does not exist if there is no noncategorical state variable.
Q1	first quantile of the corresponding noncategorical state variable at a given time step in the forecasting estimates. This variable does not exist if there is no noncategorical state variable.
Q2	second quantile of the corresponding noncategorical state variable at a given time step in the forecasting estimates. This variable does not exist if there is no noncategorical state variable.
Q3	third quantile of the corresponding noncategorical state variable at a given time step in the forecasting estimates. This variable does not exist if there is no noncategorical state variable.
max	maximum value of the corresponding noncategorical state variable at a given time step in the forecasting estimates. This variable does not exist if there is no noncategorical state variable.
P_j	j th ($j = 1, \dots, J$) percentile points of the corresponding noncategorical state variable at a given time step. The percentile points are specified by the PERCENT= <i>number-list</i> suboption of the OUT= option in the FORECAST statement, and J is the number of values in the <i>number-list</i> . These variables do not exist if there is no noncategorical state variable.
Cat_l	the relative frequency of the l th ($l = 1, \dots, L$) category for the corresponding category state variable, where L is the largest value in the <i>number-list</i> suboption of the NSTATE= <i>number-list</i> option in the SUBMITMODEL statement. The relative frequency equals the probability that the category state variable is in the l th category. These variables do not exist if there is no category state variable.

OUTSIM= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

Time ID Variable	value of the variable that is specified in the TIME= option in the ID statement
Step	value of h for the h -step-ahead forecast
Particle ID	index of the particle (sample) in the forecast
Variable	name of the state variable or the dependent variable
Variable ID	index of the state variable or the dependent variable
Weight	normalized weight of the corresponding particle (sample) in the forecast
Value	value of the state variable or the dependent variable in the forecast

OUTMODEL= Data Table Generated from the SCORE Statement

The output data table consists of binary large object columns to store the binary data, which should be used only by the IN= option in the SCORE statement. **NOTE:** The number of columns and the definition of columns might be changed without notice in future releases. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters because the data type of each column is binary.

OUTPOST= Data Table Generated from the METHOD=PMCMC(OUTPOST=) Option in the LEARN Statement

The output data table contains the following variables:

Chain	index of the chain for the sample in the MCMC method
Iteration	index of the iteration number for the sample in the MCMC method
Theta_ i	value for the i th parameter, $i = 1, \dots, n$, where n is the number of parameters that are specified in the PARAMETER statement or specified by the NPARM= option in the SUBMITMODEL statement.
LogLike	log likelihood of the corresponding sample
LogPost	log posterior (the sum of log likelihood and log prior) of the corresponding sample

ODS Table Names

PROC SMC assigns a name to each table that it creates. You can use these names to refer to the tables when you use the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in [Table 27.22](#).

Table 27.22 ODS Tables Produced in the SMC Procedure

ODS Table Name	Description	Suboption
ODS Tables Created by the Model Information		
InitialParmInfo	Initial parameter information	Default
NObs	Observation information	Default
StateSpaceModelInfo	Model specification information	Default
ODS Tables Created by the Suboptions of METHOD=PMCMC in the LEARN Statement		
AccRates	Metropolis acceptance rates	SAMSUM
AutoCorr	Autocorrelation statistics for each parameter	DIAG=AUTOCORR
AutoCorrByChain	Autocorrelation statistics for each parameter in each Markov chain	DIAG(BYCHAIN)=AUTOCORR
DiagSettings	Settings used for MCMC diagnostics	DIAG(SETTINGS)=
DiagSummaries	ESS and MCSE diagnostics for each parameter	Default
DiagSummariesByChain	ESS and MCSE diagnostics for each parameter in each Markov chain	DIAG(BYCHAIN)=SUMMARIES
Geweke	Geweke diagnostics for each parameter	DIAG=GEWEKE
GewekeByChain	Geweke diagnostics for each parameter in each Markov chain	DIAG(BYCHAIN)=GEWEKE
Heidelberger	Heidelberger-Welch diagnostics for each parameter	DIAG=HEIDEL
HeidelbergerByChain	Heidelberger-Welch diagnostics for each parameter in each Markov chain	DIAG(BYCHAIN)=HEIDEL
PostCorr	Posterior correlation matrix	STATS=CORR
PostCorrByChain	Posterior correlation matrix for each Markov chain	STATS(BYCHAIN)=CORR
PostCov	Posterior covariance matrix	STATS=COV
PostCovByChain	Posterior covariance matrix for each Markov chain	STATS(BYCHAIN)=COV
PostIntervals	Equal-tail and highest posterior density intervals for each parameter	STATS=INTERVALS
PostIntervalsByChain	Equal-tail and highest posterior density intervals for each parameter in each Markov chain	STATS(BYCHAIN)=INTERVALS
PostSummaries	Posterior mean, standard deviation, and percentiles for each parameter	Default

Table 27.22 *continued*

ODS Table Name	Description	Suboption
PostSummariesByChain	Posterior mean, standard deviation, and percentiles for each parameter in each Markov chain	STATS(BYCHAIN)=SUMMARIES
Raftery	Raftery-Lewis diagnostics for each parameter	DIAG=RAFTERY
RafteryByChain	Raftery-Lewis diagnostics for each parameter in each Markov chain	DIAG(BYCHAIN)=RAFTERY
SamplerSettings	Full MCMC sampler specification	SAMSET

Examples: SMC Procedure

Example 27.1: Stochastic Susceptible-Infected-Recovered Model with a Constant Transmission Rate

This example illustrates how to define a stochastic susceptible-infected-recovered (SSIR) model with a constant transmission rate in the SMC procedure and then estimate the parameters using the PMCMC method.

Consider the following SSIR model within the framework of the state space model:

- Initialization distribution:

$$ID_1 = Y_1$$

$$IU_1 \sim \text{uniform}(0.75 \frac{ID_1}{p}, 1.75 \frac{ID_1}{p})$$

$$RC_1 = 0$$

$$S_1 = N - ID_1 - IU_1$$

$$\xi_1 \sim \text{Poisson}(\frac{r}{ND} S_1 (ID_1 + IU_1))$$

- Transition distribution:

$$RC_t = RC_{t-1} + \frac{1}{D}(ID_{t-1} + IU_{t-1})$$

$$ID_t = (1 - \frac{1}{D})ID_{t-1} + Y_{t-1}$$

$$IU_t = (1 - \frac{1}{D})IU_{t-1} + \xi_{t-1} - Y_{t-1}$$

$$S_t = S_{t-1} - \xi_{t-1}$$

$$\xi_t \sim \text{Poisson}(\frac{r}{ND} S_t (ID_t + IU_t))$$

- Measurement distribution:

$$Y_t \sim \text{binomial}(\text{int}(IU_t), p)$$

Here ξ_t , a Poisson distributed instance, denotes the number of new infections at time $t + 1$; S_t denotes the number of susceptible individuals at time t ; Y_t is the number of new observed infections at time t ; RC_t is the cumulative number of recovered infections until time t ; ID_t is the number of documented active infectious individuals at time t ; and IU_t is the number of undocumented active infectious individuals at time t . Moreover, N denotes the population, r denotes the basic reproduction number, p is the diagnosis rate, D denotes the average time to removal after infection, $\frac{1}{D}$ denotes the recovery rate, and $\frac{r}{D}$ is the transmission rate. In this SSIR model, there are three parameters: r , D , and p ; and there are five state variables: ξ , S , ID , IU , and RC .

The prior specification for the parameters are given as follows:

$$\begin{aligned} r &\sim \text{uniform}(2.6, 3.3) \\ D &\sim \text{gamma}(325.5, 0.0285714286) \\ p &\sim \text{uniform}(0.1, 0.4) \end{aligned}$$

The initial values and bounds for the parameters are given by

$$\begin{aligned} r &= 2.9, \quad r_l = 2.6, \quad r_u = 3.3 \\ D &= 9.3, \quad D_l = 6.5, \quad D_u = 11 \\ p &= 0.25, \quad p_l = 0.1, \quad p_u = 0.4 \end{aligned}$$

where the bounds of r and p are identical to the supports of their prior distributions, whereas the bounds of D are different from the support of its prior distribution, which is $(0, \infty)$. To achieve this, the bounds can be specified in the `PARAMETER` statement as a supplement to the supports of the prior distributions.

The following statements simulate 89 observations based on the given stochastic volatility model. The first 75 observations are used as a training data set, and the remaining 14 observations are used as a test data set.

```
*macro variables;
%let N=20000000; *population;
%let Y_1 = 100;
%let IU_1 = 500;
%let RC_1 = 0;
%let D = 8.7;
%let r = 3.1;
%let p = 0.25;
data sir_all;
  call streaminit(1);
  t = 1;
  ID = &Y_1;
  IU = &IU_1;
  RC = &RC_1;
  S = &N - ID - IU;
  xi = RAND('POISSON', &r/(&N*&D)*S*(ID+IU));
  Y = &Y_1;
  sum = S + IU + ID + RC;
```

```

output;
do t = 2 to 89;
  RC = RC + 1/&D*(IU + ID);
  IU = (1-1/&D)*IU + xi - Y;
  ID = (1-1/&D)*ID + Y;
  S = S - xi;
  xi = RAND('POISSON', &r/(&N*&D)*S*(ID+IU));
  Y = RAND('BINOMIAL', &p, int(IU));
  sum = S + IU + ID + RC; *sum is always equal to N;
  output;
end;
run;

```

The following DATA steps load the first 75 observations of the data set `sir_all` into the data set `sir_train` and then load that data set into a data table in your session that is associated with the `mylib` libref. The DATA steps assume that your libref is named `mylib`, but you can substitute any appropriately defined libref. The last 14 observations of the data set `sir_all` are loaded into the data set `sir_test`.

```

data sir_train;
  set sir_all;
  if (t le 75) then output;
run;

data sir_test;
  set sir_all;
  if (t > 75) then output;
run;

data mylib.sir_train;
  set sir_train;
run;

```

The following statements use PROC SMC to define the preceding SSIR model and then use the PMCMC method to estimate the three parameters for the simulated data set. The posterior medians will be used as the parameters in the smoothing and forecasting process.

```

proc smc data=mylib.sir_train seed = 1234 nthreads = 32;
  id time = t;
  var y;
  statevar S ID IU xi RC;
  smooth nparticle=20480 parm=median alg=sir method=fixedlag(16)
    out(pct=2.5,97.5)=mylib.sm;
  forecast nparticle=20480 parm=median alg=sir lead=14
    out(pct=2.5,97.5)=mylib.fc;
  parm r=2.9, D=9.3, p=0.3 /lower=(2.6 6.5 0.1) upper=(3.3 11 0.4);

  prior r ~ uniform(2.6, 3.3);
  prior D ~ gamma(325.5, 0.0285714286);
  prior p ~ uniform(0.1, 0.4);

  initstate ID ~ degenerate(100);
  initstate IU ~ uniform(0.85*ID/p, 2*ID/p);
  initstate RC ~ degenerate(0);
  initstate S ~ degenerate(&N - ID - IU);

```

```

initstate xi ~ poisson(r/(&N*D)*S*(ID+IU));

state RC ~ degenerate(RC.l1 + 1/D*(ID.l1+IU.l1));
state ID ~ degenerate((1-1/D)*ID.l1 + Y.l1);
state IU ~ degenerate((1-1/D)*IU.l1 + xi.l1 - Y.l1);
state S ~ degenerate(S.l1 - xi.l1);
state xi ~ poisson(r/(&N*D)*S*(ID+IU));

model y~normal(p*IU,ifn(IU>1,2*sqrt(p*(1-p)*IU),2*sqrt(p*(1-p))));
learn nparticle=256 algorithm=sir method=PMCMC(nsampl=25000
  nchain=8 nbi=5000 thin=5 seed=1
  outpost=mylib.out samset samsum
  sampler=rwm(ntu=250)
  statistics=all diagnostics=all);
run;

```

The summary tables of the model information are shown in [Output 27.1.1](#).

Output 27.1.1 Model Information Summary

The SMC Procedure

State Space Model Information	
Number of Parameters	3
Number of State Variables	5
Number of Dependent Variables	1
Number of Independent Variables	0
Seed Value	1234

Number of Observations	75
Number of Missing Observations	0

Initial Parameters Information			
Parameter	Initial Value	Lower Bound	Upper Bound
r	2.90000	2.60000	3.30000
D	9.30000	6.50000	11.00000
p	0.30000	0.10000	0.40000

The summary tables of the posterior samples are shown in [Output 27.1.2](#).

Output 27.1.2 Posterior Summary

MCMC Sampler Settings	
MCMC Sampling Method	RWM
Random Seed	1
Transformed Sampling	False
Number of Chains	8
Number of Sampling Iterations	25000
Number of Burn-in Iterations	5000
Thinning Interval	5
Number of Tuning Iterations per Tuning Stage	250
Number of Tuning Stages	24
Target Metropolis Acceptance Rate	0.234
Initial RWM Proposal Scale	2.38
Tuning Weight	0.75
Acceptance Tolerance	0.075

The correlation and covariance matrices of the parameter samples that are generated by the PMCMC method are shown in [Output 27.1.3](#).

Output 27.1.3 Correlation and Covariance Matrices

Posterior Covariance Matrix			
Parameter	r	D	p
r	0.003768	0.014348	-0.00053
D	0.014348	0.05466	-0.00201
p	-0.00053	-0.00201	0.000074

The following statements load the smoothing estimates for each state variable into the Work library and then sort each data set by the time ID:

```
data smo_1; set mylib.sm(where=(VariableID=1 and TimeID>=1)); run;
proc sort data=smo_1; by timeId; run;

data smo_2; set mylib.sm(where=(VariableID=2 and TimeID>=1)); run;
proc sort data=smo_2; by timeId; run;

data smo_3; set mylib.sm(where=(VariableID=3 and TimeID>=1)); run;
proc sort data=smo_3; by timeId; run;

data smo_4; set mylib.sm(where=(VariableID=4 and TimeID>=1)); run;
proc sort data=smo_4; by timeId; run;

data smo_5; set mylib.sm(where=(VariableID=5 and TimeID>=1)); run;
proc sort data=smo_5; by timeId; run;
```

The following statements create a relevant data table, `combine_S`, in the Work library to record the smoothing estimates and actual values of the number of susceptible individuals. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.1.4](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates.

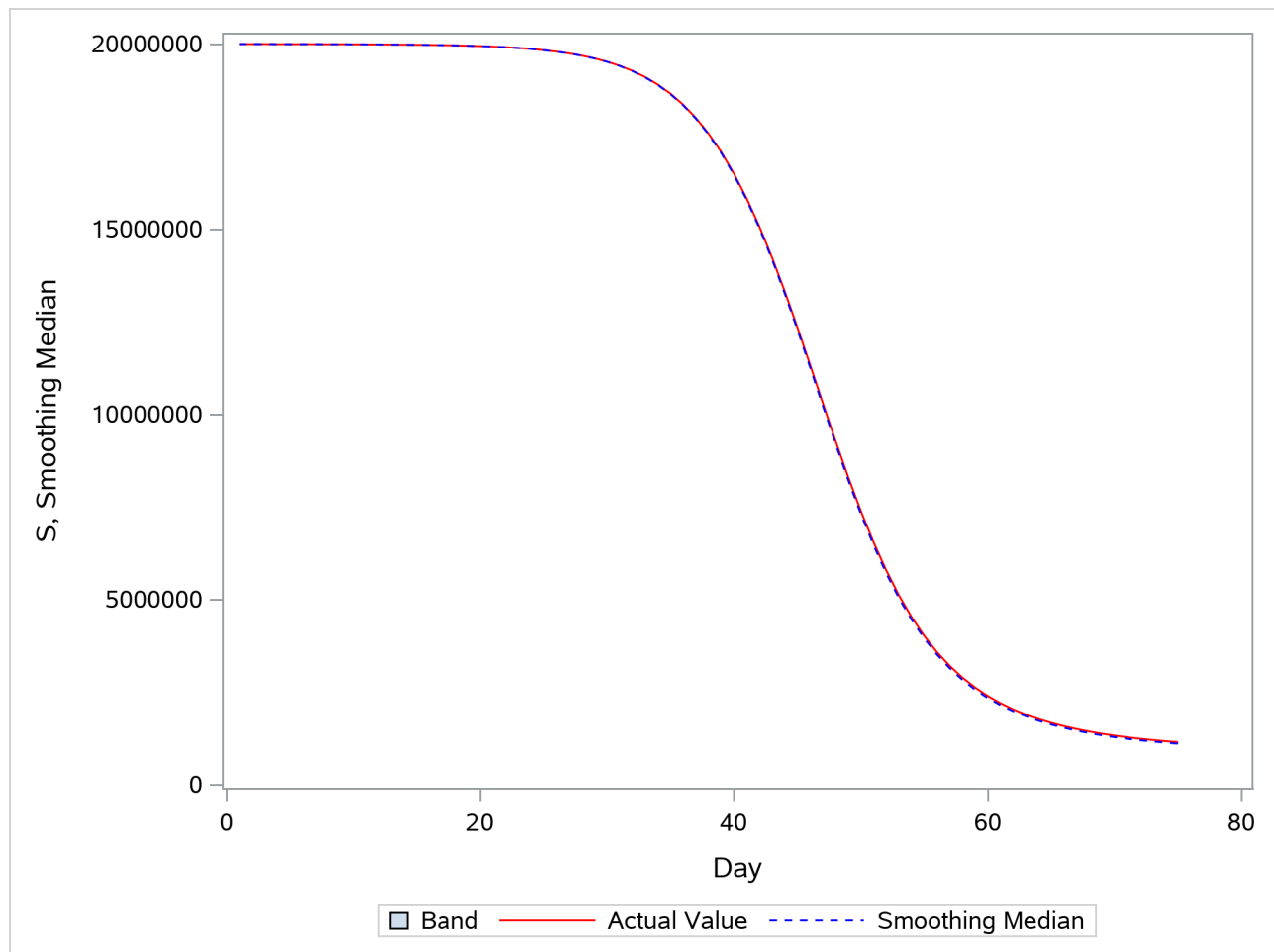

```

data combine_S;
  set smo_1 (keep=P_1 P_2 Q2);
  set sir_train (keep=t S);
run;

proc sgplot data=combine_S;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  series x=t y=S /legendlabel='Actual Value' lineattrs=(color=red);
  series x=t y=Q2 /legendlabel='Smoothing Median'
    lineattrs=(color=blue pattern=shortdash);
  yaxis label='S, Smoothing Median';
  xaxis label='Day';
run;

```

Output 27.1.4 Smoothing Estimates vs. Actual Values, S



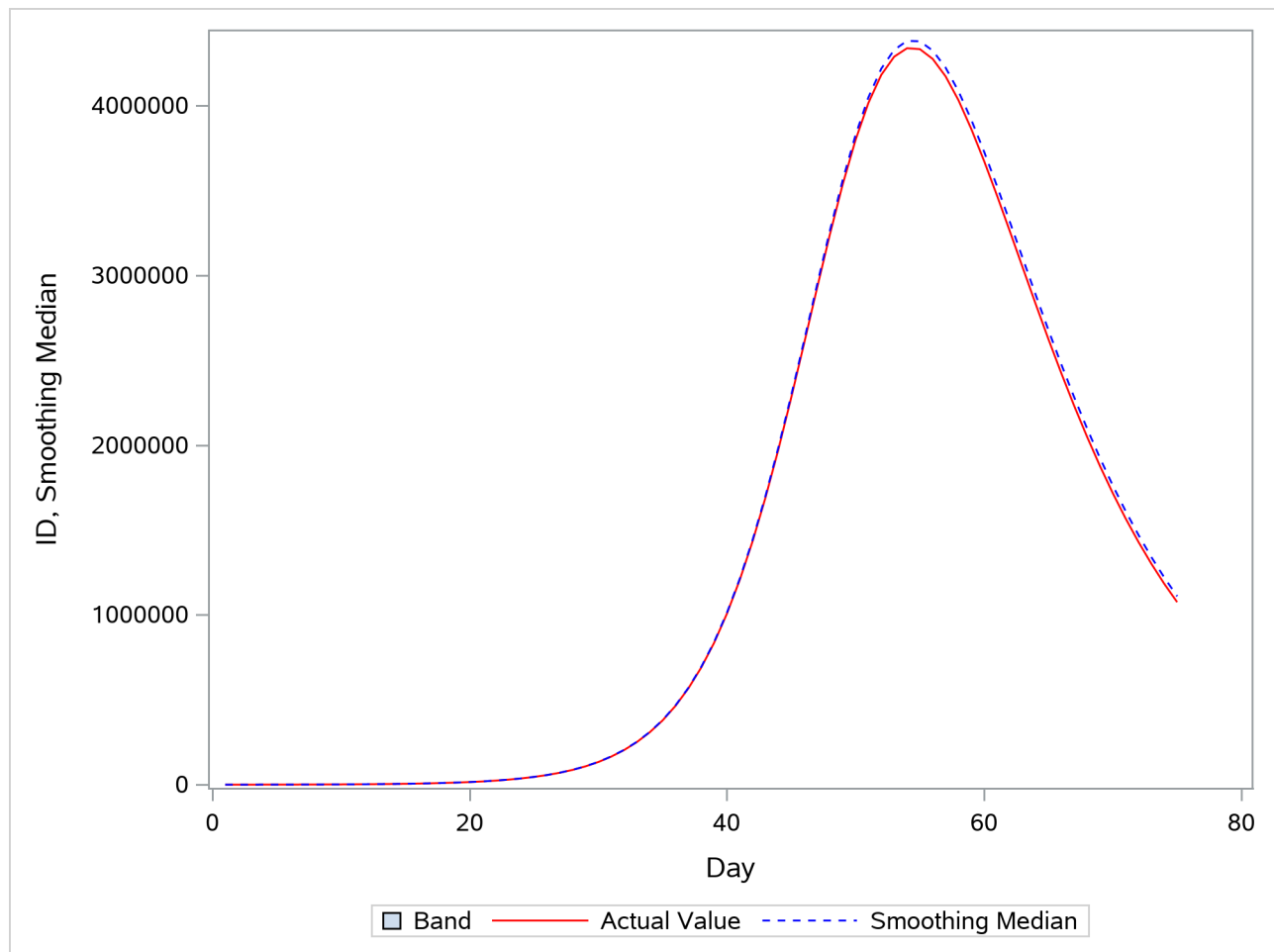
The following statements create a relevant data table, `combine_ID`, in the Work library to record the smoothing estimates and actual values of the number of documented active infectious individuals. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.1.5](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates.

```

data combine_ID;
  set smo_2 (keep=P_1 P_2 Q2);
  set sir_train (keep=t ID);
run;

proc sgplot data=combine_ID;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  series x=t y=ID /legendlabel='Actual Value' lineattrs=(color=red);
  series x=t y=Q2 /legendlabel='Smoothing Median'
    lineattrs=(color=blue pattern=shortdash);
  yaxis label='ID, Smoothing Median';
  xaxis label='Day';
run;

```

Output 27.1.5 Smoothing Estimates vs. Actual Values, ID

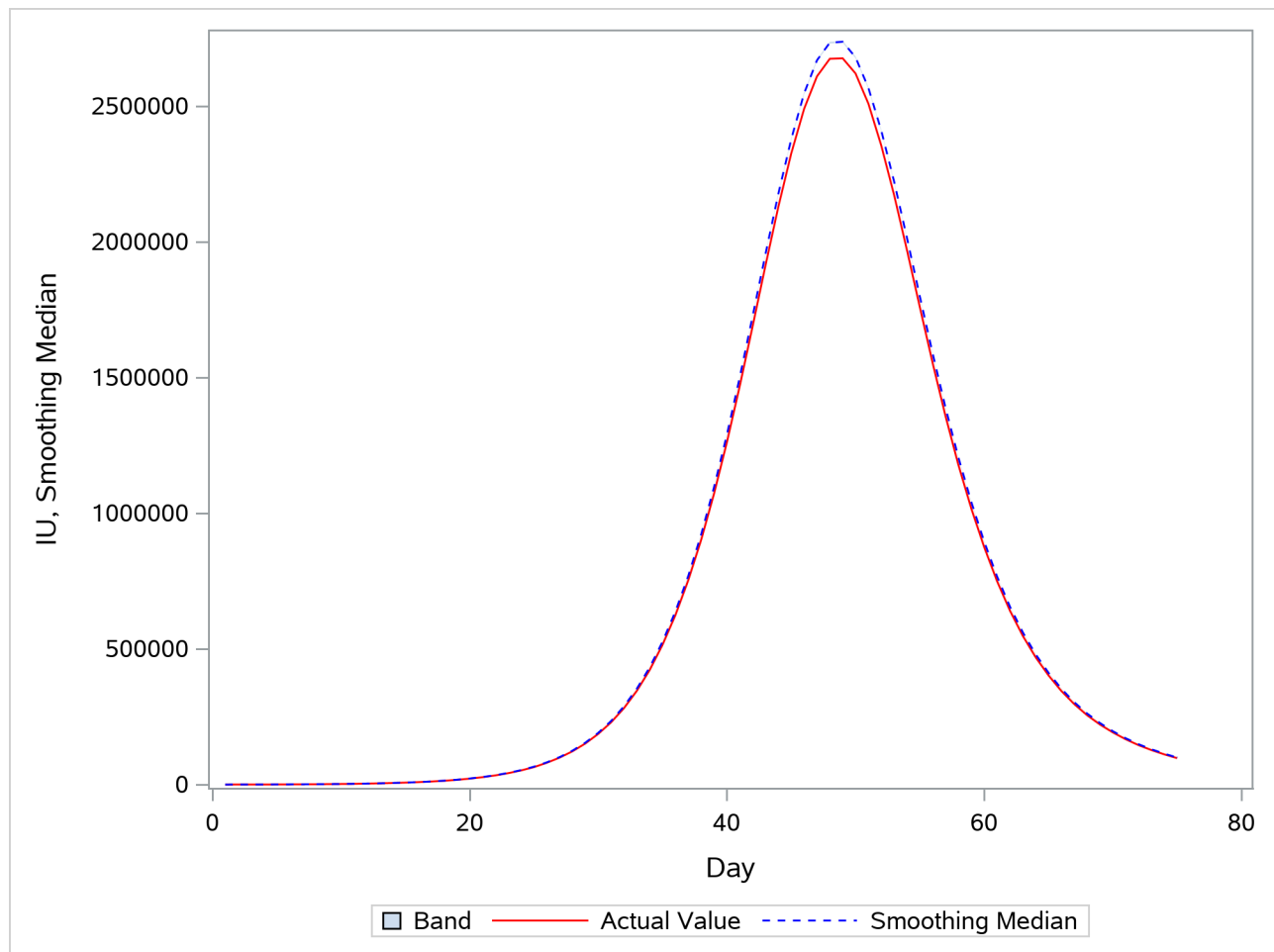
The following statements create a relevant data table, `combine_IU`, in the `Work` library to record the smoothing estimates and actual values of the number of undocumented active infectious individuals. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.1.6](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates.

```

data combine_IU;
  set smo_3 (keep=P_1 P_2 Q2);
  set sir_train (keep=t IU);
run;

proc sgplot data=combine_IU;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  series x=t y=IU /legendlabel='Actual Value' lineattrs=(color=red);
  series x=t y=Q2 /legendlabel='Smoothing Median'
    lineattrs=(color=blue pattern=shortdash);
  yaxis label='IU, Smoothing Median';
  xaxis label='Day';
run;

```

Output 27.1.6 Smoothing Estimates vs. Actual Values, IU

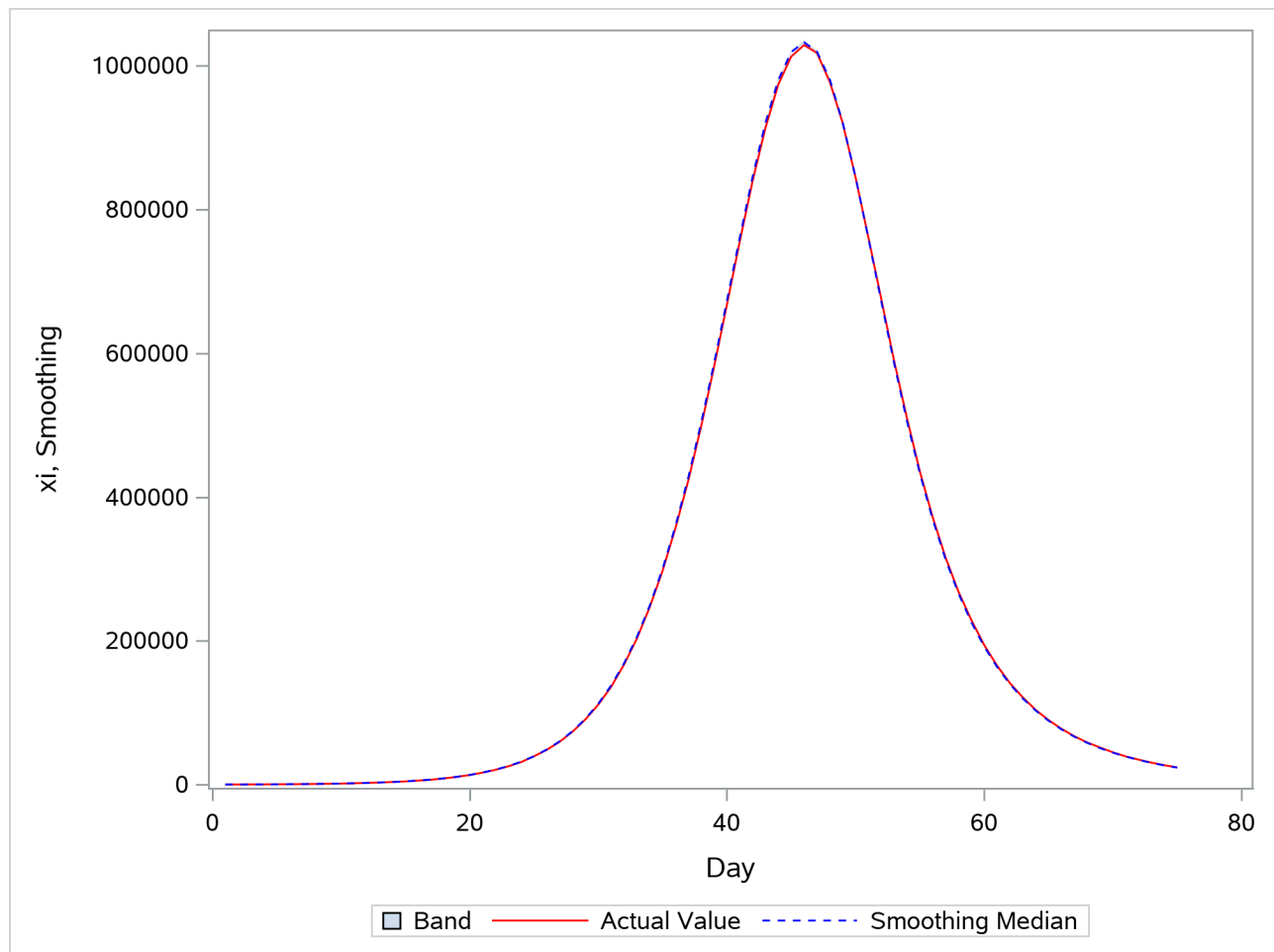
The following statements create a relevant data table, `combine_xi`, in the Work library to record the smoothing estimates and actual values of the number of new infectious individuals the next day. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.1.7](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates.

```

data combine_xi;
  set smo_4 (keep=P_1 P_2 Q2);
  set sir_train (keep=t xi);
run;

proc sgplot data=combine_xi;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  series x=t y=xi /legendlabel='Actual Value' lineattrs=(color=red);
  series x=t y=Q2 /legendlabel='Smoothing Median'
    lineattrs=(color=blue pattern=shortdash);
  yaxis label='xi, Smoothing';
  xaxis label='Day';
run;

```

Output 27.1.7 Smoothing Estimates vs. Actual Values, xi

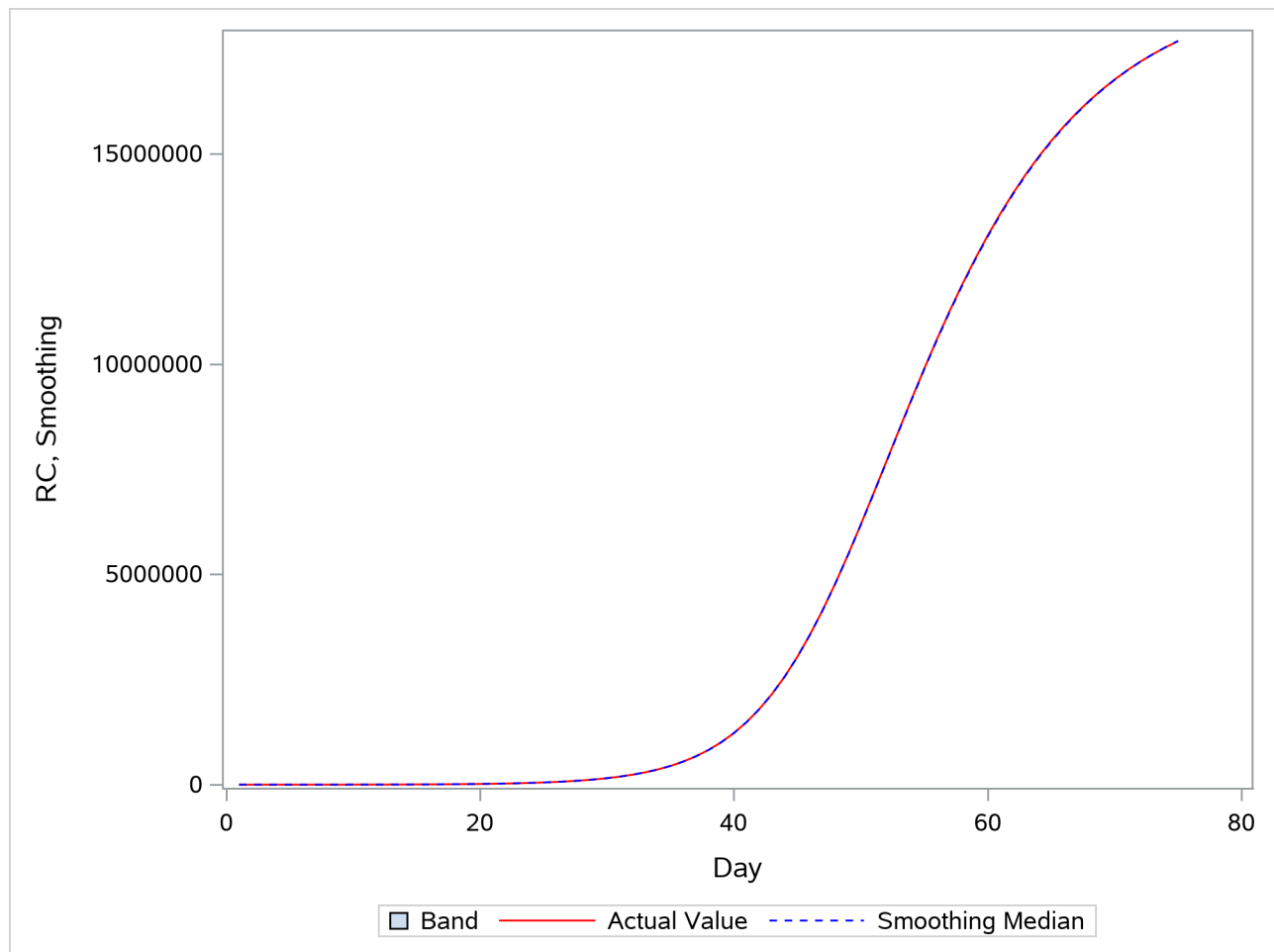
The following statements create a relevant data table, `combine_RC`, in the `Work` library to record the smoothing estimates and actual values of the cumulative number of recovered individuals. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.1.8](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates.

```

data combine_RC;
  set smo_5 (keep=P_1 P_2 Q2);
  set sir_train (keep=t RC);
run;

proc sgplot data=combine_RC;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  series x=t y=RC /legendlabel='Actual Value' lineattrs=(color=red);
  series x=t y=Q2 /legendlabel='Smoothing Median'
    lineattrs=(color=blue pattern=shortdash);
  yaxis label='RC, Smoothing';
  xaxis label='Day';
run;

```

Output 27.1.8 Smoothing Estimates vs. Actual Values, RC

The median smoothing estimates of all the state variables are close to the actual values of the simulated data set. The following statements load the forecasting result for each state variable into the Work library and then sort each data set by the time ID:

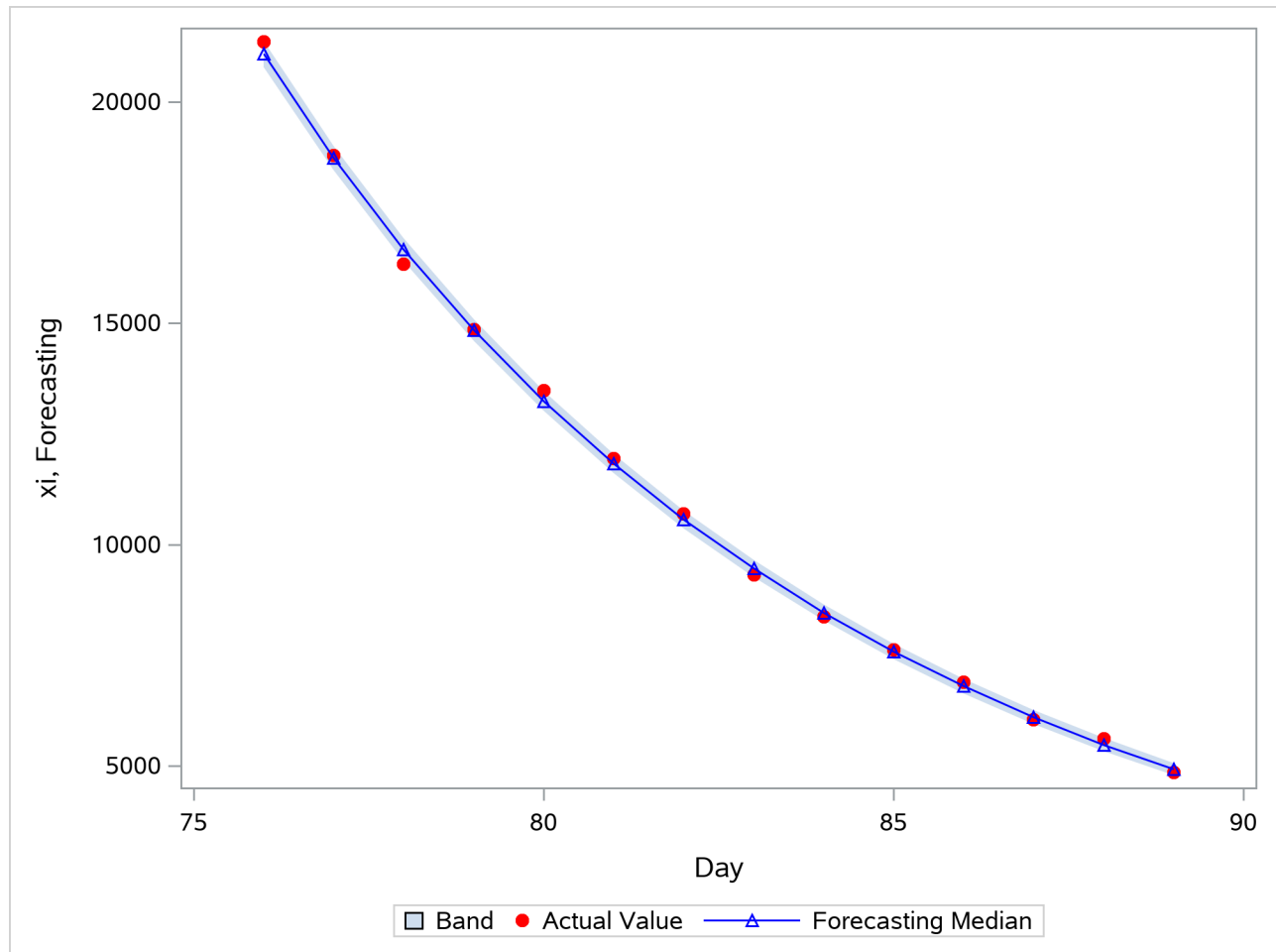
```
data fc_4; set mylib.fc(where=(VariableID=4)); run;
proc sort data=fc_4; by _t; run;

data fc_6; set mylib.fc(where=(VariableID=6)); run;
proc sort data=fc_6; by _t; run;
```

The following statements create a relevant data table, `combine_xi_forecast`, in the `Work` library to record the forecasting estimates and actual values of the number of new infectious individuals the next day. Then they draw a plot to compare the forecasting estimates to the actual values at each time step. The plot is shown in [Output 27.1.9](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the forecasting estimates.

```
data combine_xi_forecast;
  set fc_4 (keep=P_1 P_2 Q2);
  set sir_test (keep=t xi);
run;

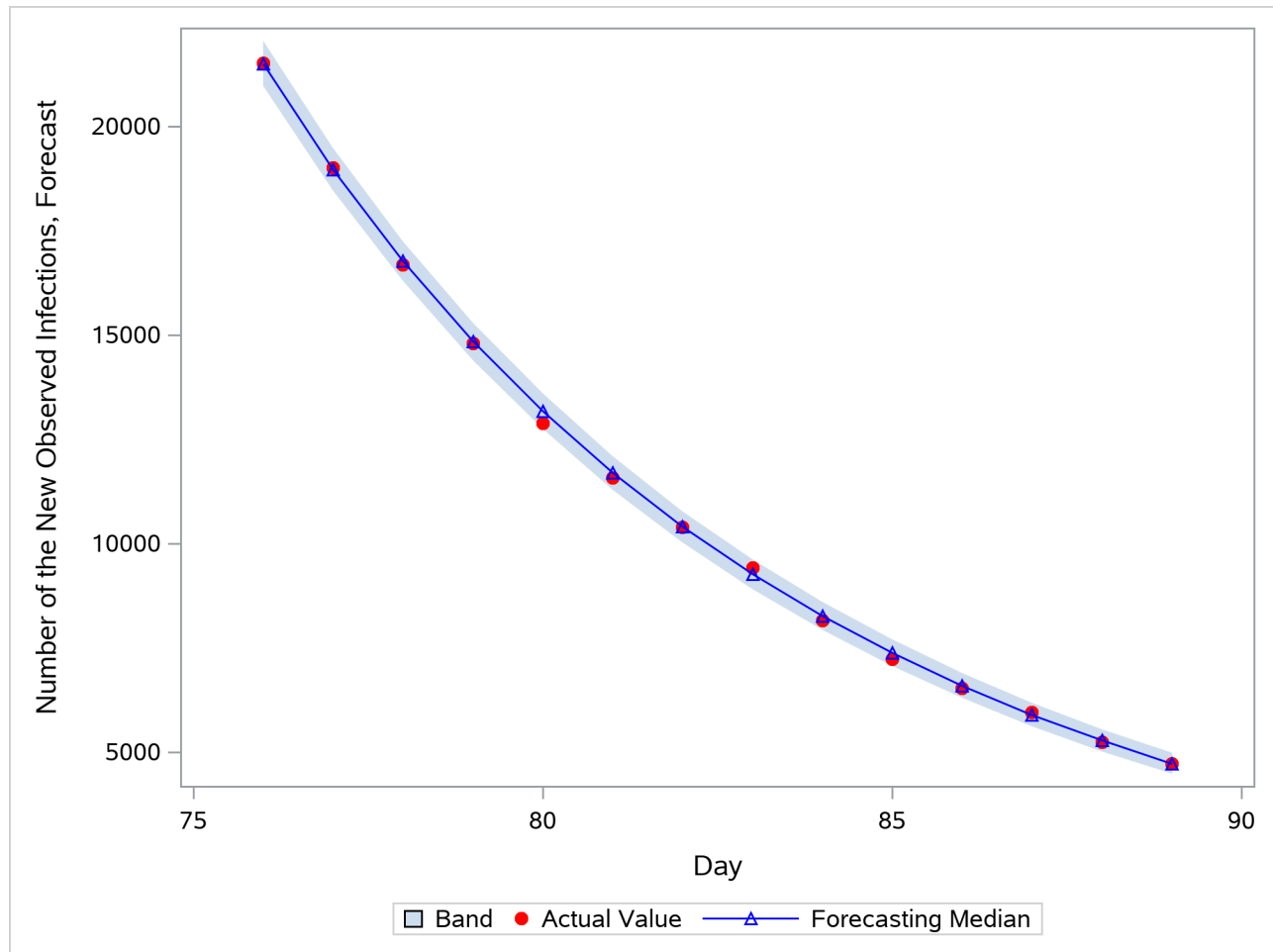
proc sgplot data=combine_xi_forecast;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  scatter x=t y=xi /legendlabel='Actual Value'
    markerattrs=(color=red size=6 symbol=circlefilled);
  series x=t y=Q2 /legendlabel='Forecasting Median'
    markers markerattrs=(color=blue size=6 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  yaxis label='xi, Forecasting';
  xaxis integer label='Day';
run;
```

Output 27.1.9 Forecasting Estimates vs. Actual Values, xi

The following statements create a relevant data table, `combine_Y_forecast`, in the `Work` library to record the forecasting estimates and actual values of the new confirmed cases. Then they draw a plot to compare the forecasting estimates to the actual values at each time step. The plot is shown in [Output 27.1.10](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the forecasting estimates.

```
data combine_Y_forecast;
    set fc_6 (keep=P_1 P_2 Q2);
    set sir_test (keep=t Y);
run;

proc sgplot data=combine_Y_forecast;
    band x=t upper=P_2 lower=P_1 / transparency=.3;
    scatter x=t y=Y / legendlabel='Actual Value'
        markerattrs=(color=red size=6 symbol=circlefilled);
    series x=t y=Q2 / legendlabel='Forecasting Median'
        markers markerattrs=(color=blue size=6 symbol=triangle)
        lineattrs=(color=blue pattern=solid);
    yaxis label='Number of the New Observed Infections, Forecast';
    xaxis integer label='Day';
run;
```

Output 27.1.10 Forecasting Estimates vs. Actual Values, New Case

Example 27.2: Stochastic Susceptible-Infected-Recovered Model with a Time-Varying Transmission Rate

Although the stochastic susceptible-infected-recovered (SSIR) model with the constant transmission rate, as shown in the section “[Example 27.1: Stochastic Susceptible-Infected-Recovered Model with a Constant Transmission Rate](#)” on page 2002, is simple and easy to understand, it is not realistic because the transition rate is often time-varying as a result of the intervention, seasonality, and other complex reasons. This example illustrates how to define an SSIR model with a time-varying transmission rate in the SMC procedure and then estimate the parameters using the PMCMC method.

Consider the following SSIR model within the framework of the state space model:

- Initialization distribution:

$$\begin{aligned}
 ID_1 &= Y_1 \\
 IU_1 &\sim \text{uniform}(0.75 \frac{ID_1}{p}, 1.75 \frac{ID_1}{p}) \\
 RC_1 &= 0 \\
 S_1 &= N - ID_1 - IU_1 \\
 \ln(\beta_1) &\sim \text{uniform}(\ln(\frac{2.75}{D}), \ln(\frac{3.3}{D})) \\
 R_{e,1} &= \exp(\ln(\beta_1)) \frac{S_1 D}{N} \\
 \xi_1 &= \exp(\ln(\beta_1)) S_1 \frac{IU_1 + ID_1}{N}
 \end{aligned}$$

- Transition distribution:

$$\begin{aligned}
 S_t &= S_{t-1} - \xi_{t-1} \\
 RC_t &= RC_{t-1} + \frac{1}{D} (ID_{t-1} + IU_{t-1}) \\
 ID_t &= (1 - \frac{1}{D}) ID_{t-1} + Y_{t-1} \\
 IU_t &= (1 - \frac{1}{D}) IU_{t-1} + \xi_{t-1} - Y_{t-1} \\
 \ln(\beta_t) &\sim \text{normal}(u_1 + u_2 t + \rho \ln(\beta_{t-1}) - u_1 - u_2(t-1), \sqrt{\sigma^2(1 - \rho^2)}) \\
 R_{e,t} &= \exp(\ln(\beta_t)) \frac{S_t D}{N} \\
 \xi_t &= \exp(\ln(\beta_t)) S_t \frac{IU_t + ID_t}{N}
 \end{aligned}$$

- Measurement distribution:

$$Y_t \sim \text{binomial}(\text{int}(IU_t), p)$$

Here ξ_t , a Poisson distributed instance, denotes the number of new infections at time $t + 1$; S_t denotes the number of susceptible individuals at time t ; Y_t is the number of new observed infections at time t ; RC_t is the cumulative number of recovered infections until time t ; ID_t is the number of documented active infectious individuals at time t ; IU_t is the number of undocumented active infectious individuals at time t ; β_t is the transmission rate at time t ; and $R_{e,t}$ is the effective reproduction number at time t . Moreover, N denotes the population; p is the diagnosis rate; D denotes the average time to removal after infection; and $\frac{1}{D}$ denotes the recovery rate. The natural logarithm of the transmission rate follows an AR(1) model with coefficients u_1 and u_2 , correlation parameter ρ , and variance $\sigma^2(1 - \rho^2)$.

In this SSIR model, there are six parameters: D , u_1 , u_2 , p , ρ , and σ ; and there are seven state variables: S , RC , ID , IU , $\ln(\beta)$, R_e , and ξ . Actually, the model can be represented without R_e because no state variables or dependent variables depend on it. However, R_e is worth investigating: $R_e < 1$ indicates containment of the disease. Moreover, the equation of ξ_t can be included in the equations of other state variables. For example, $S_t = S_{t-1} - \xi_{t-1}$ can be written as follows:

$$S_t = S_{t-1} - \exp(\ln(\beta_{t-1})) S_{t-1} \frac{IU_{t-1} + ID_{t-1}}{N}$$

The equation $IU_t = (1 - \frac{1}{D})IU_{t-1} + \xi_{t-1} - Y_{t-1}$ can be written in the same way. The reason that ξ is still kept as a state variable is because the estimated number or interval of new infections is also useful for public health organizations in a pandemic.

A scenario with a basic reproduction number that is a decreasing function of the time is considered in the data generating process. The function is defined as

$$R_{0,t} = R_{0,0} \exp(-\eta t) + (1 - \exp(-\eta t))\bar{R}_0$$

where $R_{0,t}$ denotes the basic reproduction number at time t , η denotes the decreasing rate, and $\bar{R}_0$ denotes the long run value to which $R_{0,t}$ converges when t is large. Note that the preceding function form is unknown to the modeler; the AR(1) process for the natural logarithm of the transmission rate is used to approximate the true unknown function form of $R_{0,t}$.

The following statements simulate 82 observations in which the first 75 observations are used as a training data set and the remaining 7 observations are used as a test data set:

```
*macro variables;
%let N=20000000; * 20 million people ;
%let ID_0 = 100;
%let IU_0 = 800;
%let RC_0 = 0;
%let D = 8.7;
%let R_0 = 3.1;
%let eta = 0.045;
%let p = 0.25;
data sir_data;
  t = 1;
  S = &N - &ID_0 - &IU_0;
  RC = &RC_0;
  IU = &IU_0;
  ID = &ID_0;
  R = &R_0;
  beta = R*1/&D;
  logbeta = log(beta);
  R_eff = R*S/&N;
  Y = ID;
  xi = beta*S*(IU + ID)/&N;
  sum = S + IU + ID + RC;
  output;
  call streaminit(1);
  do t = 2 to 82;
    Slag = S;
    S = S - xi;
    RC = RC + 1/&D*(IU + ID);
    IU = (1-1/&D)*IU + xi - Y;
    ID = (1-1/&D)*ID + Y;
    Y = RAND('BINomial', &p, int(IU));
    R = &R_0*exp(-&eta*t) + 1-exp(-&eta*t);
    beta = R*1/&D;
    logbeta = log(beta);
    R_eff = R*S/&N;
    xi = beta*S*(IU + ID)/&N;
```

```

        sum = S + IU + ID + RC;
    output;
end;
run;

```

The following DATA steps load the first 75 observations of the data set `sir_data` into the data set `sir_train` and then load that data set into a data table in your session that is associated with the `mylib` libref. The DATA steps assume that your libref is named `mylib`, but you can substitute any appropriately defined libref. The last 7 observations are loaded into the data set `sir_test`.

```

data sir_train;
    set sir_data;
    if (t le 75) then output;
run;

data sir_test;
    set sir_data;
    if (t > 75) then output;
run;

data mylib.sir_train;
    set sir_train;
run;

```

The following statements use PROC SMC to define the preceding SSIR model and then use the PMCMC method to estimate the three parameters for the simulated data set. The posterior medians will be used as the parameters in the smoothing and forecasting process.

```

proc smc data=mylib.sir_train seed=1234 nthreads=32;
    id time = t;
    var Y;
    statevar S IU ID logBeta RC R_EFF xi;
    smooth nparticle=20480 alg=sir parm=median
        method = fixedlag(20) out(pct=2.5, 97.5)=mylib.sm;
    forecast nparticle=20480 alg=sir lead=7 parm=median
        out(pct=2.5, 97.5)=mylib.fc;
    parm D = 9.3, u1=-1.31, u2=0, p=0.3, rho=0.8, sigma_var=0.16;

    prior D ~ gamma(325.5, 0.0285714286);
    prior sigma_var ~ igamma(11,1);
    prior rho ~ beta(4,1);
    prior p ~ uniform(0.15,0.35);
    prior u1 ~ normal(-1.31, 0.3);
    prior u2 ~ normal(0,1);

    initstate ID ~ degenerate(100);
    initstate IU ~ uniform(0.5*100/p, 1.5*100/p);
    initstate RC ~ degenerate(0);
    initstate S ~ degenerate(&N-ID-IU);
    initstate logBeta ~ uniform(log(2.75*1/D), log(3.3*1/D));
    initstate R_EFF ~ degenerate(exp(logBeta)*S*D/&N);
    initstate xi ~ degenerate(exp(logBeta)*S*(IU + ID)/&N);

    state S ~ degenerate(S.11 - xi.11);

```

```

state RC ~ degenerate(RC.l1 + 1/D*(IU.l1+ID.l1));
state IU ~ degenerate((1-1/D)*IU.l1 + xi.l1 - Y.l1);
state ID ~ degenerate((1-1/D)*ID.l1 + Y.l1);
state logBeta ~ normal(u1 + u2*_time + rho*(logBeta.l1-u1-u2*(_time-1)),
  sqrt(sigma_var*(1-rho**2)));
state R_EFF ~ degenerate(exp(logBeta)*S*D/&N);
state xi ~ degenerate(exp(logBeta)*S*(IU + ID)/&N);

model Y ~ normal(p*IU, ifn(IU>1, 2*sqrt(p*(1-p)*IU), 2*sqrt(p*(1-p))));

learn nparticle=1600 algorithm=sir method=PMCMC(nsample=20000
  nchain=8 nbi=5000 thin=5 seed=1
  outpost=mylib.out samset samsum
  sampler=rwm(ntu=200)
  statistics=all diagnostics=all);
run;

```

The summary tables of the model information are shown in [Output 27.2.1](#).

Output 27.2.1 Model Information Summary

The SMC Procedure

State Space Model Information			
Number of Parameters	6		
Number of State Variables	7		
Number of Dependent Variables	1		
Number of Independent Variables	0		
Seed Value	1234		

Number of Observations	75		
Number of Missing Observations	0		

Initial Parameters Information			
Parameter	Initial Value	Lower Bound	Upper Bound
D	9.30000	0	Infty
u1	-1.31000	-Infty	Infty
u2	0	-Infty	Infty
p	0.30000	0.15000	0.35000
rho	0.80000	0	1.00000
sigma_var	0.16000	0	Infty

The summary tables of the posterior samples are shown in [Output 27.2.2](#).

Output 27.2.2 Posterior Summary

MCMC Sampler Settings	
MCMC Sampling Method	RWM
Random Seed	1
Transformed Sampling	False
Number of Chains	8
Number of Sampling Iterations	20000
Number of Burn-in Iterations	5000
Thinning Interval	5
Number of Tuning Iterations per Tuning Stage	200
Number of Tuning Stages	24
Target Metropolis Acceptance Rate	0.234
Initial RWM Proposal Scale	2.38
Tuning Weight	0.75
Acceptance Tolerance	0.075

The correlation and covariance matrices of the parameter samples that are generated by the PMCMC method are shown in [Output 27.2.3](#).

Output 27.2.3 Correlation and Covariance Matrices

Posterior Covariance Matrix						
Parameter	D	u1	u2	p	rho	sigma_var
D	0.128537	0.000356	-0.00009	-0.00059	0.000032	0.000323
u1	0.000356	0.07613	-0.00037	-0.00006	0.000053	0.000288
u2	-0.00009	-0.00037	0.000011	4.54E-7	-2.2E-6	-0.00001
p	-0.00059	-0.00006	4.54E-7	0.00008	6.981E-8	5.31E-6
rho	0.000032	0.000053	-2.2E-6	6.981E-8	0.00001	0.000046
sigma_var	0.000323	0.000288	-0.00001	5.31E-6	0.000046	0.00086

The following statements load the smoothing estimates for $\ln(\beta)$ and R_e into the Work library and then sort each data set by the time ID:

```
data smo_s4; set mylib.sm(where=(VariableID=4 and TimeID>=1)); run;
proc sort data=smo_s4; by timeId; run;
```

```
data smo_s6; set mylib.sm(where=(VariableID=6 and TimeID>=1)); run;
proc sort data=smo_s6; by timeId; run;
```

The following statements create a relevant data table, `combine_logbeta`, in the Work library to record the smoothing estimates and actual values of $\ln(\beta)$. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.2.4](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates.

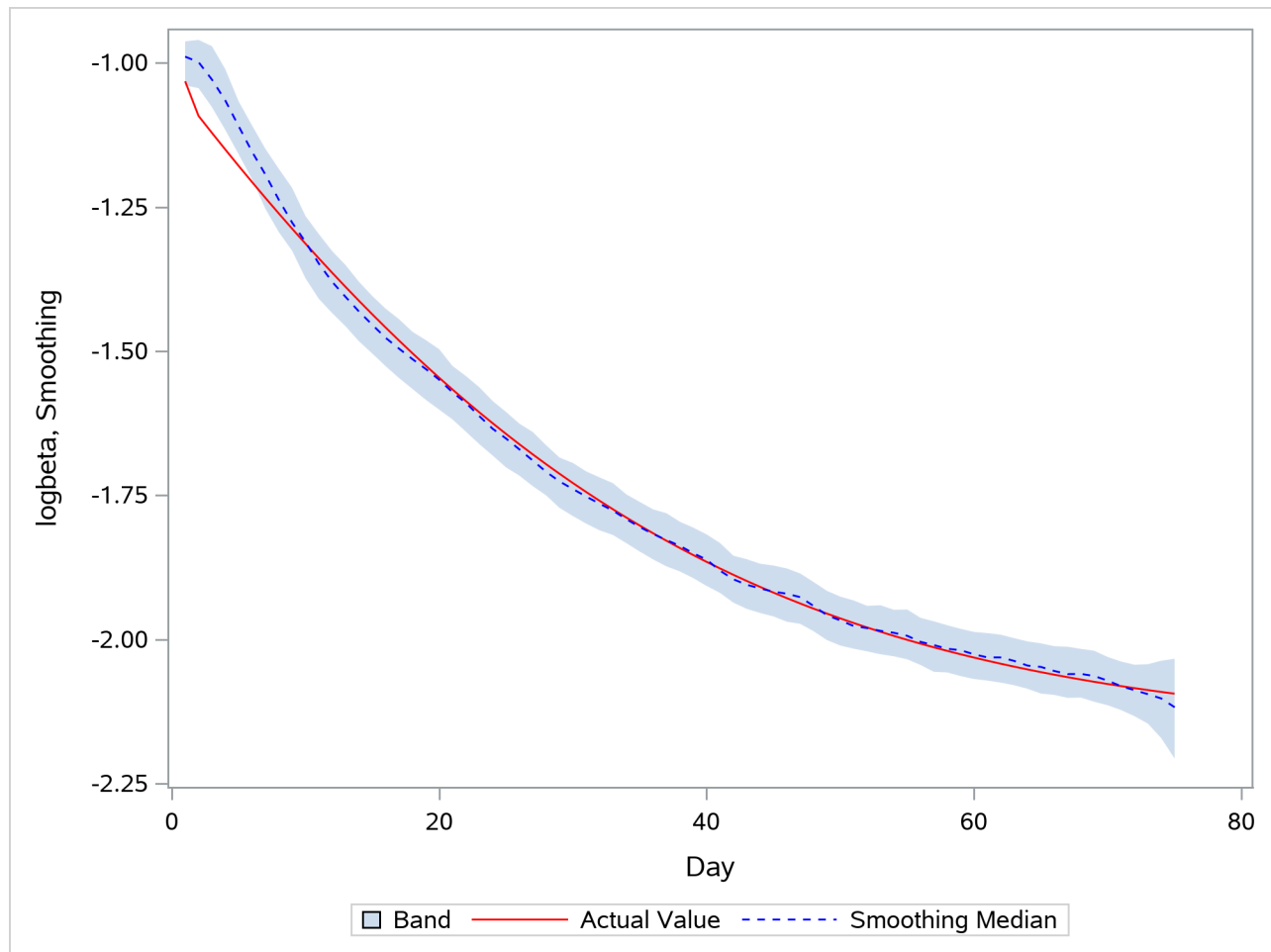
```
data combine_logbeta;
  set smo_s4 (keep=P_1 P_2 Q2);
  set sir_train (keep=t logbeta);
run;

proc sgplot data=combine_logbeta;
```

```

band x=t upper=P_2 lower=P_1 /transparency=.3;
series x=t y=logbeta /legendlabel='Actual Value' lineattrs=(color=red);
series x=t y=Q2 /legendlabel='Smoothing Median'
      lineattrs=(color=blue pattern=shortdash);
yaxis label='logbeta, Smoothing';
xaxis integer label='Day';
run;

```

Output 27.2.4 Smoothing Estimates vs. Actual Values, logbeta

The following statements create a relevant data table, `combine_REFF`, in the `Work` library to record the smoothing estimates and actual values of R_e . Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.2.5](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates.

```

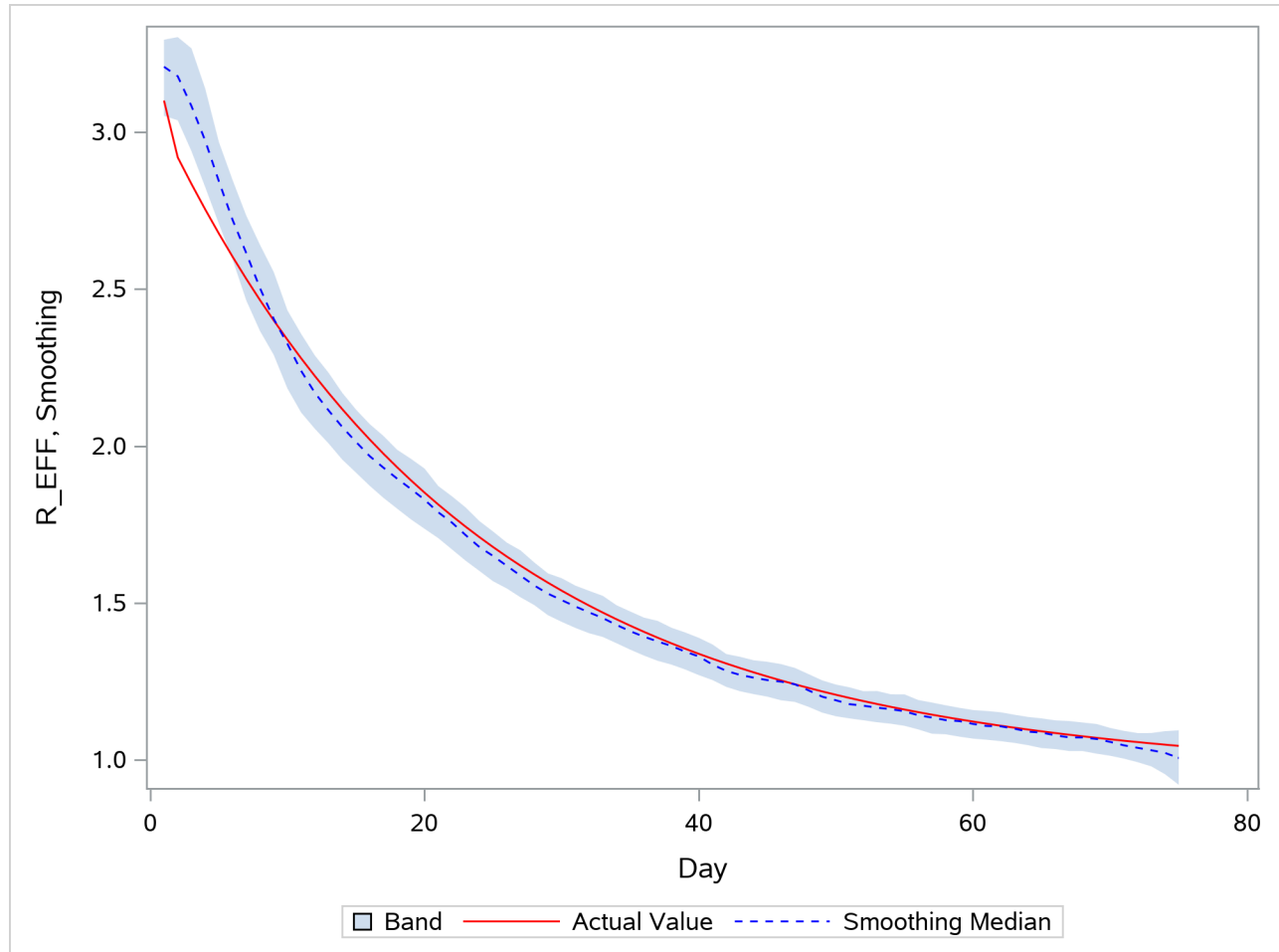
data combine_REFF;
  set smo_s6 (keep=P_1 P_2 Q2);
  set sir_train (keep=t R_EFF);
run;

proc sgplot data=combine_REFF;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  series x=t y=R_EFF /legendlabel='Actual Value' lineattrs=(color=red);

```

```
series x=t y=Q2 /legendlabel='Smoothing Median'
  lineattrs=(color=blue pattern=shortdash);
yaxis label='R_EFF, Smoothing';
xaxis integer label='Day';
run;
```

Output 27.2.5 Smoothing Estimates vs. Actual Values, Reff



The following statements load the forecasting estimates for R_e , ξ , and Y into the Work library and then sort each data set by the time ID:

```
data fc_6; set mylib.fc(where=(VariableID=6)); run;
proc sort data=fc_6; by _t; run;

data fc_7; set mylib.fc(where=(VariableID=7)); run;
proc sort data=fc_7; by _t; run;

data fc_8; set mylib.fc(where=(VariableID=8)); run;
proc sort data=fc_8; by _t; run;
```

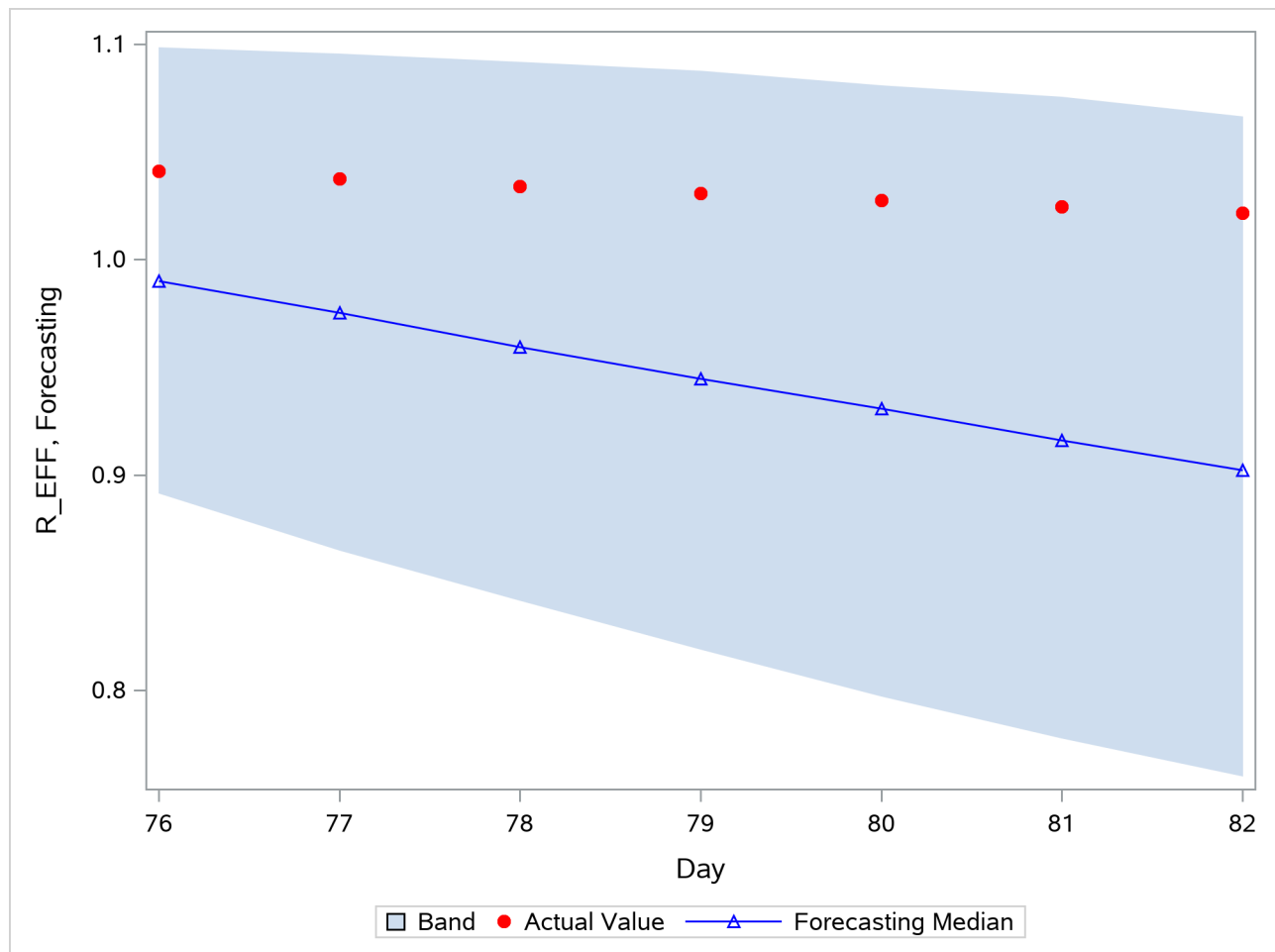
The following statements create a relevant data table, combine_REFF_forecast, in the Work library to record the forecasting estimates and actual values of R_e . Then they draw a plot to compare the forecasting estimates

to the actual values at each time step. The plot is shown in [Output 27.2.6](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the forecasting estimates.

```
data combine_REFF_forecast;
  set fc_6 (keep=P_1 P_2 Q2);
  set sir_test (keep=t R_EFF);
run;

proc sgplot data=combine_REFF_forecast;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  scatter x=t y=R_EFF /legendlabel='Actual Value'
    markerattrs=(color=red size=6 symbol=circlefilled);
  series x=t y=Q2 /legendlabel='Forecasting Median'
    markers markerattrs=(color=blue size=6 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  yaxis label='R_EFF, Forecasting';
  xaxis integer label='Day';
run;
```

Output 27.2.6 Forecasting Estimates vs. Actual Values, Reff



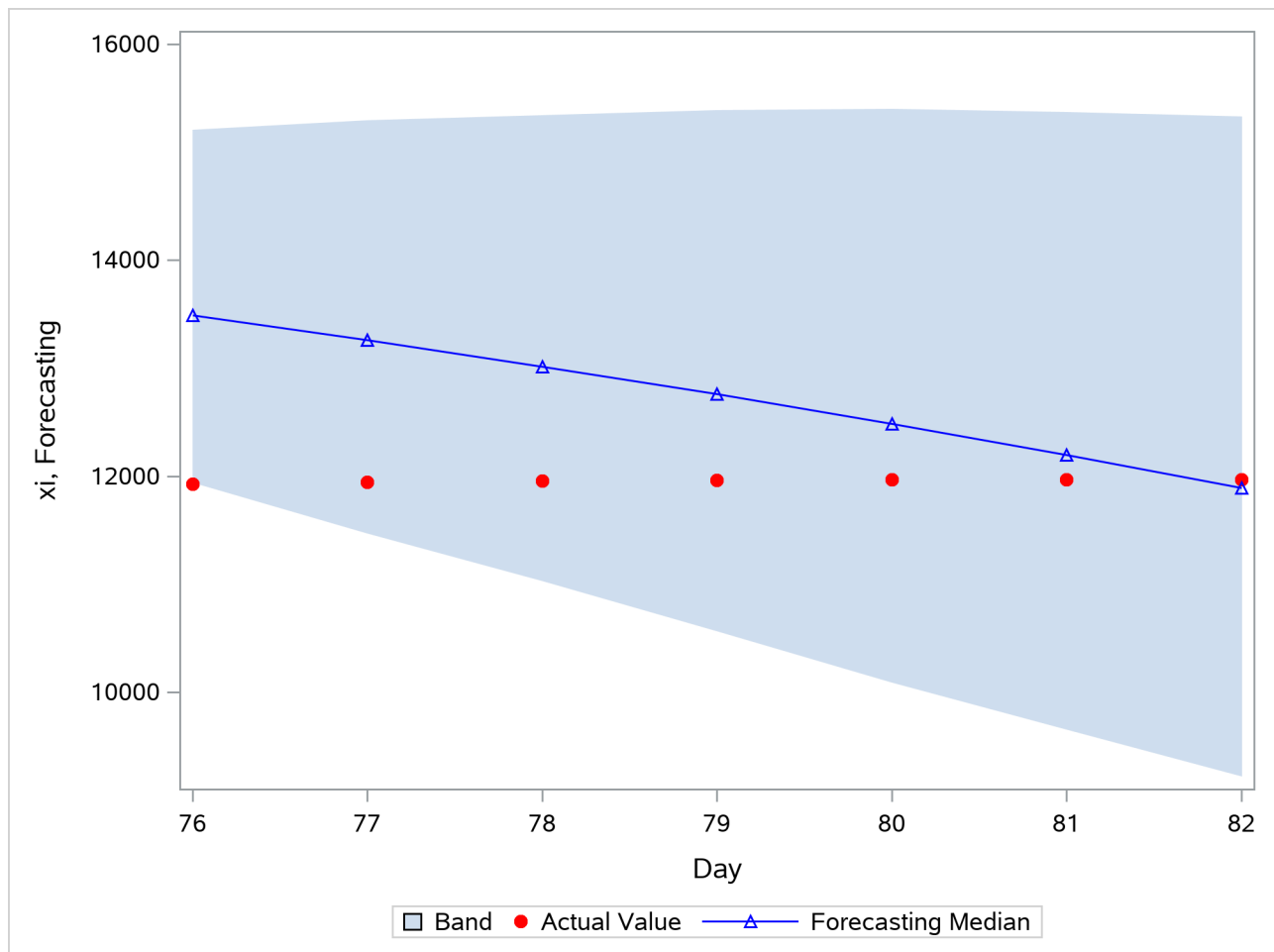
The following statements create a relevant data table, `combine_xi_forecast`, in the `Work` library to record the forecasting estimates and actual values of ξ . Then they draw a plot to compare the forecasting estimates to

the actual values at each time step. The plot is shown in [Output 27.2.7](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the forecasting estimates.

```
data combine_xi_forecast;
  set fc_7 (keep=P_1 P_2 Q2);
  set sir_test (keep=t xi);
run;

proc sgplot data=combine_xi_forecast;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  scatter x=t y=xi /legendlabel='Actual Value'
    markerattrs=(color=red size=6 symbol=circlefilled);
  series x=t y=Q2 /legendlabel='Forecasting Median'
    markers markerattrs=(color=blue size=6 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  yaxis label='xi, Forecasting';
  xaxis integer label='Day';
run;
```

Output 27.2.7 Forecasting Estimates vs. Actual Values, xi



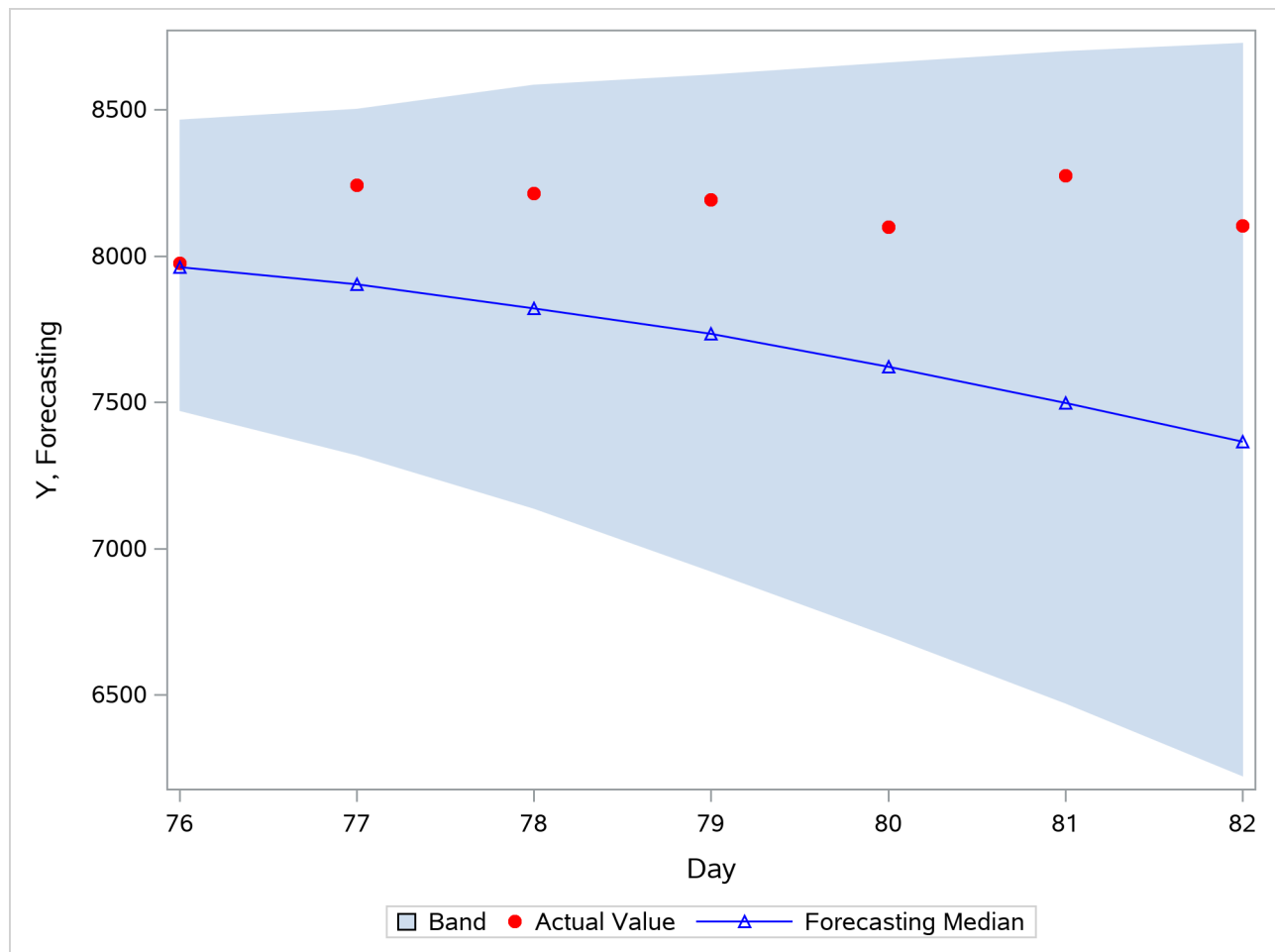
The following statements create a relevant data table, `combine_Y_forecast`, in the `Work` library to record the forecasting estimates and actual values of Y . Then they draw a plot to compare the forecasting estimates to

the actual values at each time step. The plot is shown in [Output 27.2.8](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the forecasting estimates.

```
data combine_Y_forecast;
  set fc_8 (keep=P_1 P_2 Q2);
  set sir_test (keep=t Y);
run;

proc sgplot data=combine_Y_forecast;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  scatter x=t y=Y /legendlabel='Actual Value'
    markerattrs=(color=red size=6 symbol=circlefilled);
  series x=t y=Q2 /legendlabel='Forecasting Median'
    markers markerattrs=(color=blue size=6 symbol=triangle)
    lineattrs=(color=blue pattern=solid);
  yaxis label='Y, Forecasting';
  xaxis integer label='Day';
run;
```

Output 27.2.8 Forecasting Estimates vs. Actual Values, Y



As shown in this example, by introducing the AR(1) process for the transition rate, this simple and powerful model successfully captures the dynamics in the epidemic spread, even when the true DGP follows a different

stochastic process. This model is applied to the real-data example in the section “[Example 27.3: Estimating the Effective Reproduction Number of COVID-19 Cases in Pennsylvania](#)” on page 2025.

Example 27.3: Estimating the Effective Reproduction Number of COVID-19 Cases in Pennsylvania

This example illustrates how to estimate the time-varying effective reproduction number of COVID-19 using the stochastic susceptible-infected-recovered (SSIR) model introduced in the section “[Example 27.2: Stochastic Susceptible-Infected-Recovered Model with a Time-Varying Transmission Rate](#)” on page 2014. The daily confirmed number of COVID-19 cases in Pennsylvania reported by the Center for Systems Science and Engineering at Johns Hopkins University (JHU CSSE) (Dong, Du, and Gardner 2020) is used.

The following DATA step inputs the number of new COVID-19 cases confirmed in Pennsylvania into the data set `penn_state_data`:

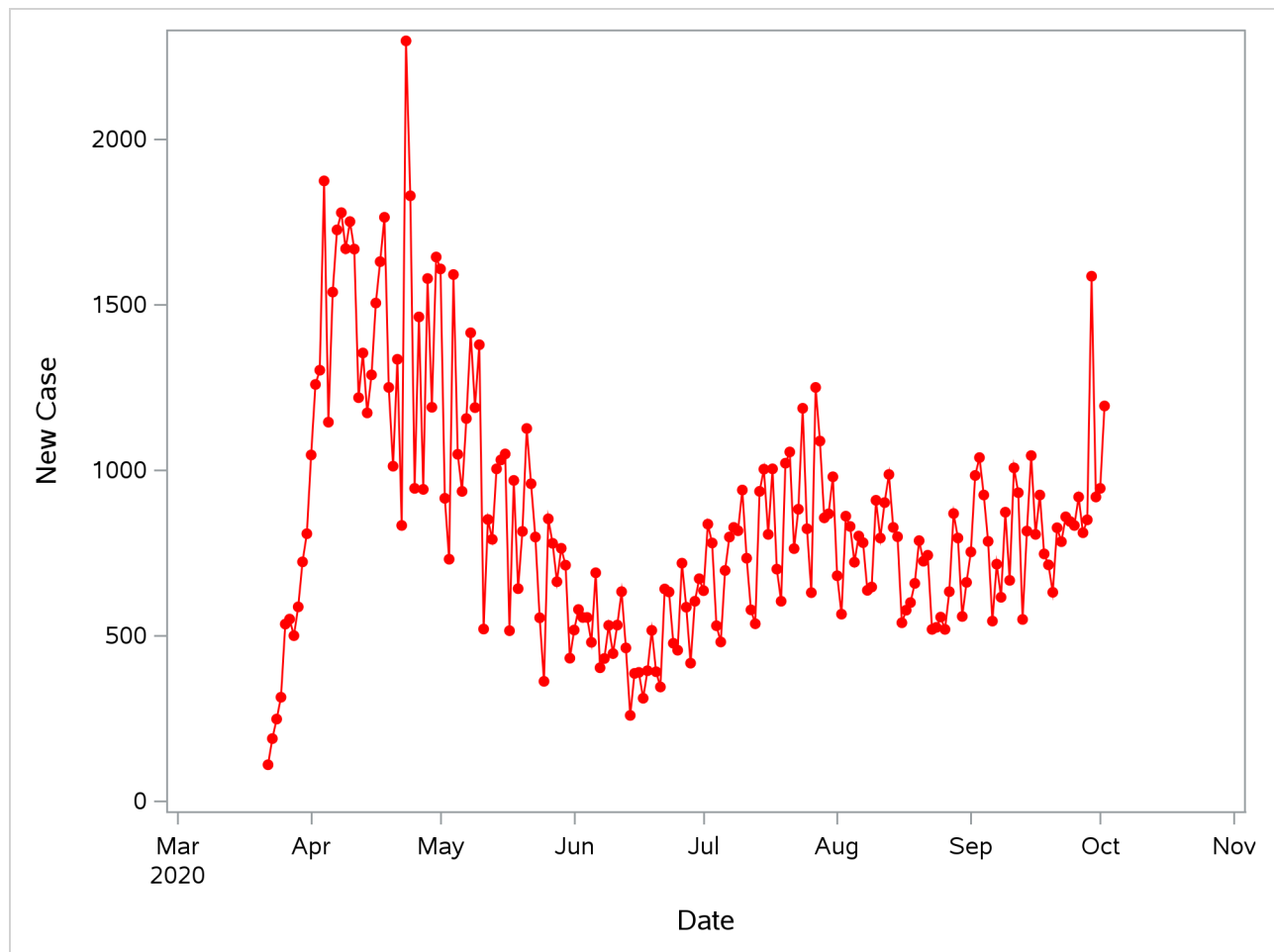
```
data penn_state_data;
    input date: anydtdte. cases newcase;
    format date DATE.;
    label date="Date" cases="cumulative cases" newcase="new case";
datalines;
22-Mar-20  509  110
23-Mar-20  698  189
24-Mar-20  946  248
25-Mar-20 1260  314
26-Mar-20 1795  535
27-Mar-20 2345  550

... more lines ...

25-Sep-20 159014  833
26-Sep-20 159933  919
27-Sep-20 160744  811
28-Sep-20 161594  850
29-Sep-20 163180 1586
30-Sep-20 164099  919
1-Oct-20   165044  945
2-Oct-20   166238 1194
;
```

The following statements plot the number of new cases confirmed in Pennsylvania from March 22 to September 28, 2020. The output is shown in [Output 27.3.1](#).

```
proc sgplot data=penn_state_data;
    series x=date y=newcase /
        markers markerattrs=(color=red size=5 symbol=circlefilled)
        lineattrs=(color=red);
    yaxis label='New Case';
    xaxis label='Date';
run;
```

Output 27.3.1 New Cases, Pennsylvania

The data start from March 22, 2020, because it is the first day when Pennsylvania had more than 100 new cases and more than 500 cumulative cases. The transition and measurement distributions for the SSIR model are exactly the same as the ones in the section “[Example 27.2: Stochastic Susceptible-Infected-Recovered Model with a Time-Varying Transmission Rate](#)” on page 2014. However, the initialization distribution must be adjusted because the first day of the data (March 22, 2020) is not the day when the first case was reported in Pennsylvania.

Consider the following initialization distribution:

$$\begin{aligned}
 ID_1 &\sim \text{uniform}(450, 550) \\
 IU_1 &\sim \text{uniform}\left(0.98\frac{110}{p} - 110, 3\frac{ID_1}{p} - 110\right) \\
 RC_1 &= 0 \\
 S_1 &= N - ID_1 - IU_1 \\
 \ln(\beta_1) &\sim \text{uniform}\left(\ln\left(\frac{2.75}{D}\right), \ln\left(\frac{3.3}{D}\right)\right) \\
 R_{e,1} &= \exp(\ln(\beta_1))\frac{S_1 D}{N} \\
 \xi_1 &= \exp(\ln(\beta_1))S_1\frac{IU_1 + ID_1}{N}
 \end{aligned}$$

Here all the denotations are defined in the same way as in the section “[Example 27.2: Stochastic Susceptible-Infected-Recovered Model with a Time-Varying Transmission Rate](#)” on page 2014. The cumulative number of reported cases until March 22, 2020, is 509, and the number of new cases that day is 110. Thus, it is reasonable to set initialization distributions for ID_1 and IU_1 as earlier. In the initialization distribution for IU_1 , $\frac{110}{p}$ can be seen as the expected value of the number of undocumented infectious individuals before that day; 110 is subtracted because the 110 cases are confirmed that day.

The following statements add a logical time ID (“t”) and split the data set such that the first 181 observations are used as the training set and the last 14 observations are used as the test set:

```

data penn_state_data;
  set penn_state_data;
  t = _N_;
run;

data penn_state_train;
  set penn_state_data;
  if (t le 181) then output;
run;

data penn_state_test;
  set penn_state_data;
  if (t > 181) then output;
run;

```

The following DATA step loads the training set into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```

data mylib.train;
  set penn_state_train;
run;

```

The following statements use PROC SMC to define the SSIR model and then use the PMCMC method to estimate the three parameters for the simulated data set. Moreover, the population of Pennsylvania is set as a macro variable. The median of the posterior parameter samples will be used as the parameters in the smoothing and forecasting process.

```

%let N = 12791692; *Pennsylvania population;
proc smc data=mylib.train seed = 1234 nthreads = 32;
  id time = date;
  var newcase;
  statevar S IU ID logBeta RC R_EFF xi;
  smooth nparticle=24000 alg=sir parm=median
    method = fixedlag(20) out(pct=2.5, 97.5)=mylib.sm;
  forecast nparticle=24000 alg=sir lead=14 parm=median
    out(pct=2.5, 97.5)=mylib.fc;
  parm D = 9.3, u1=-1.31, u2=0, p=0.25, rho=0.8, sigma_var=0.16
    /lower=(5.75 -2 -2 0.1 0.35 0.05) upper=(14 2 2 0.5 0.95 2);

  prior D ~ gamma(325.5, 0.0285714286);
  prior sigma_var ~ igamma(11,1);
  prior rho ~ beta(4,1);
  prior p ~ uniform(0.1, 0.5);
  prior u1 ~ normal(-1.31, 0.3);
  prior u2 ~ normal(0,1);

  initstate ID ~ uniform(450, 550);
  initstate IU ~ uniform(0.98*110/p-110, 3*110/p-110);
  initstate RC ~ degenerate(0);
  initstate S ~ degenerate(&N-ID-IU);
  initstate logBeta ~ uniform(log(2.75*1/D), log(3.43*1/D));
  initstate R_EFF ~ degenerate(exp(logBeta)*S*D/&N);
  initstate xi ~ degenerate(exp(logBeta)*S*(IU + ID)/&N);

  state S ~ degenerate(S.l1 - xi.l1);
  state RC ~ degenerate(RC.l1 + 1/D*(IU.l1+ID.l1));
  state IU ~ degenerate((1-1/D)*IU.l1 + xi.l1 - newcase.l1);
  state ID ~ degenerate((1-1/D)*ID.l1 + newcase.l1);
  state logBeta ~ normal(u1 + u2*_time + rho * (logBeta.l1-u1-u2*(_time-1)),
    sqrt(sigma_var*(1-rho**2)));
  state R_EFF ~ degenerate(exp(logBeta)*S*D/&N);
  state xi ~ degenerate(exp(logBeta)*S*(IU + ID)/&N);

  model newcase ~ normal(p*IU, 2*sqrt(p*(1-p)*ifn(IU>1, IU, 1)));

  learn nparticle=8000 algorithm=sir method=PMCMC(nsampl=5000
    nchain=8 nbi=4000 thin=4 seed=1
    outpost=mylib.out samset samsum
    sampler=rwm(ntu=200)
    statistics=all diagnostics=all);
run;

```

The summary tables of the model information are shown in [Output 27.3.2](#).

Output 27.3.2 Model Information Summary

The SMC Procedure

State Space Model Information	
Number of Parameters	6
Number of State Variables	7
Number of Dependent Variables	1
Number of Independent Variables	0
Seed Value	1234

Number of Observations	181
Number of Missing Observations	0

Initial Parameters Information			
Parameter	Initial Value	Lower Bound	Upper Bound
D	9.30000	5.75000	14.00000
u1	-1.31000	-2.00000	2.00000
u2	0	-2.00000	2.00000
p	0.25000	0.10000	0.50000
rho	0.80000	0.35000	0.95000
sigma_var	0.16000	0.05000	2.00000

The summary tables of the posterior samples are shown in [Output 27.3.3](#).

Output 27.3.3 Posterior Summary

MCMC Sampler Settings	
MCMC Sampling Method	RWM
Random Seed	1
Transformed Sampling	False
Number of Chains	8
Number of Sampling Iterations	5000
Number of Burn-in Iterations	4000
Thinning Interval	4
Number of Tuning Iterations per Tuning Stage	200
Number of Tuning Stages	24
Target Metropolis Acceptance Rate	0.234
Initial RWM Proposal Scale	2.38
Tuning Weight	0.75
Acceptance Tolerance	0.075

The correlation and covariance matrices of the parameter samples that are generated by the PMCMC method are shown in [Output 27.3.4](#).

Output 27.3.4 Correlation and Covariance Matrices

Posterior Covariance Matrix						
Parameter	D	u1	u2	p	rho	sigma_var
D	0.083697	-0.02117	0.000188	0.005364	0.004039	-0.01511
u1	-0.02117	0.038469	-0.00039	-0.0023	0.004254	0.006524
u2	0.000188	-0.00039	4.991E-6	5.1E-6	-0.0001	-1.69E-6
p	0.005364	-0.0023	5.1E-6	0.001763	0.002174	-0.00311
rho	0.004039	0.004254	-0.0001	0.002174	0.006983	-0.00536
sigma_var	-0.01511	0.006524	-1.69E-6	-0.00311	-0.00536	0.012344

The following statements load the smoothing result for the effective reproduction number (VariableID = 6) into the Work library and then sort that data set by the logical time ID:

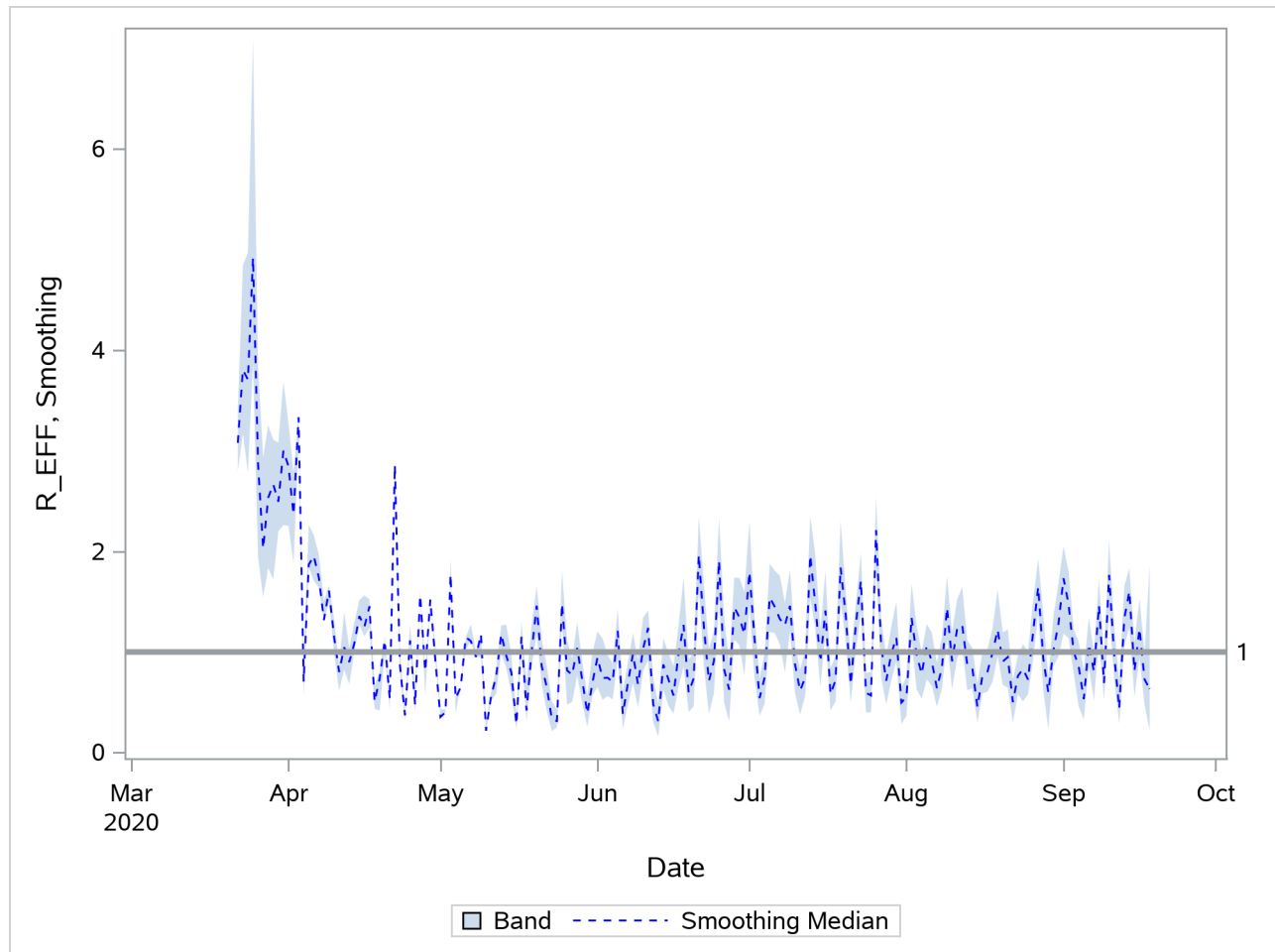
```
data smo_s6; set mylib.sm(where=(VariableID=6)); run;
proc sort data=smo_s6; by _t; run;
```

The following statements create a relevant data table, combine_REFF, in the Work library to record the smoothing estimates and actual values of the effective reproduction number. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The output is shown in [Output 27.3.5](#), where the shadow band in the plot indicates the range between the 2.5th and 97.5th percentiles of the smoothing estimates and the horizontal reference line denotes the line $y = 1$.

```
data combine_REFF;
  set smo_s6 (keep=P_1 P_2 Q2 _t);
  set penn_state_train (keep=date);
run;

proc sgplot data=combine_REFF;
  band x=date upper=P_2 lower=P_1 /transparency=.3;
  series x=date y=Q2 /legendlabel='Smoothing Median'
    lineattrs=(color=blue pattern=shortdash);
  yaxis label='R_EFF, Smoothing';
  xaxis label='Date';
  refline 1 / axis=y label='1' lineattrs=(thickness=3);
run;
```


Output 27.3.5 Smoothing Estimates vs. Actual Values, R_e



The following statements load the forecasting result for each state variable into the Work library and then sort each data set by the time ID:

```
data fc_8; set mylib.fc(where=(VariableID=8)); run;
proc sort data=fc_8; by _t; run;
```

The following statements create a relevant data table, `combine_Y_forecast`, in the Work library to record the forecasting estimates and actual values of new cases. Then they draw a plot to compare the forecasting estimates to the actual values at each time step. The plot is shown in [Output 27.3.6](#), where the shadow band indicates the range between the 2.5th and 97.5th percentiles of the forecasting estimates.

```
data combine_Y_forecast;
  set fc_8 (keep=P_1 P_2 Q2);
  set penn_state_test (keep=date t newcase);
run;

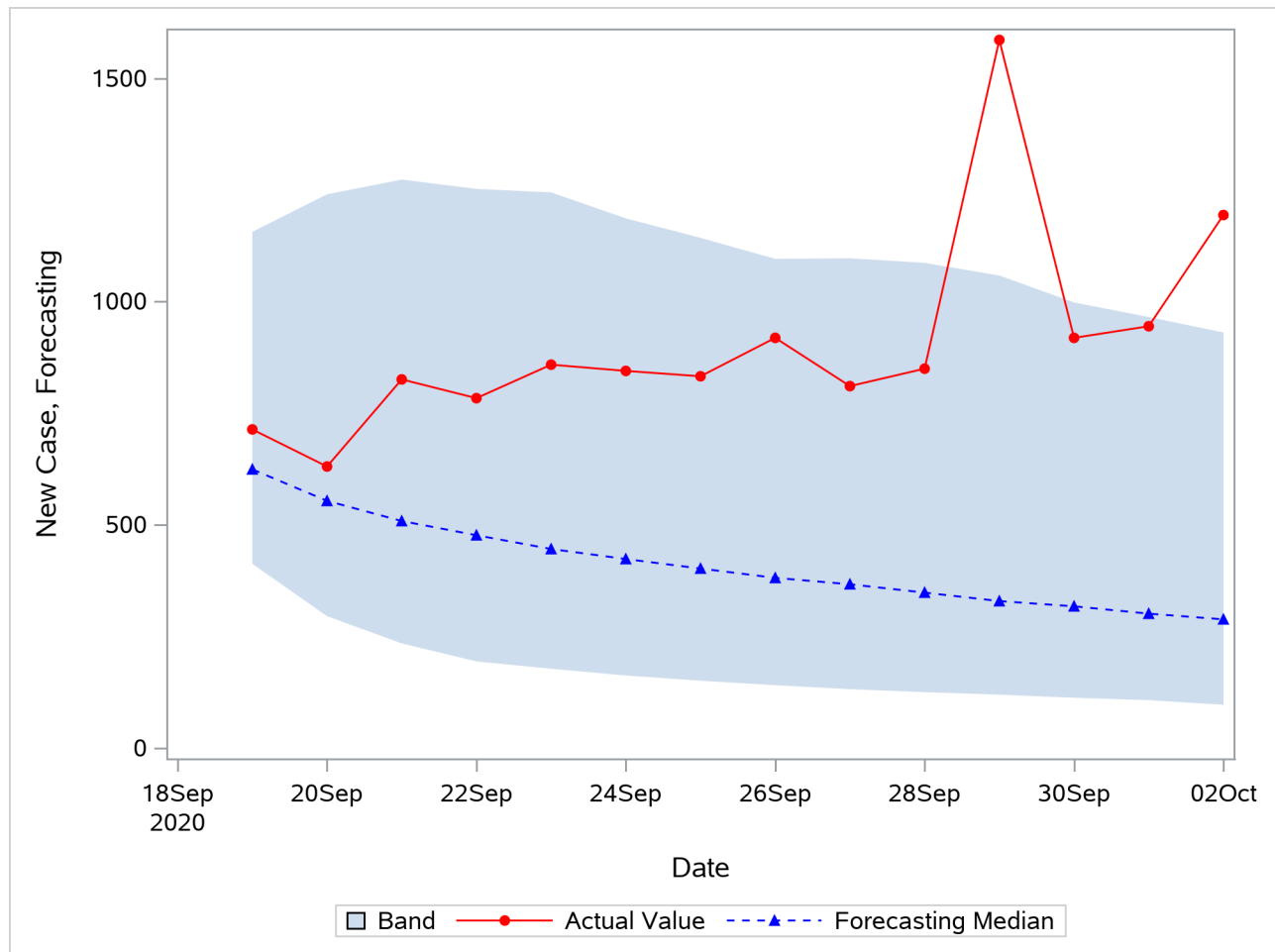
proc sgplot data=combine_Y_forecast;
  band x=date upper=P_2 lower=P_1 /transparency=.3;
  series x=date y=newcase /legendlabel='Actual Value'
    markers markerattrs=(color=red size=5 symbol=circlefilled)
    lineattrs=(color=red);
```

```

series x=date y=Q2 /legendlabel='Forecasting Median'
      markers markerattrs=(color=blue size=5 symbol=trianglefilled)
      lineattrs=(color=blue pattern=shortdash);
yaxis label='New Case, Forecasting';
xaxis label='Date';
run;

```

Output 27.3.6 Forecasting Estimates vs. Actual Values, Y



Example 27.4: Estimating the Dynamic Bayesian Network

This example illustrates how to estimate a dynamic Bayesian network (DBN). A heating, ventilation, and air conditioning (HVAC) system is used as a prototype network of DBN.

In this HVAC system, there are four state variables:

- OutSensorStatus: category variable, outer sensor status (WORK=1 or FAIL=0)
- InSensorStatus: category variable, inner sensor status (WORK=1 or FAIL=0)
- OutTemp: continuous variable, actual outer temperature
- InTemp: continuous variable, actual inner temperature

There are three observation variables:

- OutTempRead: dependent continuous variable, outer temperature reading
- InTempRead: dependent continuous variable, inner temperature reading
- HVAC: independent category variable, HVAC mode (heating, cooling or normal)

The HVAC system can be represented by the following state space model (or the DBN model), where the subscript t denotes the time index:

- Initialization distributions:

$$\text{OutSensorStatus}_1 \sim \text{Bernoulli}(0.5)$$

$$\text{InSensorStatus}_1 \sim \text{Bernoulli}(0.5)$$

$$\text{OutTemp}_1 \sim N(\text{OutTempRead}_1, 4)$$

$$\text{InTemp}_1 \sim N(\text{InTempRead}_1, 4)$$

- Transition distributions:

$$\Pr(\text{OutSensorStatus}_t = j | \text{OutSensorStatus}_{t-1} = i) = P_{ij}, \quad i, j \in \{0, 1\}$$

$$\Pr(\text{InSensorStatus}_t = j | \text{InSensorStatus}_{t-1} = i) = Q_{ij}, \quad i, j \in \{0, 1\}$$

$$\text{OutTemp}_t = \text{OutTemp}_{t-1} + \frac{45\pi}{287} \cos\left(\frac{2\pi}{287}(t-2) - \frac{\pi}{2}\right) + \epsilon_t, \epsilon_t \sim N(0, \text{Var}_{\text{OT}})$$

$$\text{InTemp}_t = a_i \text{InTemp}_{t-1} + b_i \text{OutTemp}_{t-1} + \eta_{i,t}, \eta_{i,t} \sim N(0, \text{Var}_{i,\text{IT}}), i = \text{HVAC}_{t-1}$$

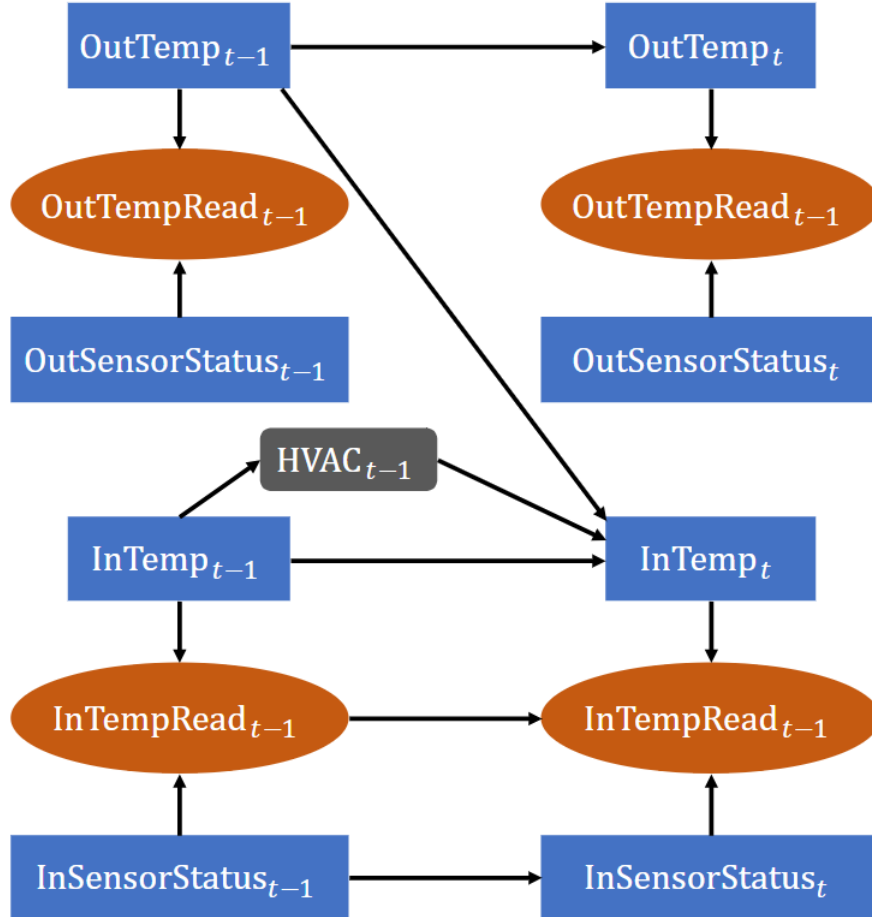
- Measurement distributions:

$$\text{OutTempRead}_t = \begin{cases} \text{OutTemp}_t + \xi_{2,t}, \xi_{2,t} \stackrel{\text{iid}}{\sim} N(0, \text{Var}_1), & \text{if } \text{OutSensorStatus}_t = 1 \\ \text{OutTempRead}_{t-1} + \xi_{1,t}, \xi_{1,t} \stackrel{\text{iid}}{\sim} N(0, \text{Var}_0), & \text{if otherwise} \end{cases}$$

$$\text{InTempRead}_t = \begin{cases} \text{InTemp}_t + \xi_{2,t}, \xi_{2,t} \stackrel{\text{iid}}{\sim} N(0, \text{Var}_1), & \text{if } \text{InSensorStatus}_t = 1 \\ \text{InTempRead}_{t-1} + \xi_{1,t}, \xi_{1,t} \stackrel{\text{iid}}{\sim} N(0, \text{Var}_0), & \text{otherwise} \end{cases}$$

Here $\text{Bernoulli}(p)$ denotes the Bernoulli distribution whose probability of success is p (success: 1; failure: 0), and $N(\mu, \sigma^2)$ denotes the normal distribution, whose mean is μ and variance is σ^2 . Figure 27.11 shows the relationships within the HVAC system.

Figure 27.11 DBN States and Their Transitions



There are 16 parameters. The initial values, bounds, and actual values that the simulation data set uses for the parameters are listed in Table 27.23. Each of the variance parameters technically has an upper bound of ∞ , but they are constrained to have an upper bound of 0.99. This represents prior information that the variances cannot be larger than 0.99, which makes sense for this example because it is known that the true values of these parameters are all less than 0.99. Also, adding these constraints makes the model significantly easier to fit.

Table 27.23 Parameter Information

Parameter	Initial Value	Range	Actual Value
$\theta_1 = P_{0,0}$	0.65	(0.01, 0.99)	0.85
$\theta_2 = P_{1,0}$	0.3	(0.01, 0.99)	0.15
$\theta_3 = Q_{0,0}$	0.65	(0.01, 0.99)	0.85
$\theta_4 = Q_{1,0}$	0.3	(0.01, 0.99)	0.15
$\theta_5 = a_1$	0.65	(0.01, 0.99)	0.9

Table 27.23 *continued*

Parameter	Initial Value	Range	Actual Value
$\theta_6 = b_1$	0.3	(0.01, 0.99)	0.2
$\theta_7 = \text{Var}_{1,\text{IT}}$	0.7	(0.01, 0.99)	0.85
$\theta_8 = a_1$	0.55	(0.01, 0.99)	0.7
$\theta_9 = b_1$	0.3	(0.01, 0.99)	0.2
$\theta_{10} = \text{Var}_{2,\text{IT}}$	0.7	(0.01, 0.99)	0.85
$\theta_{11} = a_1$	0.6	(0.01, 0.99)	0.8
$\theta_{12} = b_1$	0.3	(0.01, 0.99)	0.2
$\theta_{13} = \text{Var}_{3,\text{IT}}$	0.7	(0.01, 0.99)	0.85
$\theta_{14} = \text{Var}_{\text{OT}}$	0.6	(0.01, 0.99)	0.3
$\theta_{15} = \text{Var}_1$	0.6	(0.01, 0.99)	0.4
$\theta_{16} = \text{Var}_0$	0.5	(0.01, 0.99)	0.2

The following statements simulate 288 observations on the basis of the given HVAC system:

```

data hvacSystem;
  call streaminit(123);
  OSlag = 0;
  ISlag = 0;
  OTlag = 55;
  ITlag = 70;
  OTReadlag = 58;
  ITReadlag = 67;
  HVAClag = 3;
  pi=constant("pi");
  do t = 1 to 288*2;
    time = t;
    u = rand("uniform");
    if OSlag = 0 then OS=ifn(u<0.85,0,1);
    else OS=ifn(u<0.15,0,1);
    u = rand("uniform");
    if ISlag = 0 then IS=ifn(u<0.85,0,1);
    else IS=ifn(u<0.15,0,1);
    OT = OTlag+45*pi/287*cos(2*pi/287*(t-2)-0.5*pi)+ sqrt(0.3)*rand("normal");
    if HVAClag = 1 then IT = 0.9*ITlag + 0.2*OTlag + sqrt(0.85)*rand("normal");
    if HVAClag = 2 then IT = 0.7*ITlag + 0.2*OTlag + sqrt(0.85)*rand("normal");
    if HVAClag = 3 then IT = 0.8*ITlag + 0.2*OTlag + sqrt(0.85)*rand("normal");
    if OS = 0 then OTRead = OTReadlag + sqrt(0.4)*rand("normal");
    else OTRead = OT + sqrt(0.2)*rand("normal");
    if IS = 0 then ITRead = ITReadlag + sqrt(0.4)*rand("normal");
    else ITRead = IT + sqrt(0.2)*rand("normal");
    if IT < 68 then HVAC = 1;
    else HVAC=ifn(IT>74,2,3);
    output;
    OTReadlag = OTRead;
    ITReadlag = ITRead;
    OSlag = OS;
    ISlag = IS;
  end;

```

```

        OTlag = OT;
        ITlag = IT;
        HVAClag = HVAC;
    end;
run;

```

The following DATA step loads the data set hvacSystem into a data table in your session that is associated with the mylib libref. The DATA step assumes that your libref is named mylib, but you can substitute any appropriately defined libref.

```

data mylib.hvacSystem;
    set hvacSystem;
run;

```

In the SMC procedure, two categorical variables must be positioned before the two noncategorical ones in the state variable vector $\{x_i\}_{i=1}^4$. For the four state variables, x_1 denotes OutSensorStatus; x_2 denotes InSensorStatus; x_3 denotes OutTemp; and x_4 denotes InTemp.

For the two dependent variables, OutTempRead is denoted by y_1 and InTempRead is denoted by y_2 . The independent variable, HVAC, is denoted by z_1 .

The following statements use the SUBMITMODEL statement in the SMC procedure to define the HVAC system and then use the PMCMC method to estimate the 16 parameters for the simulated data set. The median of the posterior parameter samples will be used as the parameter in the smoothing process. Moreover, in the SUBMITMODEL statement, θ_i is represented as theta[i] for $i = 1, \dots, 16$; x_i is represented by x[i] for $i = 1, \dots, 4$; y_i is represented by y[i] for $i = 1, 2$; and z_1 is represented by z[1]. You must specify the *dependent-variables* in the SUBMITMODEL statement in the same order as $\{y_i\}_{i=1}^2$. This example can take around 14 hours or longer to run, depending on which hardware you run it on. The TRANSFORM option for the PMCMC method in the LEARN statement significantly improves the efficiency of the sampler for this model, especially when you combine it with the constraints on the variance parameters.

```

proc smc data=mylib.hvacSystem nthreads=32 seed=1;
    id time = t;
    learn nparticle=8000 algorithm=BF method=PMCMC(nsampl=40000
        samsum samset nbi=20000 thin=5 seed=123
        outpost = mylib.outle sampler = rwm(transform ntu=700 ntunestage=48)
        statistics = all diagnostics(settings)=all) ;
    smooth nparticle = 20480 alg=bf parm=median method=fixedlag(30) out=mylib.sm;
    submitmodel OTRead ITRead = HVAC / nstate(2,2)=4 nParm = 16 name=dbn
        xlag = (state=1) ylag=(model=1) zlag=(state=1);
    subroutine dbn_sampprio(theta[*]);
        outargs theta;
        theta[1] = 0.65;
        theta[2] = 0.3;
        theta[3] = 0.65;
        theta[4] = 0.3;
        theta[5] = 0.65;
        theta[6] = 0.3;
        theta[7] = 0.7;
        theta[8] = 0.55;
        theta[9] = 0.3;
        theta[10] = 0.7;
        theta[11] = 0.6;
        theta[12] = 0.3;

```

```

    theta[13] = 0.7;
    theta[14] = 0.6;
    theta[15] = 0.6;
    theta[16] = 0.5;
endsub;

subroutine dbn_lowerbounds(theta[*]);
    outargs theta;
    theta[1] = 0.01;
    theta[2] = 0.01;
    theta[3] = 0.01;
    theta[4] = 0.01;
    theta[5] = 0.01;
    theta[6] = 0.01;
    theta[7] = 0.01;
    theta[8] = 0.01;
    theta[9] = 0.01;
    theta[10] = 0.01;
    theta[11] = 0.01;
    theta[12] = 0.01;
    theta[13] = 0.01;
    theta[14] = 0.01;
    theta[15] = 0.01;
    theta[16] = 0.01;
endsub;

subroutine dbn_upperbounds(theta[*]);
    outargs theta;
    theta[1] = 0.99;
    theta[2] = 0.99;
    theta[3] = 0.99;
    theta[4] = 0.99;
    theta[5] = 0.99;
    theta[6] = 0.99;
    theta[8] = 0.99;
    theta[9] = 0.99;
    theta[11] = 0.99;
    theta[12] = 0.99;

    theta[7] = 0.99;
    theta[10] = 0.99;
    theta[13] = 0.99;
    theta[14] = 0.99;
    theta[15] = 0.99;
    theta[16] = 0.99;
endsub;

subroutine dbn_sampinit(x[*],xlag[*],ylag[*],z[*],zlag[*],theta[*],_time);
    outargs x, xlag;
    a = rand('uniform');
    x[1] = ifn(a<0.5, 1, 0); /*OutStatus*/
    a = rand('uniform');
    x[2] = ifn(a<0.5, 1, 0); /*InnerStatus*/
    x[3] = 57 + 2*rand('NORMAL', 0, 1); /*OutTempRead*/

```

```

        x[4] = 66 + 2*rand('NORMAL', 0, 1); /*InTempRead*/
    endsub;

function dbn_evalmeas(x[*],xlag[*],y[*],ylag[*],z[*],zlag[*],theta[*],_time);
    if x[1] = 0 then lp1 = LOGPDF('NORMAL',y[1],ylag[1],sqrt(theta[15]));
    else lp1 = LOGPDF('NORMAL',y[1],x[3],sqrt(theta[16]));
    if x[2] = 0 then lp2 = LOGPDF('NORMAL',y[2],ylag[2],sqrt(theta[15]));
    else lp2 = LOGPDF('NORMAL',y[2],x[4],sqrt(theta[16]));
    return (lp1+lp2);
endsub;

subroutine dbn_samptran(x[*],xlag[*],ylag[*],z[*],zlag[*],theta[*],_time);
    outargs x;
    pi=constant('PI');
    a = rand('uniform');
    if xlag[1] = 0 then x[1] = ifn(a<theta[1],0,1);
    if xlag[1] = 1 then x[1] = ifn(a<theta[2],0,1);
    a = rand('uniform');
    if xlag[2] = 0 then x[2] = ifn(a<theta[3],0,1);
    if xlag[2] = 1 then x[2] = ifn(a<theta[4],0,1);
    x[3] = xlag[3] + 45*pi/287*cos(2*pi/287*(_time-2)-0.5*pi)
        +sqrt(theta[14])*rand('NORMAL', 0, 1);
    if zlag[1] = 1 then x[4] = theta[5]*xlag[4] + theta[6]*xlag[3]
        +sqrt(theta[7])*rand('NORMAL', 0, 1);
    if zlag[1] = 2 then x[4] = theta[8]*xlag[4] + theta[9]*xlag[3]
        + sqrt(theta[10])*rand('NORMAL', 0, 1);
    if zlag[1] = 3 then x[4] = theta[11]*xlag[4] + theta[12]*xlag[3]
        +sqrt(theta[13])*rand('NORMAL', 0, 1);
endsub;

function dbn_evalprio(theta[*]);
    return (0);
endsub;
endsubmit;
run;

```

The summary tables of the model information are shown in [Output 27.4.1](#).

Output 27.4.1 Model Information Summary**The SMC Procedure**

State Space Model Information	
Number of Parameters	16
Number of State Variables	4
Number of Dependent Variables	2
Number of Independent Variables	1
Seed Value	1

Number of Observations	576
Number of Missing Observations	0

Initial Parameters Information			
Parameter	Initial Value	Lower Bound	Upper Bound
theta_1	0.65000	0.01000	0.99000
theta_2	0.30000	0.01000	0.99000
theta_3	0.65000	0.01000	0.99000
theta_4	0.30000	0.01000	0.99000
theta_5	0.65000	0.01000	0.99000
theta_6	0.30000	0.01000	0.99000
theta_7	0.70000	0.01000	0.99000
theta_8	0.55000	0.01000	0.99000
theta_9	0.30000	0.01000	0.99000
theta_10	0.70000	0.01000	0.99000
theta_11	0.60000	0.01000	0.99000
theta_12	0.30000	0.01000	0.99000
theta_13	0.70000	0.01000	0.99000
theta_14	0.60000	0.01000	0.99000
theta_15	0.60000	0.01000	0.99000
theta_16	0.50000	0.01000	0.99000

The summary tables of the posterior samples are shown in [Output 27.4.2](#).

Output 27.4.2 Posterior Summary

MCMC Sampler Settings	
MCMC Sampling Method	RWM
Random Seed	123
Transformed Sampling	True
Number of Chains	1
Number of Sampling Iterations	40000
Number of Burn-in Iterations	20000
Thinning Interval	5
Number of Tuning Iterations per Tuning Stage	700
Number of Tuning Stages	48
Target Metropolis Acceptance Rate	0.234
Initial RWM Proposal Scale	2.38
Tuning Weight	0.75
Acceptance Tolerance	0.075

The correlation and covariance matrices of the parameter samples that the PMCMC method generates are shown in [Output 27.4.3](#).

Output 27.4.3 Correlation and Covariance Matrices

Posterior Covariance Matrix											
Parameter	theta_1	theta_2	theta_3	theta_4	theta_5	theta_6	theta_7	theta_8	theta_9	theta_10	theta_11
theta_1	0.000128	-0.00022	0.000148	-0.00004	0.000089	-0.0001	0.000028	0.000055	-0.00004	0.00023	9.909E-6
theta_2	-0.00022	0.00146	-0.00023	0.000135	-0.00004	0.00003	-0.00011	-0.00008	0.000058	-0.00087	-4.06E-6
theta_3	0.000148	-0.00023	0.000401	-0.00019	0.000093	-0.0001	0.000065	0.00013	-0.0001	0.000292	0.000014
theta_4	-0.00004	0.000135	-0.00019	0.000553	0.000055	-0.00006	-0.00002	0.000036	-0.00003	-0.00034	-0.00001
theta_5	0.000089	-0.00004	0.000093	0.000055	0.000174	-0.00018	0.000015	-8.86E-6	5.716E-6	0.000094	-0.00001
theta_6	-0.0001	0.00003	-0.0001	-0.00006	-0.00018	0.000194	-0.00001	7.245E-7	4.067E-7	-0.00008	8.25E-6
theta_7	0.000028	-0.00011	0.000065	-0.00002	0.000015	-0.00001	0.00002	9.132E-6	-6.51E-6	0.000142	9.853E-7
theta_8	0.000055	-0.00008	0.00013	0.000036	-8.86E-6	7.245E-7	9.132E-6	0.00024	-0.00019	-0.00025	9.947E-6
theta_9	-0.00004	0.000058	-0.0001	-0.00003	5.716E-6	4.067E-7	-6.51E-6	-0.00019	0.00015	0.0002	-5.02E-6
theta_10	0.00023	-0.00087	0.000292	-0.00034	0.000094	-0.00008	0.000142	-0.00025	0.0002	0.002265	0.000057
theta_11	9.909E-6	-4.06E-6	0.000014	-0.00001	-0.00001	8.25E-6	9.853E-7	9.947E-6	-5.02E-6	0.000057	0.000041
theta_12	-0.00001	9.458E-6	-0.00002	0.000017	0.000015	-9.12E-6	-1.46E-6	-0.00001	5.329E-6	-0.00005	-0.00004
theta_13	0.000173	-0.00044	4.088E-6	0.000019	-0.00024	0.000209	-0.00005	0.000631	-0.00049	-0.001	0.000106
theta_14	0.000116	-0.00001	-0.00014	0.000088	0.00006	-0.00006	-0.00001	-0.00017	0.000127	0.000945	7.607E-6
theta_15	0.000013	0.000207	0.000059	0.000133	0.000176	-0.00018	-0.00002	-0.00003	0.000019	-0.00032	-6.31E-6
theta_16	-0.00007	0.000369	-0.00007	-0.00014	-0.00013	0.000126	-0.00008	0.000191	-0.00015	-0.00109	-0.00003

Posterior Covariance Matrix					
Parameter	theta_12	theta_13	theta_14	theta_15	theta_16
theta_1	-0.00001	0.000173	0.000116	0.000013	-0.00007
theta_2	9.458E-6	-0.00044	-0.00001	0.000207	0.000369
theta_3	-0.00002	4.088E-6	-0.00014	0.000059	-0.00007
theta_4	0.000017	0.000019	0.000088	0.000133	-0.00014
theta_5	0.000015	-0.00024	0.00006	0.000176	-0.00013
theta_6	-9.12E-6	0.000209	-0.00006	-0.00018	0.000126
theta_7	-1.46E-6	-0.00005	-0.00001	-0.00002	-0.00008
theta_8	-0.00001	0.000631	-0.00017	-0.00003	0.000191
theta_9	5.329E-6	-0.00049	0.000127	0.000019	-0.00015
theta_10	-0.00005	-0.001	0.000945	-0.00032	-0.00109
theta_11	-0.00004	0.000106	7.607E-6	-6.31E-6	-0.00003
theta_12	0.000038	-0.00011	-2.05E-8	8.629E-6	0.000025
theta_13	-0.00011	0.003591	-0.00041	-0.00047	0.000845
theta_14	-2.05E-8	-0.00041	0.00161	-0.00002	-0.00008
theta_15	8.629E-6	-0.00047	-0.00002	0.000442	0.000104
theta_16	0.000025	0.000845	-0.00008	0.000104	0.001074

The following statements load the smoothing estimates for each state variable into the Work library and then sort each data set by the time ID:

```
data smo_s1; set mylib.sm(where=(variableID=1)); run;
proc sort data=smo_s1; by timeID; run;

data smo_s2; set mylib.sm(where=(variableID=2)); run;
proc sort data=smo_s2; by timeID; run;

data smo_s3; set mylib.sm(where=(variableID=3)); run;
proc sort data=smo_s3; by timeID; run;
```

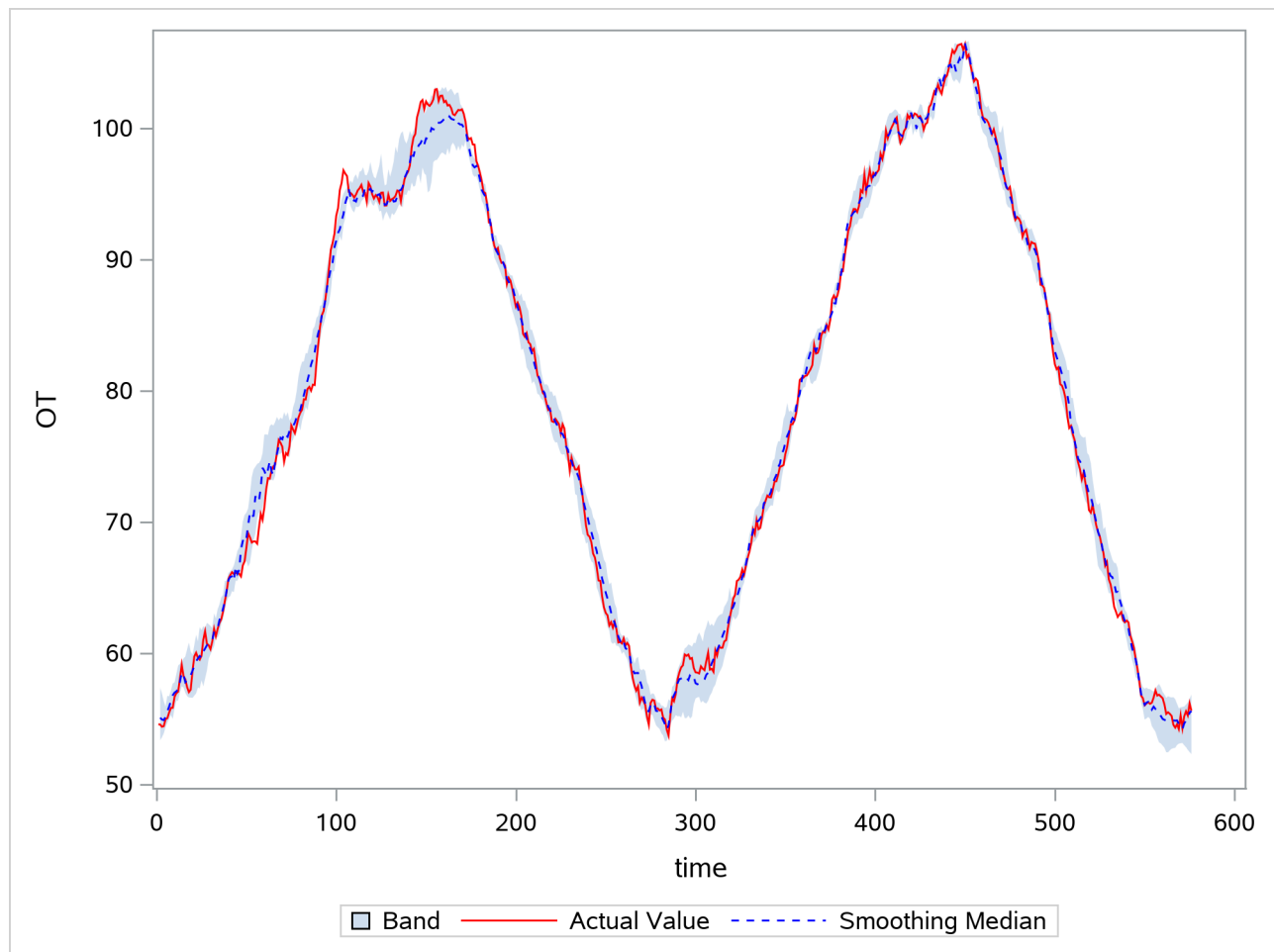
```
data smo_s4; set mylib.sm(where=(variableID=4)); run;
proc sort data=smo_s4; by timeId; run;
```

The following statements create a relevant data table, `combine_s3`, in the `Work` library to record the smoothing estimates and actual values of `OutTemp`. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.4.4](#), where the shadow band indicates the range from the 2.5th to the 97.5th percentiles of the smoothing estimates.

```
data combine_s3;
  set smo_s3;
  set hvacSystem;
run;

proc sgplot data=combine_s3;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  series x=Time y=OT /legendlabel='Actual Value' lineattrs=(color=red);
  series x=Time y=Q2/ legendlabel='Smoothing Median'
    lineattrs=(color=blue pattern=shortdash);
run;
```

Output 27.4.4 Smoothing Estimates vs. Actual Values, `OutTemp`

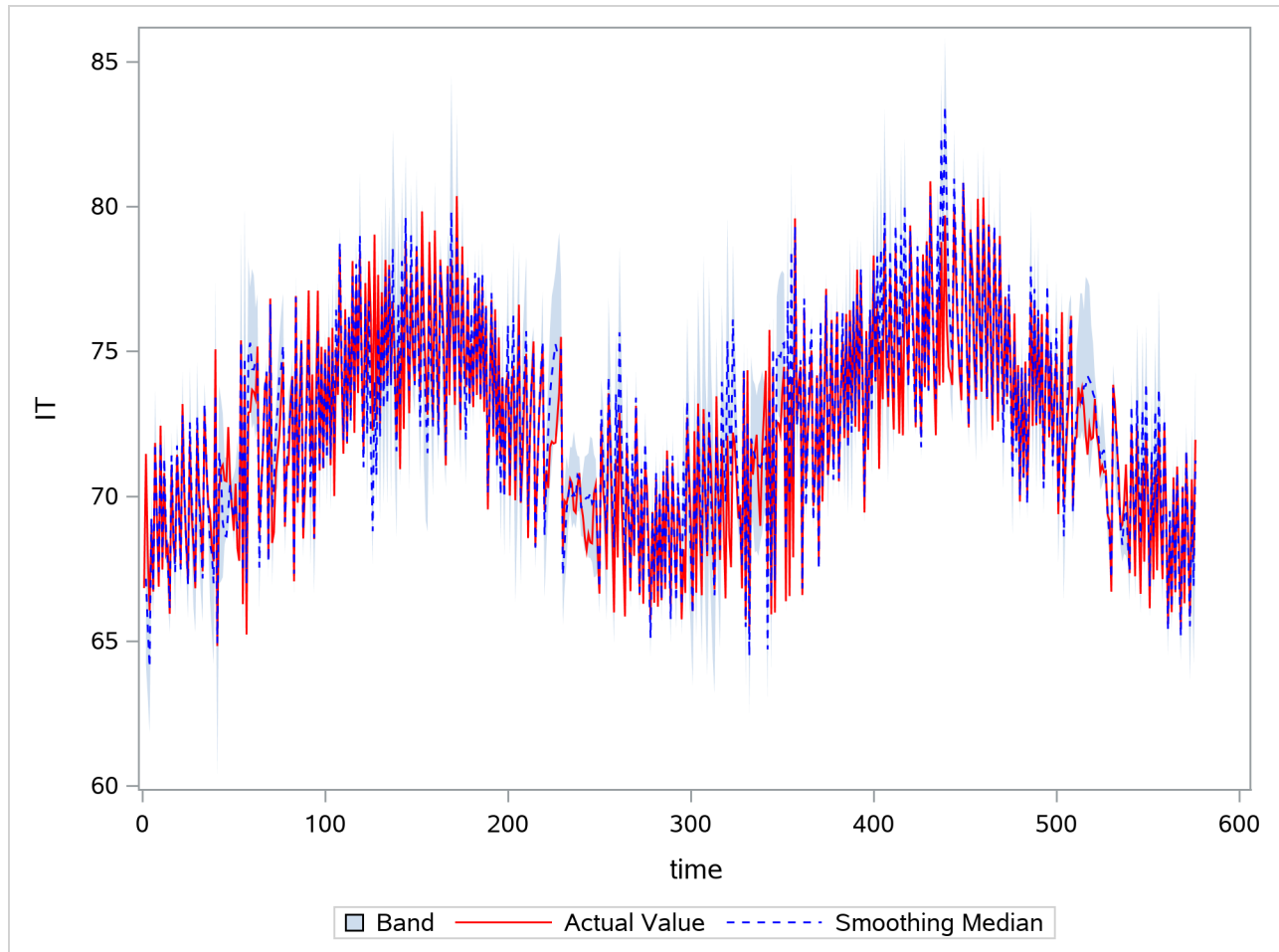


The following statements create a relevant data table, `combine_s4`, in the Work library to record the smoothing estimates and actual values of `InTemp`. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot is shown in [Output 27.4.5](#), where the shadow band indicates the range from the 2.5th to the 97.5th percentiles of the smoothing estimates.

```
data combine_s4;
  set smo_s4;
  set hvacSystem;
run;

proc sgplot data=combine_s4;
  band x=t upper=P_2 lower=P_1 /transparency=.3;
  series x=Time y=IT /legendlabel='Actual Value' lineattrs=(color=red);
  series x=Time y=Q2/ legendlabel='Smoothing Median'
    lineattrs=(color=blue pattern=shortdash);
run;
```

Output 27.4.5 Smoothing Estimates vs. Actual Values, `InTemp`



[Output 27.4.4](#) and [Output 27.4.5](#) show both the outer temperature and inner temperature are accurately estimated by the model.

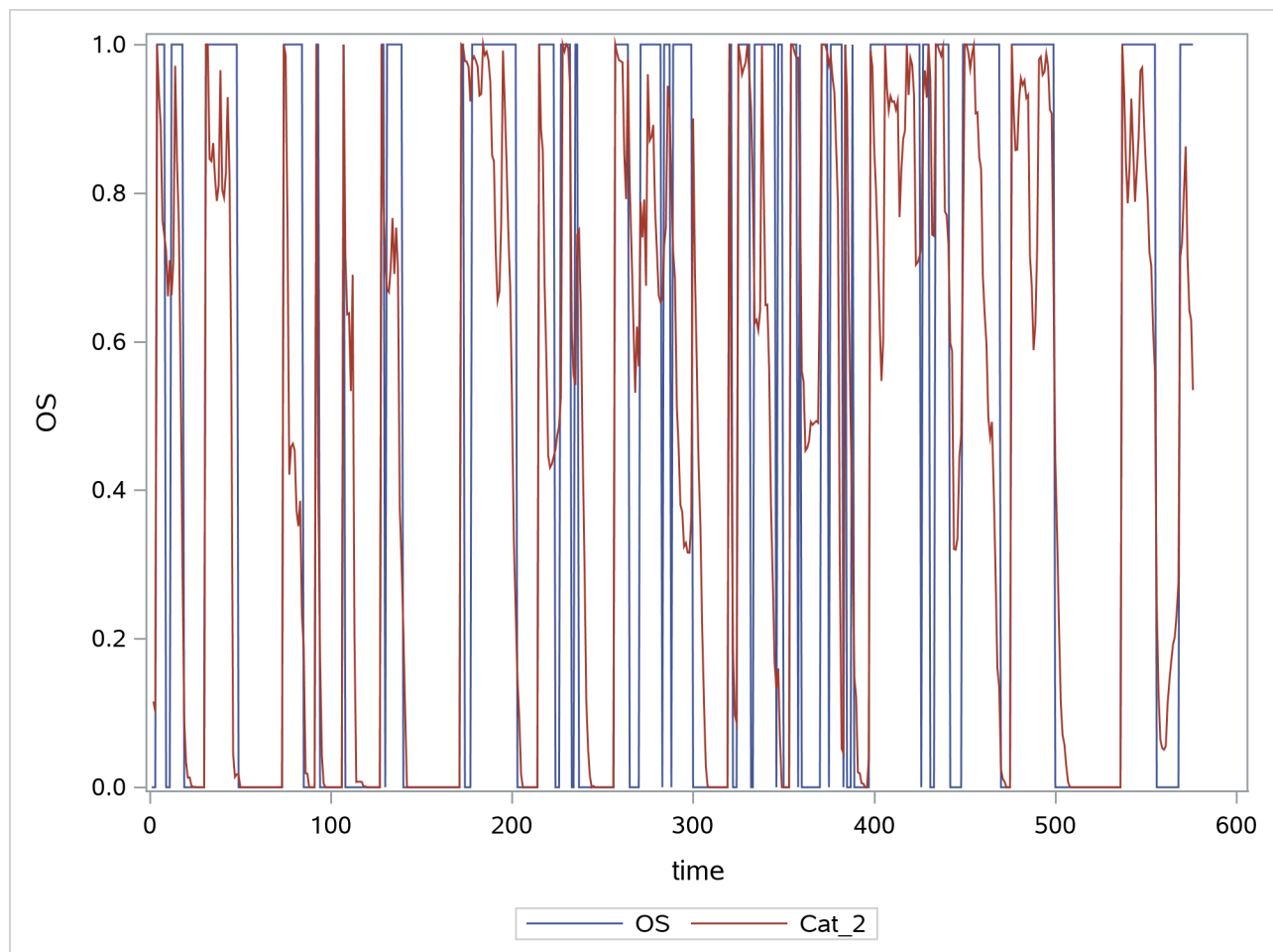
For the category variables, `OutSensorStatus` and `InSensorStatus`, the smoothing estimates of the frequency

of the work status (WORK: 1) are used to compare the actual values. The following statements create a relevant data table, `combine_s1`, in the Work library to record the smoothing estimates and actual values of the category variable `OutSensorStatus`. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot, which is shown in [Output 27.4.6](#), indicates that the hidden outer sensor status is successfully decoded by the model.

```
data combine_s1;
  set smo_s1;
  set hvacSystem;
run;

proc sgplot data=combine_s1;
  series x=Time y=OS;
  series x=Time y=cat_2;
run;
```

Output 27.4.6 Smoothing Estimates vs. Actual Values, `OutSensorStatus`



The following statements create a relevant data table, `combine_s2`, in the Work library to record the smoothing estimates and actual values of the category variable `InSensorStatus`. Then they draw a plot to compare the smoothing estimates to the actual values at each time step. The plot, which is shown in [Output 27.4.7](#), indicates that the hidden inner sensor status is successfully decoded by the model.

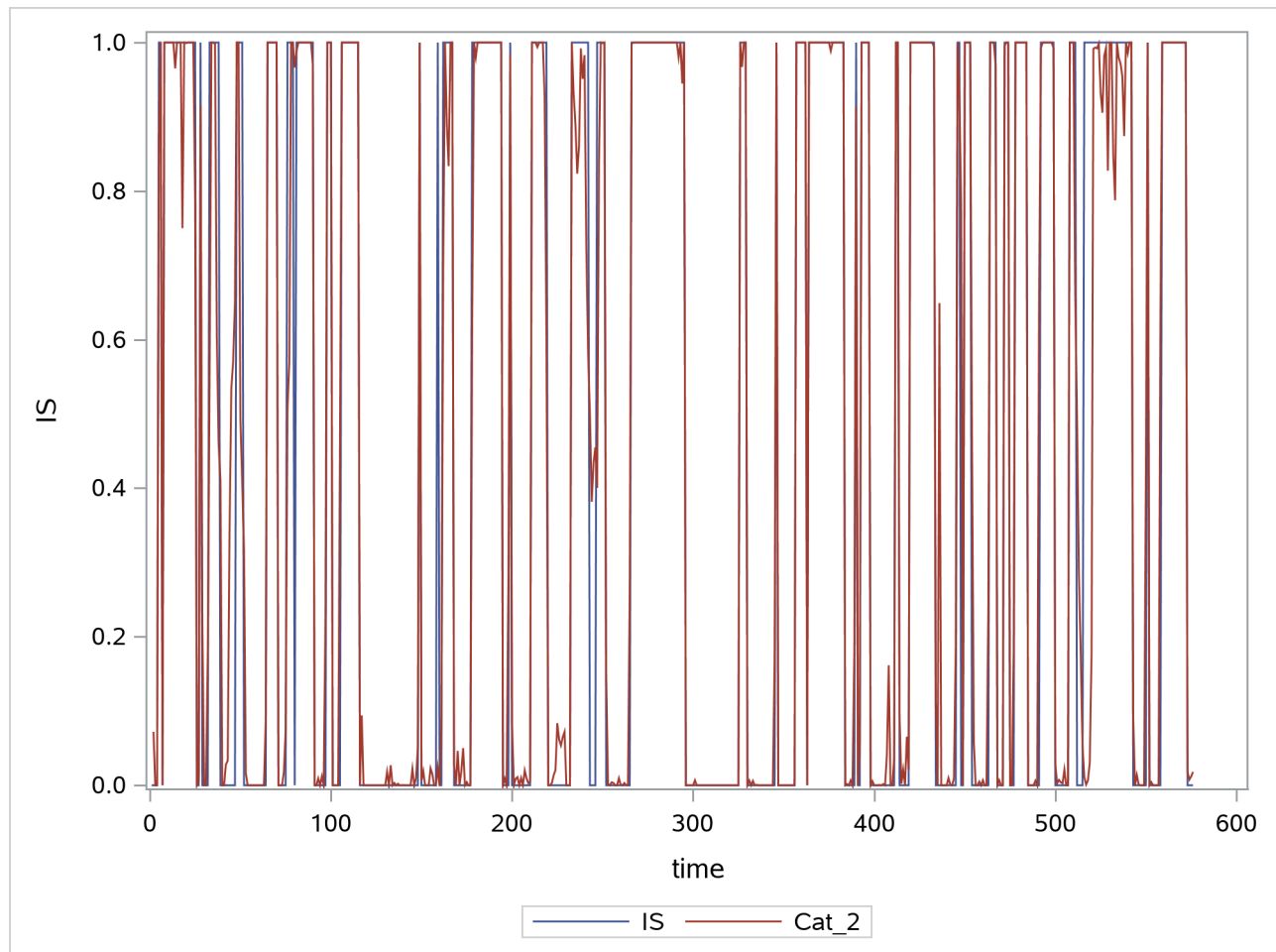
```

data combine_s2;
  set smo_s2;
  set hvacSystem;
run;

proc sgplot data=combine_s2;
  series x=Time y=IS;
  series x=Time y=cat_2;
run;

```

Output 27.4.7 Smoothing Estimates vs. Actual Values, InSensorStatus



Example 27.5: The Fully Adapted Auxiliary Particle Filter Using a Gaussian ARCH Model

This example illustrates the advantages of the fully adapted auxiliary particle filter by using a Gaussian ARCH model.

Consider the following model:

- initialization distribution:

$$x_0 \sim \text{degenerate}(0)$$

- transition distribution:

$$x_t | x_{t-1} \sim \text{normal}(0, \beta_0 + \beta_1 x_{t-1}^2)$$

- measurement distribution:

$$y_t | x_t \sim \text{normal}(x_t, \lambda^2)$$

For each distribution, $\beta_0 > 0$, $\beta_1 > 0$ and $\lambda^2 > 0$.

The fully adapted auxiliary and proposal transition distributions are derived respectively as

$$y_t | x_{t-1} \sim \text{normal}(0, \beta_0 + \beta_1 x_{t-1}^2 + \lambda_t^2)$$

$$x_t | x_{t-1}, y_t \sim \text{normal} \left(\frac{(\beta_0 + \beta_1 x_{t-1}^2) y_t}{\beta_0 + \beta_1 x_{t-1}^2 + \lambda_t^2}, \frac{(\beta_0 + \beta_1 x_{t-1}^2) \lambda_t^2}{\beta_0 + \beta_1 x_{t-1}^2 + \lambda_t^2} \right)$$

First, the decrease in computing time of the APF(ADP) algorithm compared to the APF algorithm is illustrated using the EVALUATE statement. Data sets that have varying numbers of observations are generated. PROC SMC then runs for each data set with varying numbers of particles, and the run time is recorded.

The following statements generate a data set that has a specified number of observations by using the parameter values $\lambda^2 = 0.00027$, $\beta_0 = 0.0000495$, and $\beta_1 = 0.89275$:

```
%let lambda_sq = 0.00027;
%let beta0=0.0000495;
%let beta1=0.89275;
%macro generate_data(nobs=);
  proc iml;
    call randseed(123);
    x = j(&nobs,1,.);
    y = j(&nobs,1,.);
    timeID = j(&nobs,1,.);
    x[1]=0;
    call randgen(rny,"Normal");
    y[1]=x[1]+sqrt(&lambda_sq)*rny;
    timeID[1]=1;
```



```

do t=2 to &nobs;
  call randgen(rnx,"Normal");
  x[t]=sqrt(&beta0+&beta1*x[t-1]*x[t-1])*rnx;
  call randgen(rny,"Normal");
  y[t]=x[t]+sqrt(&lambda_sq)*rny;
  timeID[t]=t;
end;
create arch var {TimeID x y};
append ;
close arch;
quit;
%mend generate_data;

```

The following macro repeatedly generates data sets and runs PROC SMC with the fully adapted auxiliary and proposal distributions by using the APF(ADP) and APF algorithms. The average run time and total run time over the 100 replications of the log-likelihood computation are recorded.

```

%macro runby_npart_nobs(nobs=,N=,alg=,algn=,nb=,);
  %generate_data(nobs=&nobs);
  data mylib.arch;
    set arch;
  run;
  %do r=1 %to 100;
    %let _timer_start = %sysfunc(datetime());
    proc smc data=mylib.arch seed = 123;
      id time = timeID;
      var y;
      statevar x;
      parameters lambda_sq=&lambda_sq beta0=&beta0 beta1=&beta1;
      initstate x ~ degenerate(0) ;
      state x ~ normal(0, sqrt(beta0+beta1*x.ll*x.ll))/proposal=
        normal((beta0+beta1*x.ll*x.ll)*y/(beta0+beta1*x.ll*x.ll+lambda_sq),
          sqrt(lambda_sq*(beta0+beta1*x.ll*x.ll)/
            (beta0+beta1*x.ll*x.ll+lambda_sq)));
      model y ~ normal(x, sqrt(lambda_sq));
      auxmodel y ~ normal(0, sqrt(beta0+beta1*x.ll*x.ll+lambda_sq));
      evaluate nparticle = &N alg=&alg out=mylib.eval;
    run;
    data duration&nb.&r.;
      duration = datetime() - &_timer_start;
    run;
  %end;

  data duration&nb;
    set
      %do r=1 %to 100;
        duration&nb.&r.
      %end;
  ;

  proc univariate data=duration&nb noprint;
    output out= durationsum&algn.&nb. mean=MEAN&algn. sum=SUM&algn.;
  run;

```

```

data durationsum&align.&nb.;
  set durationsum&align.&nb;
  N = &N;
  T = &nobs;
run;
%mend runby_npart_nobs;

```

The previous macro is then run with $T = 200, 500$, and 1000 and with $N = 500, 1000, 2500$, and 5000 :

```

%macro runit;
  %let value_nobs = 200 500 1000;
  %let value_npart = 500 1000 2500 5000;
  %local iter iter_nobs iter_npart next_value_nobs next_value_npart;
  %let iter=1; %let iter_nobs=1; %let iter_npart=1;
  %do %while (%scan(&value_nobs, &iter_nobs) ne );
    %let next_value_nobs = %scan(&value_nobs, &iter_nobs);
    %do %while (%scan(&value_npart, &iter_npart) ne );
      %put &iter_nobs &iter_npart &iter;
      %let next_value_npart = %scan(&value_npart, &iter_npart);
      %runby_npart_nobs (nobs=&next_value_nobs, N=&next_value_npart,
        alg=%str(APF(ADP)), align=apf_adp, nb=&iter);
      %runby_npart_nobs (nobs=&next_value_nobs, N=&next_value_npart,
        alg=APF, align=apf, nb=&iter);
      data durationallsum&iter;
        merge durationsumapf_adp&iter. (in=a) durationsumapf&iter. (in=b);
        by T N;
        if (a=b);
      run;
      %if &iter =1 %then %do;
        %let dataread=durationallsum&iter;
      %end;
      %else %do;
        %let dataread= durationallsum durationallsum&iter;
      %end;
      data durationallsum;
        set &dataread;
      run;
      %let iter_npart = %eval(&iter_npart + 1);
      %let iter = %eval(&iter + 1);
    %end;
    %let iter_nobs = %eval(&iter_nobs + 1);
    %let iter_npart=1;
  %end;
%mend runit;
%runit;

```

The following statements plot the average log-likelihood computation run time over 100 replications. The results are shown in [Figure 27.12](#).

```

proc sgpanel data=durationallsum;
  panelby T / layout=panel columns=1 rows=3 uniscale=column;
  series x=N y=MEANapf/markers markerattrs=(color=blue size=7 symbol=Diamond)
    legendlabel='APF';
  series x=N y=MEANapf_adp/markers markerattrs=(color=red size=7 symbol=asterisk)

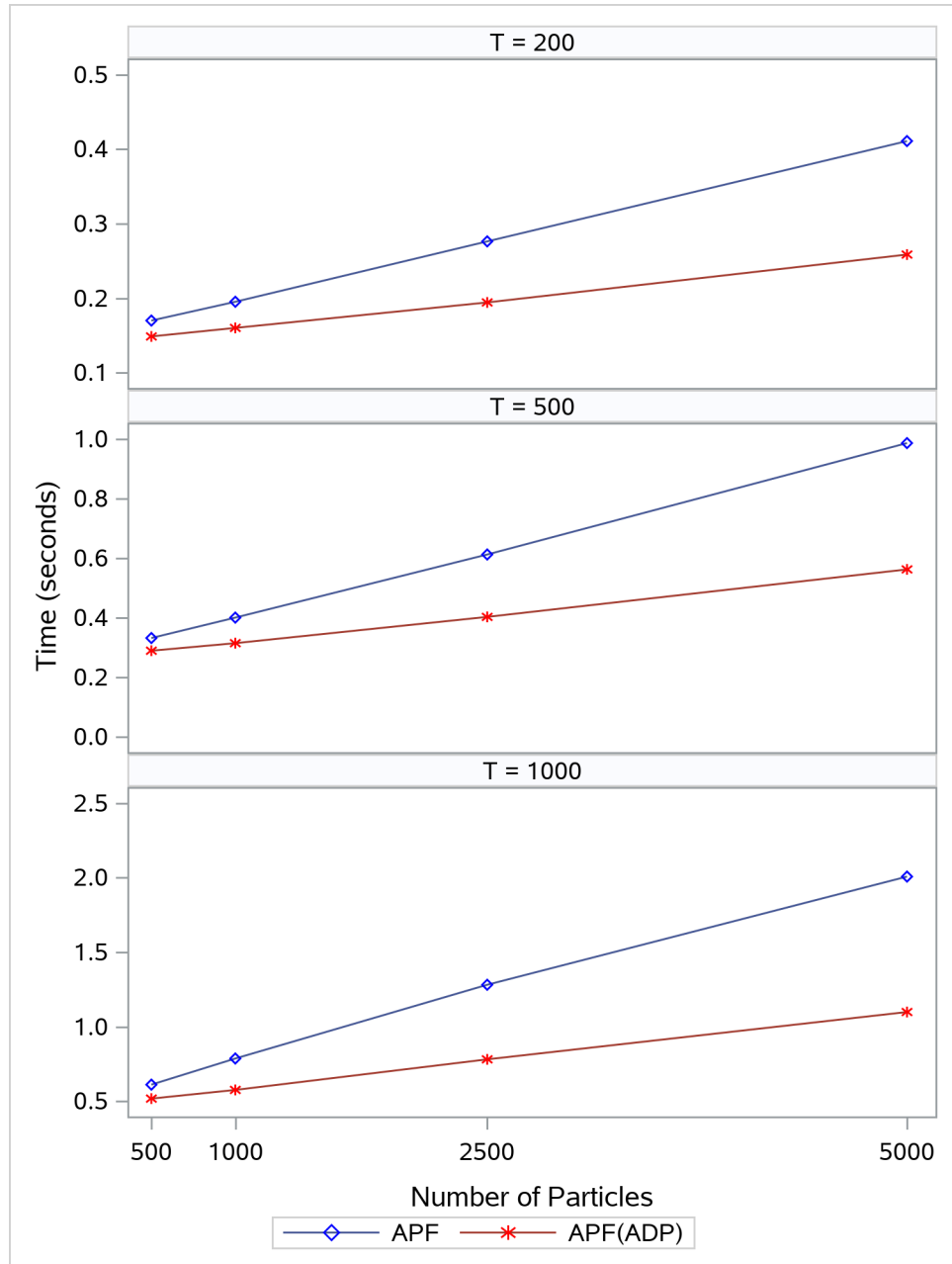
```

```

legendlabel='APF (ADP) ' ;
rowaxis thresholdmin=1 thresholdmax=1 label='Time (seconds) ' ;
colaxis label='Number of Particles'
valueshint values=(500 1000 2500 5000) ;
run;

```

Figure 27.12 Average Log-Likelihood Computation Run Time over 100 Replications



The following statements plot the total log-likelihood computation run time over 100 replications by using a single computing node. The results are shown in [Figure 27.13](#).

```

proc sgpanel data=durationallsum;
  panelby T / layout=panel columns=1 rows=3 uniscale=column;
  series x=N y=SUMapf/markers markerattrs=(color=blue size=7 symbol=Diamond)
    legendlabel='APF';
  series x=N y=SUMapf_adp/markers markerattrs=(color=red size=7 symbol=asterisk)
    legendlabel='APF (ADP)';
  rowaxis thresholdmin=1 thresholdmax=1 label='Time (seconds)';
  colaxis label='Number of Particles'
    valueshint values=(500 1000 2500 5000);
run;

```

Figure 27.13 Total Log-Likelihood Computation Run Time over 100 Replications

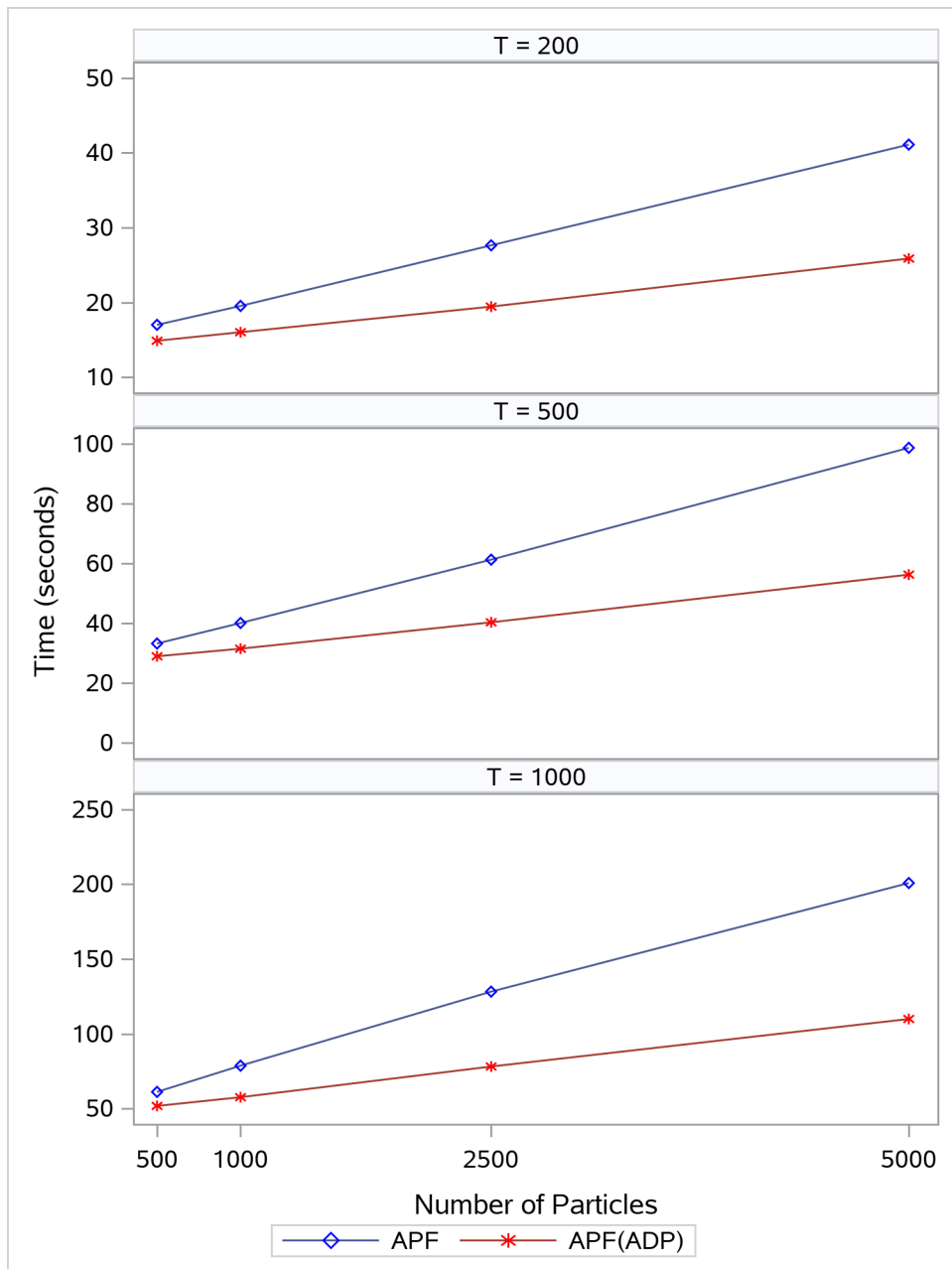


Figure 27.12 and Figure 27.13 show that the APF(ADP) algorithm is faster than the APF algorithm with fully adapted auxiliary and proposal distributions, especially when the number of particles and/or the number of time points is large. A similar conclusion can be made about the FILTER, FORECAST, LEARN, and SMOOTH statements.

In addition to being faster, the APF(ADP) algorithm results in a smaller variance for the estimated log likelihood and a smaller optimal number of particles compared to APF and BF. The next code aims to illustrate this.

The following statements simulate 500 observations on the basis of the defined ARCH model and load the data set arch into a data table in your session that is associated with the mylib libref:

```
%let lambda_sq = 0.00027;
%let beta0=0.0000495;
%let beta1=0.89275;
proc iml;
  call randseed(123);
  x = j(500,1,.);
  y = j(500,1,.);
  timeID = j(500,1,.);
  x[1]=1;
  call randgen(rny,"Normal");
  y[1]=x[1]+sqrt(&lambda_sq)*rny;
  timeID[1]=1;
  do t=2 to 500;
    call randgen(rnx,"Normal");
    x[t]=sqrt(&beta0+&beta1*x[t-1]*x[t-1])*rnx;
    call randgen(rny,"Normal");
    y[t]=x[t]+sqrt(&lambda_sq)*rny;
    timeID[t]=t;
  end;
  create arch var {TimeID x y};
  append ;
  close arch;
quit;

data mylib.arch;
  set arch;
run;
```

Following the guideline for choosing the optimal number of particles provided by Pitt et al. (2012), the next group of statements estimate the log likelihood 100 times for a given number of particles and filtering algorithm, and then they compute the standard deviation of the log likelihood over the 100 replications:

```
%macro repeat_run(npart=,alg=,algn=,nb=);
  %do i=1 %to 100;
    proc smc data=mylib.arch seed = &i;
      id time = timeID;
      var y;
      statevar x;
      parameters lambda_sq=&lambda_sq beta0=&beta0 beta1=&beta1;
      initstate x ~ degenerate(1) ;
      state x ~ normal(0, sqrt(beta0+beta1*x.l1*x.l1))
    end;
  end;
```

```

        %if %length(&alg) gt 3 %then /proposal=
        normal((beta0+beta1*x.ll*x.ll)*y/(beta0+beta1*x.ll*x.ll+lambda_sq),
        sqrt(lambda_sq*(beta0+beta1*x.ll*x.ll)/(beta0+beta1*x.ll*x.ll+lambda_sq)));;
model y ~ normal(x, sqrt(lambda_sq));
auxmodel y ~ normal(0, sqrt(beta0+beta1*x.ll*x.ll+lambda_sq));
evaluate nparticle = &npart alg=&alg out=mylib.eval&nb.&i.;
run;
data eval&nb.&i.(keep = LogLikelihood);
    set mylib.eval&nb.&i. end=eof;
    if eof=1;
run;
%end;
data evalall&nb;
    set
    %do i=1 %to 100;
        eval&nb.&i.
    %end;
;

proc univariate data= evalall&nb vardef=n noprint;
    output out= evalallsum&algn.&nb. mean=MEAN std=STDDEV var=VARIANCE n=R;
run;

data evalallsum&algn.&nb.;
    set evalallsum&algn.&nb(drop=R);
    N = &npart;
    algorithm = "&alg";
run;
%mend repeat_run;

```

The following statements run the macro with $N = 15, 20, 25, 50, 75, 90, 100, 200, 300, 400$, and 500 particles for the APF(ADP), APF, and BF algorithms:

```

%macro runit2;
    %let value = 15 20 25 50 75 90 100 200 300 400 500;
    %local iter next_value;
    %let iter=1;
    %do %while (%scan(&value, &iter) ne );
        %let next_value = %scan(&value, &iter);
        %repeat_run(npart=&next_value, alg=%str(APF(ADP)), algn=1, nb=&iter);
        %repeat_run(npart=&next_value, alg=APF, algn=2, nb=&iter);
        %repeat_run(npart=&next_value, alg=BF, algn=3, nb=&iter);
        data evalallsum&iter;
            set evalallsum1&iter evalallsum2&iter evalallsum3&iter;
        run;
        %if &iter =1 %then %do;
            %let dataread=evalallsum&iter;
        %end;
        %else %do;
            %let dataread= evalallsum evalallsum&iter;
        %end;

        data evalallsum;
            set &dataread;
    %end;
%macro runit2;

```

```

run;
%let iter = %eval(&iter + 1);
%end;
%mend runit2;
%runit2;

```

The log-likelihood standard deviation across the different number of particles and filtering algorithms is then plotted using the following code. The results are shown in [Figure 27.14](#). Following Pitt et al. (2012), a reference line corresponding to a standard deviation of 0.92 is added to help determine the optimal number of particles.

```

data evalallsum;
  set evalallsum;
  ref = 0.92;
run;

proc sort data=evalallsum;
  by algorithm N;
run;

proc sgpanel data=evalallsum;
  title ;
  panelby algorithm / layout=panel columns=3 rows=1 uniscale=all;
  series x=N y=STDDEV/markers markerattrs=(color=blue size=7 symbol=plus)
         legendlabel='StdDev ';
  series x=N y=ref / legendlabel='Cutoff';
  rowaxis thresholdmin=1 thresholdmax=1 label='Standard Deviation' max=25;
  colaxis label='Number of Particles'
         valueshint
         values=(0 15 20 25 50 75 90 100 200 300 400 500);
  keylegend/position=bottom;
run;

```

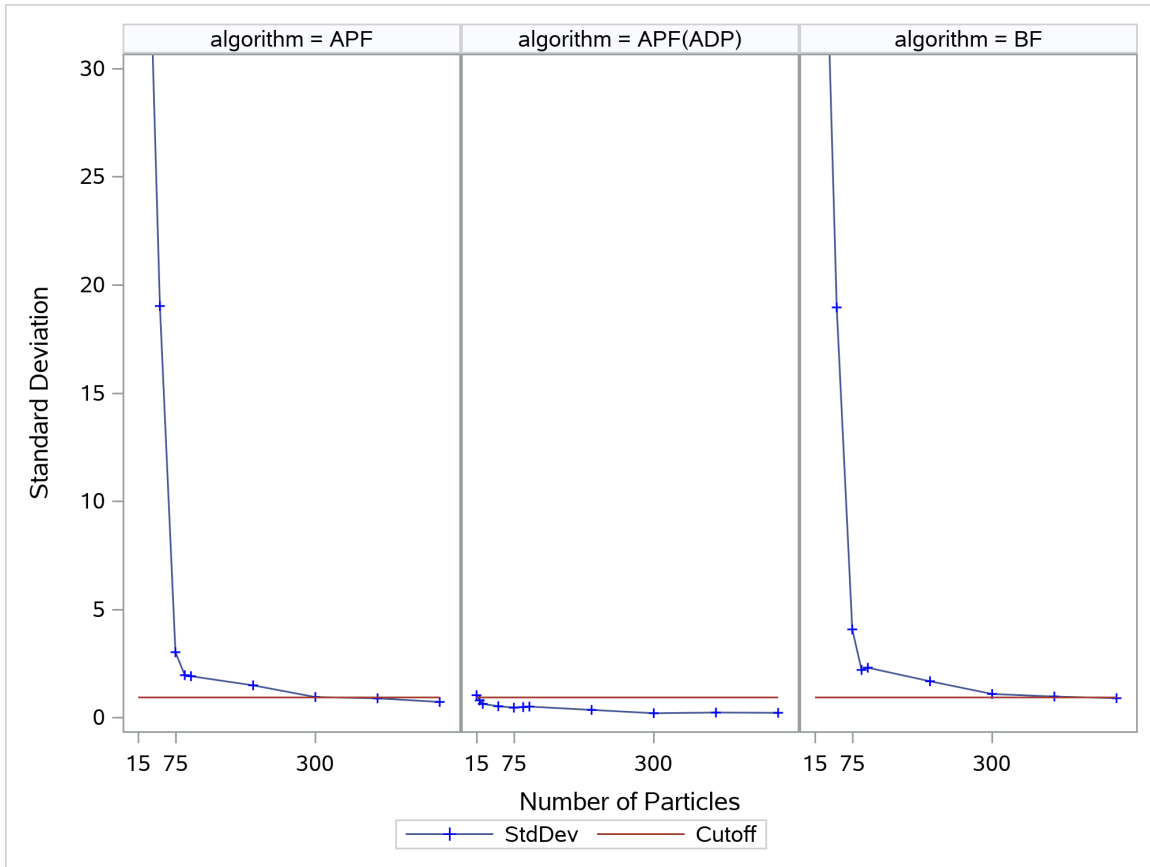
Figure 27.14 Log-Likelihood Variability and Optimal Number of Particles

Figure 27.14 shows that the variability in the log likelihood is significantly smaller for APF(ADP) than for APF and BF. Moreover, the optimal numbers of particles that are determined as the number of particles corresponding to a standard deviation of 0.92 (intersection of the red reference line and standard deviation line) are 20, 450, and 450 for APF(ADP), APF, and BF, respectively. Because APF(ADP) requires fewer particles in order to reach the optimal standard deviation, this further improves the speed of computation.

A final advantage of APF(ADP) over APF and BF is its lower Markov chain autocorrelation. To illustrate this, particle Markov chain Monte Carlo (MCMC) is run using the arch data set and the APF(ADP), APF, and BF algorithms.

The parameters β_0 and β_1 are transformed so that the new parameters θ_0 and θ_1 are unconstrained: $\theta_0 = \log(\beta_0)$ and $\theta_1 = \log(\beta_1)$. The following prior distributions are specified:

$$\theta_0 \sim \text{normal}(-10, 4)$$

$$\theta_1 \sim \text{normal}(-0.13, 2.8)$$

$$\lambda^2 \sim \text{inverse gamma}(0.01, 0.01)$$

The following macro runs particle MCMC and saves the acceptance rate and diagnostics summary tables for a given filtering algorithm and number of particles:


```

%macro run_learn(npart=, alg=, algn=);
  ods output DiagSummaries= DiagSummary_&algn.
    AccRates=AccRates_&algn.;
  proc smc data=mylib.arch seed =123;
    id time = timeID;
    var y;
    statevar x;
    parameters lambda_sq=0.005 theta0=-6 theta1=-0.3;
    initstate x ~ degenerate(1) ;
    state x ~ normal(0, sqrt(exp(theta0)+exp(theta1)*x.l1*x.l1))
      %if %length(&alg) gt 3 %then /proposal=
        normal((exp(theta0)+exp(theta1)*x.l1*x.l1)*y/(exp(theta0)
          +exp(theta1)*x.l1*x.l1+lambda_sq),
        sqrt(lambda_sq*(exp(theta0)+exp(theta1)*x.l1*x.l1)/(exp(theta0)+
          exp(theta1)*x.l1*x.l1+lambda_sq)));;
    model y ~ normal(x, sqrt(lambda_sq));
    auxmodel y ~ normal(0, sqrt(exp(theta0)+exp(theta1)*x.l1*x.l1+lambda_sq));
    prior lambda_sq ~ igamma(0.01,0.01);
    prior theta0 ~ normal(-10,4);
    prior theta1 ~ normal(-0.13,2.8);
    learn nparticle = &npart algorithm=&alg
      method = PMCMC(nsampl = 50000 nbi=10000 samsum samset seed = 1
        thin=1 outpost = mylib.outle_&algn.
        statistics=(corr cov interval(alpha = (0.01))
        summary(perc = (0.5 2.5 25 50 75 97.5 99)))
        diagnostics(settings)=all sampler = rwm(ntu=700 ntunestage=48)) ;
  run;

  data AccRates_&algn.;
    set AccRates_&algn.;
    Algorithm = "&alg";
    N = &npart;
  run;
%mend run_learn;

```

The macro is then run for the APF(ADP), APF, and BF algorithms by using the optimal number of particles determined previously. A computing time variable is defined as (Number of particles) * (Autocorrelation time)/1000.

```

%run_learn(npart=20,alg=%str(APF(ADP)),algn=apf_adp);
%run_learn(npart=450,alg=APF,algn=apf);
%run_learn(npart=450,alg=BF,algn=bf);

data AccRates;
  set AccRates_apf_adp AccRates_bf AccRates_apf;
run;

data DiagSummary_bf;
  set DiagSummary_bf;
  ComputingTime = 450*CorrTime/1000;
run;
proc transpose data=DiagSummary_bf out=DiagSummary_bf_w;
  id Parameter;

```

```

run;
data DiagSummary_bf_w; set DiagSummary_bf_w;
  order=2; Algorithm='BF'; N = 450;run;

data DiagSummary_apf;
  set DiagSummary_apf;
  ComputingTime = 450*CorrTime/1000;
run;
proc transpose data=DiagSummary_apf out=DiagSummary_apf_w;
  id Parameter;
run;
data DiagSummary_apf_w; set DiagSummary_apf_w;
  order=1; Algorithm='APF'; N = 450;run;

data DiagSummary_fapf;
  set DiagSummary_fapf;
  ComputingTime = 20*CorrTime/1000;
run;
proc transpose data=DiagSummary_fapf out=DiagSummary_fapf_w;
  id Parameter;
run;
data DiagSummary_fapf_w; set DiagSummary_fapf_w;
  order=3; Algorithm='APF(ADP)'; N = 20;run;

data DiagSummary;
  set DiagSummary_fapf_w DiagSummary_bf_w DiagSummary_apf_w;
run;

```

The following statements print the acceptance rates for each algorithm, as shown in [Output 27.5.1](#). The acceptance rates of the three algorithms are around the 0.234 optimal value for the random walk Metropolis sampler, despite APF(ADP) using far fewer particles.

```

proc sort data=AccRates; by descending N;run;
proc print data=AccRates noobs;
  var Algorithm N AccRate;
run;

```

Output 27.5.1 Acceptance Rates

Algorithm	N	AccRate
BF	450	0.260185
APF	450	0.253605
APF(ADP)	20	0.208684

The following statements print the autocorrelation time and the indicator variable that demonstrates whether autocorrelation remains in the chain, as shown in [Output 27.5.2](#) and [Output 27.5.3](#), respectively:

```

data CorrTime;
  set DiagSummary;
  where _NAME_ = 'CorrTime';
run;
proc sort data=CorrTime; by order;run;
proc print data=CorrTime noobs;
  var Algorithm N lambda_sq theta0 theta1;

```

```

run;
data AutoCorrRemains;
  set DiagSummary;
  where _NAME_ = 'AutoCorrRemains';
run;
proc sort data=AutoCorrRemains; by order;run;
proc print data=AutoCorrRemains noobs;
  var Algorithm N lambda_sq theta0 theta1;
run;

```

Output 27.5.2 Autocorrelation Time

Algorithm	N	lambda_sq	theta0	theta1
APF	450	6.3872	478.600	496.600
BF	450	6.1982	491.500	488.600
APF(ADP)	20	29.9046	43.622	45.618

Output 27.5.3 Autocorrelation Remains

Algorithm	N	lambda_sq	theta0	theta1
APF	450	0	1	1
BF	450	0	1	1
APF(ADP)	20	0	0	0

Output 27.5.2 and Output 27.5.3 show that significant autocorrelation still remains in the Markov chain for θ_0 and θ_1 when the APF and BF algorithms are used, indicating that you need to run the Markov chain longer or increase the tuning time of the proposal distribution.

Finally, the following statements print the computing time, which is shown in Output 27.5.4:

```

data ComputingTime;
  set DiagSummary;
  where _NAME_ = 'ComputingTime';
run;
proc sort data=ComputingTime; by order;run;
proc print data=ComputingTime noobs;
  var Algorithm N lambda_sq theta0 theta1;
run;

```

Output 27.5.4 Computing Time

Algorithm	N	lambda_sq	theta0	theta1
APF	450	2.87424	215.370	223.470
BF	450	2.78919	221.175	219.870
APF(ADP)	20	0.59809	0.872	0.912

Output 27.5.4 shows that APF(ADP) has a lower computing time, defined as (Number of particles) * (Autocorrelation time)/1000, than APF and BF for all the parameters in the model. Clearly, the APF(ADP) algorithm with 20 particles is more efficient than the APF or BF algorithm with 450 particles.

References

- Andrieu, C., Doucet, A., and Holenstein, R. (2010). “Particle Markov Chain Monte Carlo Methods.” *Journal of the Royal Statistical Society, Series B* 72:269–342.
- Cappé, O., Godsill, S. J., and Moulines, E. (2007). “An Overview of Existing Methods and Recent Advances in Sequential Monte Carlo.” *Proceedings of the IEEE* 95:899–924.
- Dong, E., Du, H., and Gardner, L. (2020). “An Interactive Web-Based Dashboard to Track COVID-19 in Real Time.” *Lancet Infectious Diseases* 20:533–534.
- Doucet, A., de Freitas, N., and Gordon, N. (2001). “An Introduction to Sequential Monte Carlo Methods.” In *Sequential Monte Carlo Methods in Practice*, 3–14. New York: Springer.
- Doucet, A., Godsill, S., and Andrieu, C. (2000). “On Sequential Monte Carlo Sampling Methods for Bayesian Filtering.” *Statistics and Computing* 10:197–208.
- Doucet, A., and Johansen, A. M. (2011). “A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later.” In *Oxford Handbook of Nonlinear Filtering*, 656–704. Oxford: Oxford University Press.
- Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). “Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation.” *IEE Proceedings F (Radar and Signal Processing)* 140:107–113.
- Kitagawa, G. (1996). “Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models.” *Journal of Computational and Graphical Statistics* 5:1–25.
- Kitagawa, G., and Sato, S. (2001). “Monte Carlo Smoothing and Self-Organising State-Space Model.” In *Sequential Monte Carlo Methods in Practice*, 177–195. New York: Springer.
- Liu, J. S., and Chen, R. (1995). “Blind Deconvolution via Sequential Imputations.” *Journal of the American Statistical Association* 90:567–576.
- Pitt, M. K., and Shephard, N. (1999). “Filtering via Simulation: Auxiliary Particle Filters.” *Journal of the American Statistical Association* 94:590–599.
- Pitt, M. K., Silva, R. S., Giordani, P., and Kohn, R. (2012). “On Some Properties of Markov Chain Monte Carlo Simulation Methods Based on the Particle Filter.” *Journal of Econometrics* 171:134–151.
- Ristic, B., Arulampalam, S., and Gordon, N. (2004). *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Boston: Artech House.
- Rubin, D. (1988). “Using the SIR Algorithm to Simulate Posterior Distributions.” *Bayesian Statistics* 3:395–402.
- Shephard, N., ed. (2005). *Stochastic Volatility: Selected Readings*. Oxford: Oxford University Press.
- Taylor, S. (1982). “Financial Returns Modelled by the Product of Two Stochastic Processes: A Study of the Daily Sugar Prices, 1961–75.” *Time Series Analysis: Theory and Practice* 1:203–226.

Chapter 28

TSINFO Procedure

Contents

Overview: TSINFO Procedure	2059
PROC TSINFO Compared with the TIMEID Procedure	2060
Using CAS Sessions and CAS Engine Librefs	2060
Getting Started: TSINFO Procedure	2061
Syntax: TSINFO Procedure	2062
Functional Summary	2062
PROC TSINFO Statement	2062
BY Statement	2063
ID Statement	2063
Details: TSINFO Procedure	2064
Time ID Diagnostics	2064
Inferring Time Intervals and Alignments	2064
Printed Output	2065
Examples: TSINFO Procedure	2066
Example 28.1: Examining a Monthly Time ID Variable and Inferring a Date Interval	2066
Example 28.2: Examining Multiple BY Groups	2066

Overview: TSINFO Procedure

The TSINFO procedure evaluates a variable in an input data table for its suitability as a time ID variable in SAS procedures and solutions that are used for time series analysis. PROC TSINFO assesses how well a time interval specification fits SAS date values, SAS datetime values, or observation numbers that are used to index a time series. The time interval used in this analysis can be either specified explicitly as input or inferred based on values of the time ID variable. The TSINFO procedure produces diagnostic information in the form of data tables and ODS tables. These diagnostic results summarize characteristics of the time ID variable that can help determine its use as an index in other time series procedures and solutions.

PROC TSINFO is intended for use as a tool to either identify the time interval of a variable or prepare problematic data sets for use in subsequent time series analyses. In particular, this procedure can be used to investigate inconsistencies between time ID values and the ID statement options that are used in other SAS procedures and solutions.

PROC TSINFO Compared with the TIMEID Procedure

The TSINFO procedure is the next generation of the TIMEID procedure (a SAS/ETS procedure) for time series information analysis. PROC TSINFO was developed specifically for SAS Viya. The syntax of PROC TSINFO is similar to the syntax of PROC TIMEID, from which it borrows the underlying methodology and goals. PROC TSINFO is designed to run on a cluster of machines that distribute the data and computations to multiple threads, and it requires your data tables to be available on a SAS Cloud Analytic Services (CAS) server.

Table 28.1 shows the differences between the TSINFO and TIMEID procedures.

Table 28.1 PROC TSINFO versus PROC TIMEID

Feature	TSINFO	TIMEID
Threading	Multithreaded	Single-threaded
Input data	CAS tables	SAS data sets
Requires sorted input data?	No	Yes, by BY variables and ID variable bottom-up

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the `CASHOST` and `CASPORT` of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the `TERMINATE` option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Getting Started: TSINFO Procedure

When a data table contains a time ID variable that has corrupted, missing, or duplicate values, PROC TSINFO can help isolate and identify these problematic observations. For larger data tables whose quality is unknown, it can be useful to get a general overview of the relative number of observations that have problematic time ID values. When prior knowledge of the time interval that separates observations is incomplete, PROC TSINFO can be used to infer the interval.

The following example uses the Sashelp.Air data set to illustrate how to analyze some time series information. The following DATA step loads the Air data set from the Sashelp directory to a table named mylib.air in your CAS session. This DATA step assumes that your CAS engine libref is named mylib, but you can substitute any appropriately defined CAS engine libref.

```
data mylib.air(replace=yes);
  set sashelp.air;
  do class=1 to 10;
    output;
  end;
run;
```

The following statements use the TSINFO procedure to infer the interval of the time series:

```
proc tsinfo data=mylib.air outintervaldetails=mylib.interval1
  nthreads=2;
  id date align=BEGIN;
  by class;
run;
```

PROC TSINFO requires the following inputs:

- The input table, specified in the DATA= option in the PROC statement
- BY variables (class), specified in the BY statement
- The variable that contains the timestamps in the ID statement

Syntax: TSINFO Procedure

The following statements are available in the TSINFO procedure:

```
PROC TSINFO < options > ;
BY variables ;
ID variable < options > ;
```

The PROC TSINFO and ID statements are required. The following sections describe the PROC TSINFO statement and then describe the other statements in alphabetical order.

Functional Summary

Table 28.2 summarizes the statements and options that are used with the TSINFO procedure.

Table 28.2 PROC TSINFO Functional Summary

Description	Statement	Option
Data Options		
Specifies the input data table	PROC TSINFO	DATA=
Specifies the detailed output interval data table	PROC TSINFO	OUTINTERVALDETAILS=
Specifies the number of threads to be used in the procedure	PROC TSINFO	NTHREADS=
Time ID Options		
Specifies the alignment of time ID values	ID	ALIGN=
Specifies the ending time ID value	ID	END=
Specifies the starting time ID value	ID	START=

PROC TSINFO Statement

```
PROC TSINFO options ;
```

The PROC TSINFO statement invokes the TSINFO procedure.

You must specify the following data tables:

DATA=*libref.data-table*

specifies the SAS input data table that contains the input data for the procedure. If the DATA= option is not specified, the most recently created SAS data set is used. *libref.data-set* is a two-level name, where *libref* refers to the library and session identifier, and *data-table* specifies the name of the input data table.

OUTINTERVALDETAILS =libref.data-table

names the output data table to contain the time ID interval information for each BY group in the DATA= data set. For more information, see the section “[OUTINTERVALDETAILS= Data Set](#)” on page 2065. *libref.data-table* is a two-level name, where *libref* refers to the library and session identifier, and *data-table* specifies the name of the input data table.

You can also specify the following *option*:

NTHREADS=number

specifies the number of threads to use in the computation.

By default, NTHREADS=1.

BY Statement

BY variables ;

You can use a BY statement to obtain separate analyses for groups of observations that are defined by the *variables*, which can be either character or numeric.

When a BY statement is included, the procedure expects the input data to be sorted in the order of the BY variables.

ID Statement

ID variable </options> ;

The ID statement names a numeric *variable* that identifies observations in the input and output data. The ID variable values are assumed to be SAS date or datetime values.

You can specify the following *options* to determine how to space the time ID values and align them relative to a SAS date or datetime interval:

ALIGN=BEGINNING | MIDDLE | ENDING | INFER

specifies the alignment of the identifying SAS date or datetime that is used to represent intervals. The value of the ALIGN= option is used in the analysis of the time ID variable. You can specify the following values:

BEGINNING BEG B	uses the beginning date in the interval as the identifying date for the interval.
MIDDLE MID M	uses the middle date in the interval as the identifying date for the interval.
ENDING END E	uses the end date in the interval as the identifying date for the interval.
INFER	infers that the alignment of values within time intervals from the time ID values.

By default, ALIGN=BEGIN.

END=*value*

specifies a SAS date, datetime, or time value that represents when the SAS date or datetime values end. If the largest *variable* value is less than *value*, then this option has no effect.

START=*value*

specifies a SAS date, datetime, or time value that represents the *variable* value at which the SAS date or datetime values begins.

Details: TSINFO Procedure

Time ID Diagnostics

For a specified time interval, PROC TSINFO decomposes the raw time ID values in an input data table into the following three quantities, whose values are represented by nonnegative integers at each unique time ID value in the input series:

intervalcounts	the number of observations that share each time interval in the data set.
offsets	the numerical difference between a time ID value and the aligned value for that time interval. The unit of measure that is used to express this distance is days for date values and seconds for datetime values. The offset is computed for each time ID value, t_i , by using the following SAS expression:

$$\text{offset}_i = t_i - \text{INTNX}(\text{interval}, t_i, 0, \text{alignment})$$

spans	the number of intervals between each time ID value and the previous time ID value. The spans value is equivalent to the number returned by the following SAS expression:
-------	--

$$\text{spans}_i = \text{INTCK}(\text{interval}, t_{i-1}, t_i)$$

Inferring Time Intervals and Alignments

A time interval is inferred from the time ID values in the input data. The technique that is used to infer a time interval involves searching for the interval that fits the greatest number of time ID values. First, time ID values are sampled from the input data to generate a set of candidate intervals. Then the candidate interval that is consistent with greatest number of time ID values is chosen to represent the time series.

When the **ALIGN=INFER** option is specified, the convention that is used to specify time interval alignment is inferred from the time ID variable values by using a similar technique. When both the time interval and its alignment are to be inferred, each of the possible alignments (**BEGIN**, **MIDDLE**, and **END**) is considered in the search. Precedence in the search is given to intervals that use the **BEGIN** alignment.

Data Table Output

The TSINFO procedure creates an ODS output data table. This data table contains the variables that are specified in the BY statement along with variables that characterize the time ID values. The information in this data table summarizes time ID diagnostic information across all BY groups in the table that is specified in the DATA= option.

OUTINTERVALDETAILS= Data Set

The OUTINTERVALDETAILS= data set contains statistics about the time interval that is specified in the ID statement or inferred from the time ID values for each BY group. The following variables represent these statistics:

TIMEID	time ID variable name
START	starting time ID interval
END	ending time ID interval
INTERVAL	time interval that was specified or is recommended
INTNAME	time interval base name that was specified or is recommended
MULTIPLIER	time interval multiplier that was specified or is recommended
SHIFT_INDEX	time interval shift index that was specified or is recommended
ALIGNMENT	time interval alignment that was specified or is recommended
SEASONALITY	seasonality determined from the specified or recommended time interval
FORMAT	format of the time ID variable

The START and END variables are reported using the interval and alignment values that are either specified in the ID statement or inferred from the time ID values.

Printed Output

The TSINFO procedure always prints a summary of the processing that is performed on the time series data. The following variables represent these statistics:

Interval	inferred interval
Count	number of times inferred
Conditional	number of times conditional
Alignment	inferred alignment
Nseason	number of season cycles

Examples: TSINFO Procedure

Example 28.1: Examining a Monthly Time ID Variable and Inferring a Date Interval

This example illustrates how a time ID variable can be inferred from data when a sufficient number of observations are present.

The following DATA step loads the Air data set onto the CAS server. These statements assume that your CAS engine libref is named mylib, but you can substitute any appropriately defined CAS engine libref.

```
data mylib.air(replace=yes);  
    set sashelp.air;  
run;
```

The following TSINFO procedure statements generate an ODS display of the time series that characterizes interval counts, alignment, and seasonality in the time ID variable:

```
proc tsinfo data=mylib.air outintervaldetails=mylib.interval2 nthreads=2;  
    id date align=B;  
run;
```

There are 144 observations in the mylib.Air data table. The 144 observations are enough to determine that the Date time ID variable is represented by the MONTH interval.

Example 28.2: Examining Multiple BY Groups

This example illustrates how a time ID variable can be examined independently over each BY group and summarized over all observations in the data table that is specified in the DATA= option.

The following DATA step creates the BY-group variable, class, and loads the Air data set onto the CAS server:

```
data mylib.air(replace=yes);  
    set sashelp.air;  
    do class=1 to 10;  
        output;  
    end;  
run;
```

The following TSINFO procedure statements generate data tables that summarize data that have 10 BY groups:

```
proc tsinfo data=mylib.air outintervaldetails=mylib.interval2 nthreads=2;  
    id date align=B;  
    by class;  
run;
```

The summarized information shows that BY groups 1 to 10 in the mylib.interval2 data table contain the interval fit information of the time ID values and shows the number of counts, alignment, and seasonality for each by group. All BY groups conform exactly to the MONTH interval. The information can be useful in diagnosing time series that contain many time intervals.

Chapter 29

TSMODEL Procedure

Contents

Overview: TSMODEL Procedure	2068
Comparison of the TSMODEL and TIMESERIES Procedures	2068
Comparison of the TSMODEL and TIMEDATA Procedures	2068
Using CAS Sessions and CAS Engine Librefs	2071
Getting Started: TSMODEL Procedure	2072
Syntax: TSMODEL Procedure	2074
Functional Summary	2075
PROC TSMODEL Statement	2076
BY Statement	2081
ID Statement	2081
INSCALARS Statement	2089
OUTARRAYS Statement	2089
OUTSCALARS Statement	2090
PRINT Statement	2090
VAR Statement	2091
REQUIRE Statement	2091
SUBMIT Statement	2092
ENDSUBMIT Statement	2093
SAS Programming Statements	2093
Details: TSMODEL Procedure	2093
Accumulation	2094
Missing Value Interpretation	2095
Summary Statistics	2095
SAS Programming Statements	2095
Predefined Symbols	2096
Auxiliary Tables	2097
Table Output	2100
Printed Output	2103
Examples: TSMODEL Procedure	2103
Example 29.1: Accumulating Transactional Data into Time Series Data	2103
Example 29.2: User-Defined Program Statements	2105
Example 29.3: Using Auxiliary Tables	2107
References	2109

Overview: TSMODEL Procedure

The TSMODEL procedure is a SAS procedure that executes user-defined programs on time series data. The procedure analyzes timestamped transactional data with respect to time and accumulates the data into a time series format.

The TSMODEL procedure forms time series from input timestamped transactional data and writes the accumulated time series variables to an output table. Time series are delineated based on the distinct values of the variables that are listed in the BY statement.

Timestamped transactional data are recorded at no fixed interval. Analysts often want to use time series analysis techniques that require fixed-time intervals. Therefore, the transactional data must be accumulated to form a fixed-interval time series, such as daily, weekly, or monthly.

The TSMODEL procedure forms time series vectors from timestamped data and then provides these vectors as array variables for subsequent processing by your program statements. Your program statements are processed independently for each BY group. The TSMODEL procedure is similar to the SAS DATA step for time series data. The SAS DATA step processes data by each row, whereas PROC TSMODEL processes time series vectors for the BY groups.

Because PROC TSMODEL runs on SAS Cloud Analytic Services (CAS), it can process the time series data in parallel. Time series for the BY groups are divided across the nodes of the CAS session, and then threads are used on each node to process the node's BY groups concurrently.

All results of the time series analysis can be saved to output tables.

Comparison of the TSMODEL and TIMESERIES Procedures

The TSMODEL procedure has similarities to the TIMESERIES procedure in SAS/ETS. The TIMESERIES procedure enables you to perform a variety of standard time series analysis techniques with its various statements, whereas the TSMODEL procedure enables you to define your own analyses via user-defined program statements that you include in the procedure's statement block. The TSMODEL procedure provides no built-in time series analysis capabilities. You must provide a program to analyze the time series data. For more information about PROC TIMESERIES, see *SAS/ETS User's Guide*.

Comparison of the TSMODEL and TIMEDATA Procedures

The syntax of the TSMODEL procedure is similar to the syntax of the TIMEDATA procedure in SAS/ETS software. For more information about PROC TIMEDATA, see *SAS/ETS User's Guide*.

PROC TSMODEL requires that you specify CAS tables for all input data and all output data. Unlike PROC TIMEDATA, no actual processing of the time series data occurs in PROC TSMODEL. Data in PROC TSMODEL are processed in the CAS session that is associated with the CAS librefs that you specify in the PROC TSMODEL statement.

Like PROC TIMEDATA, PROC TSMODEL supports auxiliary input tables, which you specify in the AUXDATA= option. However, PROC TSMODEL requires that either all or none of the BY variables are

present in an AUXDATA= table. No partial BY group matching for the AUXDATA= tables is supported at this time.

A simple example provides insight about the relative ease of moving from PROC TIMEDATA to PROC TSMODEL. Consider the following PROC TIMEDATA example, which rescales the Sale and Price variables for each of the Product BY groups:

```
proc timedata data=mylib.pricex
    outsum=mylib.pricexsum
    outarray=mylib.pricexoa
    outscalar=mylib.pricexos;
by Product;
id date interval=month start='01jan1998'd end='01dec2002'd;
var Sale / accumulate=total;
var Price / accumulate=avg;
outarray relsale relprice;
outscalar sbase pbase;
sbase=Sale[1];
pbase=Price[1];
do i=1 to _length_;
    if Sale[i] ne . then do;
        relsale[i] = Sale[i]/sbase;
    end;
    if Price[i] ne . then do;
        relprice[i]=Price[i]/pbase;
    end;
end;
run;
```

When you convert the preceding statements to use PROC TSMODEL, the code looks like this:

```
proc tsmodel data=mylib.pricex
    outsum=mylib.pricexsum
    outarray=mylib.pricexoa
    outscalar=mylib.pricexos;
by Product;
id date interval=month start='01jan1998'd end='01dec2002'd;
var Sale /accumulate=total;
var Price /accumulate=avg;
outarray relsale relprice;
outscalar sbase pbase;
submit;
sbase=Sale[1];
pbase=Price[1];
do i=1 to _length_;
    if Sale[i] ne . then do;
        relsale[i] = Sale[i]/sbase;
    end;
    if Price[i] ne . then do;
        relprice[i]=Price[i]/pbase;
    end;
end;
endsubmit;
quit;
```

Examine these two examples closely to see the differences. At first glance, the differences might not be very

obvious. Although the differences in code syntax are subtle, the underlying differences in the procedure processing are profound.

One important difference between PROC TIMEDATA and PROC TSMODEL is that PROC TSMODEL requires you to use the SUBMIT and ENDSUBMIT statements to begin and end the set of programming statements that you want to execute on the time series BY groups. You can include SAS macro statements in the statements in your SUBMIT/ENDSUBMIT block. Table 29.1 shows the differences between the processing of programming statements in PROC TSMODEL compared to PROC TIMEDATA.

Table 29.1 Comparison of PROC TSMODEL and PROC TIMEDATA

Processing Step	PROC TSMODEL	PROC TIMEDATA
Statement parsing	Any parse-time errors are reported only after the ENDSUBMIT statement. Those errors come from the parsing and compilation of your code in your CAS session. SAS line numbers for incorrect statements are not reported in the SAS log.	Statements are parsed as the procedure works its way through its statement block. Any parse-time errors are interspersed with the program statements in the SAS log.
PROC FCMP subroutines and functions	Does not support as many built-in PROC FCMP functions and subroutines.	Supports built-in PROC FCMP functions and subroutines.
RUN_MACRO	Not supported because it cannot be performed by the CAS server when it executes your program statements in the context of your CAS session.	Can call the PROC FCMP subroutine RUN_MACRO to run a SAS macro, which can then call a SAS procedure.
ID statement with a SAS date or datetime variable	An ID variable is required because the value of the ID variable is needed to determine the time index of each row within the BY group. The accumulation frequency that you specify in the INTERVAL= option in the ID statement is required.	An ID statement is not required.
Accumulation modes	ACCUMULATE=NONE, FIRST, LAST, NOBS, and MEDIAN are not supported.	ACCUMULATE=NONE, FIRST, LAST, NOBS, and MEDIAN are supported.
Output data	Can be printed and plotted through other SAS procedures.	Supports PRINT and PLOT options to display data.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: TSMODEL Procedure

This section outlines the use of the TSMODEL procedure and describes some of the analysis techniques that you can perform on timestamped transactional data.

Suppose that a bank wants to analyze the transactions that are associated with each of its customers over time. Further, suppose that the table `mylib.transactions` contains four variables that are related to these transactions: `Customer`, `Date`, `Withdrawals`, and `Deposits`. The following examples illustrate possible ways to analyze these transactions by using the TSMODEL procedure.

The following statements accumulate the timestamped transactional data to form a daily time series based on the accumulated daily totals of each type of transaction (`Withdrawals` and `Deposits`). These statements assume that your CAS engine libref is named `mylib`, but you can substitute any appropriately defined CAS engine libref.

```
proc tsmodel data=mylib.transactions
    out=mylib.timeseries(replace=yes)
    outarray=mylib.arrays(replace=yes);
  by Customer;
  id Date interval=day accumulate=total;
  var withdrawals deposits;
  outarrays Balance;
  submit;
  Balance[1] = deposits[1] - withdrawals[1];
  do t = 2 to _LENGTH_;
    Balance[t] = Balance[t-1] + (deposits[t] - withdrawals[t]);
  end;
  endsubmit;
quit;
```

The `OUT=` option requests that the resulting time series data for each customer be stored in the table `mylib.timeseries`. The `OUTARRAY=` option requests that the resulting time series data along with a newly created variable, `Balance`, be stored in the table `mylib.arrays`. Both tables are created in the CAS session's current library.

The `INTERVAL=DAY` option requests that the transactions be accumulated on a daily basis within each `Customer` according to the values of the `ID` variable `Date`. The `ACCUMULATE=TOTAL` option requests that the sum of the transactions be calculated. After the transactional data are accumulated into a time series format, the example code computes a daily balance for each customer. Many of the procedures provided with SAS software can be used to perform further processing on the resulting time series data. The tables that are produced by PROC TSMODEL can also be used as input to subsequent PROC TSMODEL steps.

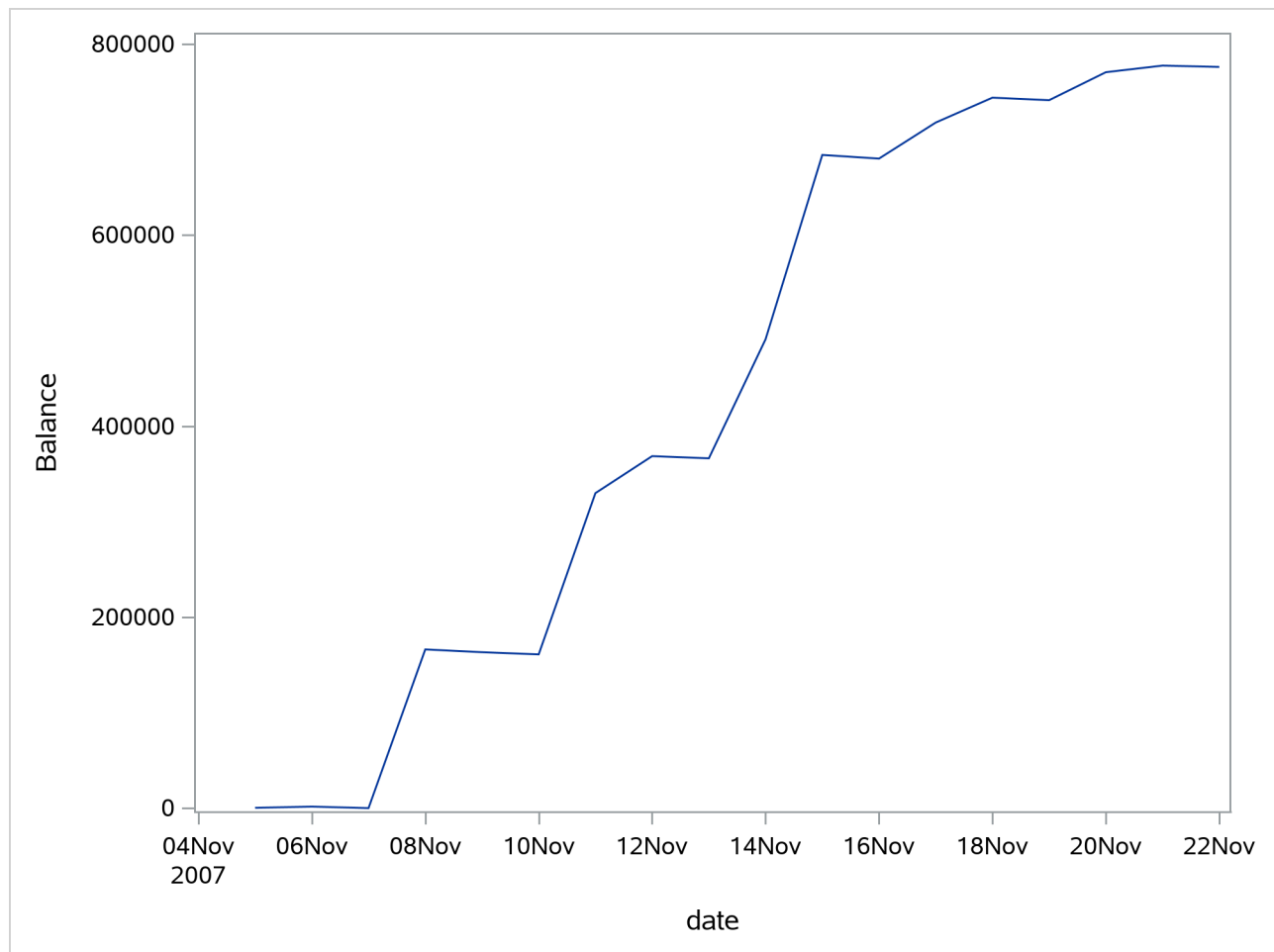
The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in Figure 29.1.

Figure 29.1 Summary of Time Series Processing
The TSMODEL Procedure

Processing Summary	
CAS table	TRANSACTIONS
Number of analysis variables	2
Number of rows read	600
Number of groups read	3
Memory for group packages (KB)	1
Time to load groups (seconds)	0.0079259872
Minimum time ID	05NOV2007
Maximum time ID	22NOV2007
Minimum time periods	18
Maximum time periods	18
Number of nodes run	1
Number of nodes with data	1
Number of nodes with groups	1
Number of threads budgeted	32
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	32
Maximum threads active	32
Number of groups processed by submitted code	3
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0168471336
Number of output table rows produced	54
Number of array table rows produced	54

You might want to plot the generated Balance series for some particular customer. The following code produces a graph for the customer named 'Bill'; the graph is shown in [Figure 29.2](#).

```
proc sgplot data=mylib.arrays (where=(customer='Bill')) ;
    series x=Date y=balance;
run;
```

Figure 29.2 Balance for Bill

Syntax: TSMODEL Procedure

The following statements are available in the TSMODEL procedure:

```

PROC TSMODEL options ;
  BY variables ;
  ID variable INTERVAL=interval < options > ;
  OUTARRAYS array-name-list ;
  OUTSCALARS scalar-name-list ;
  INSCALARS scalar-name-list ;
  VAR variable-list / options ;
  REQUIRE package-list ;
  PRINT print-options ;
  SUBMIT < FILE= SAS-file-ref | 'File-path' > < submit-options > ;
  Programming statements ;
  ENDSUBMIT ;

```

The PROC TSMODEL and ID statements are required.

The following sections present a summary of the statements and options that are used in PROC TSMODEL, a description of the TSMODEL statement, and then descriptions of the other statements.

Functional Summary

Table 29.2 summarizes the statements and options that control the TSMODEL procedure.

Table 29.2 Functional Summary

Description	Statement	Option
Statements		
Specifies BY-group processing	BY	
Specifies variables to analyze	VAR	
Specifies the time ID variable	ID	
Specifies the arrays to output	OUTARRAYS	
Specifies the scalars to output	OUTSCALARS	
Specifies the scalars that are input	INSCALARS	
Specifies the packages to include	REQUIRE	
Specifies the beginning of program statements	SUBMIT	
Specifies the end of program statements	ENDSUBMIT	
Print results from program execution	PRINT	
Table Options		
Specifies the auxiliary input tables	PROC TSMODEL	AUXDATA=
Specifies the input table	PROC TSMODEL	DATA=
Specifies the output table	PROC TSMODEL	OUT=
Specifies the array output table	PROC TSMODEL	OUTARRAY=
Specifies the scalar output table	PROC TSMODEL	OUTSCALAR=
Specifies the summary statistics output table	PROC TSMODEL	OUTSUM=
Specifies the table to contain the BY-group message output log	PROC TSMODEL	OUTLOG=
Specifies whether messages are output to the OUTLOG= table by severity	PROC TSMODEL	LOGCONTROL=
Specifies the input table for scalar variables	PROC TSMODEL	INSCALAR=
Specifies a collector object for an output table	PROC TSMODEL	OUTOBJ=
Specifies a repeater object for an input table	PROC TSMODEL	INOBJ=
Accumulation and Seasonality Options		
Specifies the accumulation frequency	ID	INTERVAL=
Specifies the length of the seasonal cycle	PROC TSMODEL	SEASONALITY=

Table 29.2 *continued*

Description	Statement	Option
Specifies the interval alignment	ID	ALIGN=
Specifies the method for partitioning input tables	ID	DISTRIBUTE=
Specifies the use of the best international format for the time ID values in output tables	ID	NLFORMAT=
Specifies the starting time ID value	ID	START=
Specifies the ending time ID value	ID	END=
Specifies the accumulation statistic	ID, VAR	ACCUMULATE=
Specifies how to interpret missing values	ID, VAR	SETMISSING=
Specifies the method for trimming BY groups	ID	TRIMID=
Miscellaneous Options		
Specifies whether the program stops running when an error occurs during execution of functions or methods	PROC TSMODEL	ERRORSTOP=
Specifies the forecast horizon or lead used to extend the table	PROC TSMODEL	LEAD=
Specifies the maximum number of threads to use for multithreaded processing	PROC TSMODEL	NTHREADS=
Specifies the format for the time ID variable	ID	FORMAT=
Specifies the file that contains the user-defined program	SUBMIT	FILE=
Specifies variable values available to the user-defined program	SUBMIT	DYNAMICS=

PROC TSMODEL Statement

PROC TSMODEL *options* ;

The PROC TSMODEL statement invokes the TSMODEL procedure. You can specify the following *options*:

AUXDATA=*libref.data-table*

names a table that contains auxiliary input data for the procedure to use for supplying time series variables. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 2071. For more information, see the section “Auxiliary Tables” on page 2097.

DATA=*libref.data-table*

names the input data table for PROC TSMODEL to use. The default is the most recently created data table. *libref.data-table* is a two-level name, where

<i>libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>libref</i> , see the section “ Using CAS Sessions and CAS Engine Librefs ” on page 2071.
<i>data-table</i>	specifies the name of the input data table.

ERRORSTOP=YES | NO

specifies whether the program stops running when an error occurs during execution of package functions or object methods. When ERRORSTOP=YES, the program stops running when an error is encountered while a package function or object method is being executed. When ERRORSTOP=NO, the program continues running despite errors.

By default, ERRORSTOP=NO.

If you specify the BY statement, then the ERRORSTOP= option applies independently to each BY group. When ERRORSTOP=YES, if one BY group encounters an error and stops execution, this does not cause the other BY groups or the procedure to stop running.

INOBJ=(*object-name=libref.data-table ...*)

specifies pairs, each of which binds a repeater object specified by *object-name* with an input table specified by *libref.data-table*. You can specify one or more object-table pairs as needed to associate the repeater objects that you declare in your user-defined program with their input tables. You must specify a binding for any repeater object that you declare in your program; otherwise, a parse-time error is generated when you submit the program and no execution occurs. Consider the following SAS code:

```
inobj=(inest=mylib.outest
       inspec=mylib.outspec)
```

This code binds the repeater objects named INEST and INSPEC to the tables mylib.outest and mylib.outspec, respectively. In addition to the columns that are required to satisfy the built-in table schema of a particular repeater object, each specified table must have all or none of the BY variables of the primary DATA= table. When the INOBJ= table has none of the BY variables, all the table rows are input.

Repeater objects are defined in various packages that use PROC TSMODEL as a method to input data that are required for each application. For more information about creating repeater objects for a package, see *SAS Visual Forecasting: Time Series Packages*. For more information about package access, see the section “[REQUIRE Statement](#)” on page 2091.

INSCALAR=libref.data-table

specifies a table to supply scalar dynamic variables to be included and made accessible to your program code as it executes.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 2071.

When you specify BY variables for the DATA= table, the INSCALAR= table must contain those BY variables. If you do not specify BY variables, then the INSCALAR= table is read unqualified for the BY variables across the CAS workers in the CAS session. In this case, only a single value for each variable is needed, and only a single row is required in the table. If the table contains multiple rows in this case, an error is generated when the procedure is called. If you specify this option, then you must also specify one or more INSCALARS statements to specify the variables that you want to include for your program to access.

If BY variables are specified in a BY statement, then the table specified in the INSCALAR= option must contain all the specified BY variables or none of them. If BY variables are present in the INSCALAR= table, then the values for the variables specified in the INSCALARS statement are input subject to BY-group processing. If BY variables are not present in the INSCALAR= table, then only a single value of each variable specified in the INSCALARS statement is input for all BY groups. It is an error for the INSCALARS= table to contain more than one value (row) for the variables if the table is not subject to BY-group processing. If the INSCALARS= table is subject to BY-group processing and multiple values (rows) are present for any BY group, then the value can be input from any row, leading to inconsistent results. For consistent results, you should prepare the INSCALAR= tables such that only a single value is input for each BY group.

LEAD=*n*

specifies the number of periods ahead to extend time series arrays for the variables in both the VAR and OUTARRAYS statements that are output to the table. You can specify this option to accommodate a forecast lead or horizon when you are preparing time series data for forecasting.

The value of *n* is relative to the ending value of the input time ID for each BY group as specified by the [TRIMID=](#) option in the ID statement; it is not relative to the last nonmissing observation of a particular series. By default, LEAD=0.

LOGCONTROL=(*severity=IGNORE* | *KEEP* < *severity=IGNORE* | *KEEP* . . . >)

specifies pairs that define error severity and associated message disposition for the [OUTLOG=](#) option. You can specify multiple LOGCONTROL= options.

You can specify zero or more pairs. In these pairs, *severity* can take one of the following values:

NONE	specifies messages that have no severity classification.
NOTE	specifies messages whose severity classification is NOTE.
WARNING	specifies messages whose severity classification is WARNING.
ERROR	specifies messages whose severity classification is ERROR.

You can specify the following values to indicate the associated message disposition:

IGNORE	ignores messages of the specified severity.
KEEP	retains messages of the specified severity.

This option is applied only when you specify an OUTLOG= option; otherwise, it has no effect. By default, LOGCONTROL=(ERROR=KEEP) when the OUTLOG= option is specified. This default value retains only messages whose severity classification is ERROR and discards all others. However, the default behavior no longer applies when you specify the LOGCONTROL= option. For example, if you specify LOGCONTROL=(NONE=KEEP), then only messages that have no severity classification are retained and all others (that is, those whose classification is NOTE, WARNING, or ERROR) are discarded.

NTHREADS=*n*

specifies the number of threads that are used per worker node in a CAS session. Threads are used to both preprocess and analyze input data in parallel. You must specify a value *n* that is greater than or equal to 0. If you specify 0, the number of threads is set to the maximum number of licensed cores. By default, NTHREADS=0.

OUT=libref.data-table

names the output table to contain the time series variables that are specified in the subsequent VAR statements.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 2071.

If BY variables are specified, they are also included in this output table. The ID variable’s fixed-interval time ID sequence is included in the OUT= table. The time series variables are accumulated based on the INTERVAL= option and the variable’s ACCUMULATE= option. The OUT= table is particularly useful when you want to further analyze, model, or forecast the resulting time series with other SAS procedures.

OUTARRAY=libref.data-table

names the output table to contain the time series vectors that are specified in the VAR and OUTARRAYS statements. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 2071.

This table also contains the variables that are specified in the BY, ID, and VAR statements in addition to the arrays that are specified in the OUTARRAYS statements.

OUTLOG=libref.data-table

names the output table to contain textual messages that are collected from the execution of the BY-group processing.

Messages captured for the BY group are subject to prefiltering by their severity based on the setting of the LOGCONTROL= option. If PUTTOLOG=YES is specified, then messages from the PUT programming statement are also included in this table. This table has rows only for BY groups that generate text messages. Messages that are related to the PROC TSMODEL syntax are not included in this table. Normally, this table contains no rows.

OUTOBJ=(object-name=libref.data-table ...)

specifies pairs, each of which binds a collector object specified by *object-name* with an output table specified by *libref.data-table*.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 2071.

You can specify one or more object-table pairs as needed to associate the collector objects that you declare in your user-defined program with their output tables. You must specify a binding for any collector object that you declare in your program; otherwise, a parse-time error is generated when you submit the program and no execution occurs. Consider the following SAS code:

```
outobj=(oss=mylib.saleoss1
        pest=mylib.salepest1
        ofor=mylib.salefor1
        oind=mylib.saleind1
        ostat=mylib.salestat1)
```

This code binds the collector objects named OSS, PEST, OFOR, OIND, and OSTAT to the tables named SALEOSS1, SALEPEST1, SALEFOR1, SALEIND1, and SALESTAT1, respectively.

Collector objects are defined in various packages that can be run by PROC TSMODEL. For more information about using packages, see *SAS Visual Forecasting: Time Series Packages*. For more information about package access, see the section “[REQUIRE Statement](#)” on page 2091.

OUTSCALAR=libref.data-table

names the output table to contain the scalar names that are specified in the OUTSCALARS statements. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 2071.

This table also contains the variables that are specified in the BY statement and the scalars that are specified in the OUTSCALARS statements.

OUTSUM=libref.data-table

names the output table to contain the descriptive statistics. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 2071.

The descriptive statistics are based on the accumulated time series when you specify the ACCUMULATE= option, the SETMISSING= option, or both options in the ID or VAR statement. The accumulated time series also contains the time horizon that is extended by the value that you specify in the LEAD= option. The OUTARRAY= table shows the data table that is used to calculate the statistics. The results table contains the statistics only for the variables in the OUTARRAYS and VAR statements. It is particularly useful when you want to analyze large numbers of series and you need a summary of the variables.

PUTTOLOG=YES | NO

specifies whether to capture messages from the PUT programming statement to the BY group's row in the table that is specified in the **OUTLOG=** option. You can specify the following values:

NO does not capture messages.

YES captures messages.

This option is applied only when you specify the **OUTLOG=** option; otherwise, it has no effect. This option should be used only to aid in debugging your user-defined programs on small amounts of data. This option can produce large amounts of output that increases processing time and memory usage in your CAS session processes and should be used with caution. By default, **PUTTOLOG=NO**.

SEASONALITY=number

specifies the length of the seasonal cycle. For example, **SEASONALITY=3** means that every group of three time periods forms a seasonal cycle. By default, the length of the seasonal cycle is 1 (no seasonality) or the length that is implied by the **INTERVAL=** option in the **ID** statement. For example, **INTERVAL=MONTH** implies that the length of the seasonal cycle is 12. For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*).

BY Statement

BY variables ;

You can include a **BY** statement with **PROC TSMODEL** to obtain separate processing for groups of observations that are defined by the **BY variables**. The rows in each distinct **BY** group are used to accumulate the time series vectors from the timestamped row data according to a desired frequency (which is specified in the **INTERVAL=** option) and the accumulation mode (which is specified in the **ACCUMULATE=** option).

For more information about table output and **BY** group processing, see the section “[Table Output](#)” on page 2100.

ID Statement

ID variable INTERVAL=interval < options > ;

The **ID** statement names a numeric *variable* that identifies the temporal order (time sequence) of observations in the input and output tables. The values of variables are assumed to be SAS date, SAS datetime, or observation values. In addition, the **ID** statement specifies the frequency to be associated with the time series. The **ID** statement *options* also specify a global treatment for how the time series variables are accumulated from the **BY** group's rows and how the time **ID** values are aligned to form the time series. The specified information affects all variables that are specified in subsequent **VAR** statements. The **ID** statement and the **INTERVAL=** option are required to specify the desired accumulation frequency.

You must specify the following option:

INTERVAL=*interval*

specifies the frequency of the accumulated time series. For example, if the input table consists of quarterly observations, then specify **INTERVAL=QTR**.

Interval names are constructed from a basic interval type with an optional multiplier and shift. The general form of an interval name is as follows:

type< *multiplier*>.< *shift*>

The interval *type* for SAS date values are summarized as follows:

YEAR

specifies yearly intervals. Abbreviations are YEAR, YEARS, YEARLY, YR, ANNUAL, ANNUALLY, and ANNUALS. The starting subperiod *shift* is in months ([MONTH](#)).

YEARV

specifies ISO 8601 yearly intervals. The ISO 8601 year starts on the Monday immediately preceding January 4 (or on January 4 if it is a Monday). Note that it is possible for the ISO 8601 year to start in December of the preceding year. Also, some ISO 8601 years contain a leap week. For more information about ISO weeks, see Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). The starting subperiod *shift* is in ISO 8601 weeks ([WEEKV](#)).

R445YR

is the same as YEARV except that the starting subperiod *shift* is in retail 4-4-5 months ([R445MON](#)).

R454YR

is the same as YEARV except that the starting subperiod *shift* is in retail 4-5-4 months ([R454MON](#)). For more information about the retail 4-5-4 calendar, see National Retail Federation (2007).

R544YR

is the same as YEARV except that the starting subperiod *shift* is in retail 5-4-4 months ([R544MON](#)).

SEMIYEAR

specifies semiannual intervals (every six months). Abbreviations are SEMIYEAR, SEMIYEARS, SEMIYEARLY, SEMIYR, SEMIANNUAL, and SEMIANN.

The starting subperiod *shift* is in months ([MONTH](#)). For example, SEMIYEAR.3 intervals are March–August and September–February.

QTR

specifies quarterly intervals (every three months). Abbreviations are QTR, QUARTER, QUARTERS, QUARTERLY, QTRLY, and QTRS. The starting subperiod *shift* is in months ([MONTH](#)).

R445QTR

specifies retail 4-4-5 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The starting subperiod *shift* is in retail 4-4-5 months ([R445MON](#)).

R454QTR

specifies retail 4-5-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. For more information about the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *shift* is in retail 4-5-4 months ([R454MON](#)).

R544QTR

specifies retail 5-4-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The starting subperiod *shift* is in retail 5-4-4 months ([R544MON](#)).

MONTH

specifies monthly intervals. Abbreviations are MONTH, MONTHS, MONTHLY, and MON. The starting subperiod *shift* is in months (MONTH). For example, MONTH2.2 intervals are February–March, April–May, June–July, August–September, October–November, and December–January of the following year.

R445MON

specifies retail 4-4-5 monthly intervals. The 3rd, 6th, 9th, and 12th months are five ISO 8601 weeks long with the exception that some 12th months contain leap weeks. All other months are four ISO 8601 weeks long. R445MON intervals begin with the 1st, 5th, 9th, 14th, 18th, 22nd, 27th, 31st, 35th, 40th, 44th, and 48th weeks of the ISO year. The starting subperiod *shift* is in retail 4-4-5 months ([R445MON](#)).

R454MON

specifies retail 4-5-4 monthly intervals. The 2nd, 5th, 8th, and 11th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R454MON intervals begin with the 1st, 5th, 10th, 14th, 18th, 23rd, 27th, 31st, 36th, 40th, 44th, and 49th weeks of the ISO year. For more information about the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *shift* is in retail 4-5-4 months ([R454MON](#)).

R544MON

specifies retail 5-4-4 monthly intervals. The 1st, 4th, 7th, and 10th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R544MON intervals begin with the 1st, 6th, 10th, 14th, 19th, 23rd, 27th, 32nd, 36th, 40th, 45th, and 49th weeks of the ISO year. The starting subperiod *shift* is in retail 5-4-4 months ([R544MON](#)).

SEMIMONTH

specifies semimonthly intervals. SEMIMONTH breaks each month into two periods, starting on the 1st and 16th days. Abbreviations are SEMIMONTH, SEMIMONTHS, SEMIMONTHLY, and SEMIMON. The starting subperiod *shift* is in SEMIMONTH periods. For example, SEMIMONTH2.2 specifies intervals from the 16th of one month through the 15th of the next month.

TENDAY

specifies 10-day intervals. TENDAY breaks the month into three periods, the 1st through the 10th day of the month, the 11th through the 20th day of the month, and the remainder of the month. (TENDAY is a special interval typically used for reporting automobile sales data.) The starting subperiod *shift* is in TENDAY periods. For example, TENDAY4.2 defines 40-day periods that start at the second TENDAY period.

WEEK

specifies weekly intervals of seven days. Abbreviations are WEEK, WEEKS, and WEEKLY. The starting subperiod *shift* is in days (DAY), with the days of the week numbered as 1=Sunday, 2=Monday, 3=Tuesday, 4=Wednesday, 5=Thursday, 6=Friday, and 7=Saturday. For example, WEEK.7 means weekly with Saturday as the first day of the week.

WEEKV

specifies ISO 8601 weekly intervals of seven days. Each week starts on Monday. The starting subperiod *shift* is in days (DAY). Note that WEEKV differs from WEEK in that WEEKV.1 starts on Monday, WEEKV.2 starts on Tuesday, and so on.

WEEKDAY**WEEKDAYdW****WEEKDAYddW****WEEKDAYdddW**

specifies daily intervals with weekend days included in the preceding weekday. Note that for a five-day work week that starts on Monday, the appropriate interval is WEEKDAY5.2. Abbreviations are WEEKDAY and WEEKDAYS. The starting subperiod *shift* is in weekdays (WEEKDAY).

The WEEKDAY interval is the same as DAY except that weekend days are absorbed into the preceding weekday. Thus, there are five WEEKDAY intervals in a calendar week: Monday, Tuesday, Wednesday, Thursday, and the three-day period Friday-Saturday-Sunday.

The default weekend days are Saturday and Sunday, but any one to six weekend days can be listed after the WEEKDAY string and followed by a W. Weekend days are specified as '1' for Sunday, '2' for Monday, and so forth. For example, WEEKDAY67W specifies a Friday-Saturday weekend. WEEKDAY1W specifies a six-day work week with a Sunday weekend. WEEKDAY17W is the same as WEEKDAY.

DAY

specifies daily intervals. Abbreviations are DAY, DAYS, and DAILY. The starting subperiod *shift* is in days (DAY).

The interval *type* for SAS datetime values are summarized as follows:

DTSAS-date-interval

specifies a SAS datetime interval that corresponds to the SAS date interval but operates on SAS datetime values. The SAS datetime interval is created by adding the prefix "DT" to the SAS date interval name. For example, DTMONTH.

HOURL

specifies hourly intervals. Aliases are HOUR, DTHOUR, HOURS, DTHOURS, HOURLY, DTHOURLY, HR, and DTHR. The starting subperiod *shift* is in hours (HOUR).

MINUTE

specifies minute intervals. Aliases are MINUTE, DTMINUTE, MINUTES, DTMINUTES, MIN, and DTMIN. The starting subperiod *shift* is in minutes (MINUTE).

SECOND

specifies second intervals. Aliases are SECOND, DTSECOND, SECONDS, DTSECONDS, SEC, and DTSEC. The starting subperiod *shift* is in seconds (SECOND).

The interval *type* for SAS observation values is summarized as follows:

OBS

n

specifies that observation numbers identify the time periods. You can specify INTERVAL=*n* as an alias for INTERVAL=OBS*n*.

If the SEASONALITY= option is not specified in the PROC TSMODEL statement, then the length of the seasonal cycle is implied by the INTERVAL= option. For example, INTERVAL=QTR implies a seasonal cycle of length 4. For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*).

You can also specify the following *options*:

ACCUMULATE=option

specifies how to accumulate the table observations within each time period into a single value. The frequency (width of each time interval) is specified in the INTERVAL= option. The ID *variable* contains the time ID values. The value of the time ID *variable* identifies the time period of the observation. The accumulated values for each time period form the time series, which is used in subsequent analysis.

The ACCUMULATE= option is necessary when multiple input observations identify the same time period. An example of this is timestamped transactional data.

The following *options* determine how to accumulate the observations within each time period based on the ID variable and the frequency specified by the INTERVAL= option. Each value indicates how the accumulated value is calculated:

TOTAL SUM	calculates as the total sum of the nonmissing values.
AVERAGE AVG	calculates as the average of the nonmissing values.
MINIMUM MIN	calculates as the minimum of the nonmissing values.
MAXIMUM MAX	calculates as the maximum of the nonmissing values.
N	calculates as the number of nonmissing observations.
NMISS	calculates as the number of missing observations.
STDDEV STD	calculates as the standard deviation of the nonmissing values.
CSS	calculates as the corrected sum of squares of the nonmissing values.
USS	calculates as the uncorrected sum of squares of the nonmissing values.

By default, `ACCUMULATE=TOTAL`.

The `SETMISSING=` option is useful for specifying how to treat missing values in the accumulated time series variable. If missing values should be interpreted as 0, then specify `SETMISSING=0`. For more information about accumulation, see the section “[Accumulation](#)” on page 2094.

ALIGN=option

controls the alignment of SAS date or datetime values that are used to identify the time period of output observations. Although any date or datetime value within the time period can identify the time period, the `ALIGN=` option requests that the representative date or datetime for the time period be calculated as the beginning date or datetime of the time period, the ending date or datetime of the time period, or the middle date or datetime of the time period. In addition to aligning the time ID values consistently for observations that are supplied by the user, the `ALIGN=` option specifies the method for calculating the time ID values for observations in the forecast and backcast time periods, which often are not supplied by the user. You can specify the following options:

- BEGINNING | BEG | B** represents each time period using the beginning SAS date or datetime value of the time period.
- ENDING | END | E** represents each time period using the ending SAS date or datetime value of the time period.
- MIDDLE | MID | M** represents each time period using the middle SAS date or datetime value of the time period. The middle is calculated as the average of the beginning and ending values.

By default, `ALIGN=BEGINNING`.

DISTRIBUTE=option

specifies the method that is used to distribute the input time series observations in the input tables across the CAS worker nodes in the CAS session. This process, which is known as “autopartitioning,” is performed in order to form complete time series that are suitable for processing. You can specify the following *options*:

- AUTO** automatically selects the best approach, using either the `PREACCUMULATE` or `SHUFFLE` *option*, to distribute the input time series observations across the CAS worker nodes in the CAS session. The selection occurs on the basis of both the accumulation time interval that you specify in the `INTERVAL=` option and the level of scattering of the time series observations across the CAS worker nodes.
- PREACCUMULATE | PREACC** generates partially accumulated time series (that is, those that have a fixed length and a fixed interval) on each CAS worker node in the CAS session and then distributes their observations to their destination CAS worker nodes. These partially accumulated time series are accumulated again upon arrival at their destination CAS worker nodes in order to form complete time series. This option can usually reduce the amount of time that is required in order to distribute time series observations across the CAS worker nodes, especially in the case in which you specify an accumulation interval in the `INTERVAL=` option that is larger than the natural frequency of the input data. However, this option can require additional memory in order to generate the partially accumulated time series on the CAS worker nodes.

SHUFFLE

distributes the raw time series observations to their destination CAS worker nodes in the CAS session before accumulating them into complete time series. This option is recommended when you specify an accumulation time interval in the **INTERVAL=** option that is close to the natural frequency of the input data. This option requires less memory than the **PREACCUMULATE** option because it does not have to generate partially accumulated time series in the memory of each CAS worker node. However, this option can lead to a longer run time than the **PREACCUMULATE** option when it is used outside the recommended setting.

By default, **DISTRIBUTE=AUTO**.

END=*value*

specifies a SAS date or datetime value that represents the end of the data. If the last time ID variable value is less than *value*, then the series is extended with missing values. If the last time ID variable value is greater than *value*, then the series is truncated. For example, **END=“&sysdate”D** uses the automatic macro variable **SYSDATE** to extend or truncate the series to the current date. You can specify the **START=** and **END=** options to ensure that the data that are associated within each **BY** group contain the same number of observations.

FORMAT=*format*

specifies the SAS format for the time ID values. If this option is omitted, then the default format is inferred from the **INTERVAL=** option. You can specify **FORMAT=_DATA_** to force the SAS format of the time ID variable to be automatically propagated to the time ID values.

NLFORMAT=*option*

enables the selection of the best international SAS format for the time ID values on the basis of the accumulation time interval that you specify in the **INTERVAL=** option. This option is ignored when you specify the **FORMAT=** option. You can specify the following *options*:

NO selects the best English-language-based SAS format for the time ID values.

YES selects the best international SAS format for the time ID values.

By default, **NLFORMAT=NO**.

SETMISSING=*option*

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series. You can use the following *options* to determine how to interpret missing values:

n interprets a missing value as having the value *n*. You can specify any number for *n*, but not a missing value. If a missing value indicates a 0 value, specify **SETMISSING=0**. You typically use **SETMISSING=0** for transactional data because no recorded data usually implies no activity.

MISSING interprets a missing value as a missing value. Use this option if a missing value indicates an unknown value.

AVERAGE | AVG interprets a missing value as the average value of all accumulated nonmissing values in the span of the series.

MINIMUM MIN	interprets a missing value as the minimum value of all accumulated nonmissing values in the span of the series.
MEDIAN MED	interprets a missing value as the median value of all accumulated nonmissing values in the span of the series.
MAXIMUM MAX	interprets a missing value as the maximum value of all accumulated nonmissing values in the span of the series.
FIRST	interprets a missing value as the first nonmissing value of all accumulated nonmissing values in the span of the series.
LAST	interprets a missing value as the last nonmissing value of all accumulated nonmissing values in the span of the series.
PREVIOUS PREV	interprets a missing value as the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.
NEXT	interprets a missing value as the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.

By default, SETMISSING=MISSING.

START=*value*

specifies a SAS date or datetime value that represents the beginning of the data. If the first time ID variable value is greater than *value*, then missing values are added at the beginning of the series. If the first time ID variable value is less than *value*, then the series is truncated. You can specify the START= and END= options to ensure that data associated with each BY group contain the same number of observations.

TRIMID=*method*

specifies the method for trimming the data in the BY groups when time series vectors are input to the user-defined program. The output time ID variable span that is calculated by the *method* is dependent on the input time ID variable span, irrespective of missing values of the time series variables. Depending on the *method* and the input time ID variable data, leading or trailing missing values can be added to the time series variables.

After the output time ID variable span is calculated by the *method*, the ending value of the output time ID variable will be recalculated according to the value of the LEAD= option (if one is specified).

You can specify one of the following *methods*:

- NONE** uses the same starting and ending values of the output time ID variable for all BY groups. The span of the output time ID variable includes all values that are input as a time ID value for all BY groups. The time series variables are extended with leading or trailing missing values as required.
- LEFT** uses the identifying date for the first time period that is input for the BY group as the starting value of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading missing values. The ending value of the time ID variable is the same for all BY groups. The time series variables are extended with trailing missing values as required.

- RIGHT** uses the identifying date for the last time period that is input for the BY group as the ending value of the output time ID variable for each BY group. The time series values in each BY group are not extended with trailing missing values. The starting value of the time ID variable is the same for all BY groups. The time series variables are extended with leading missing values as required.
- BOTH** uses the span of the input time ID variable for the BY group as the span of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading or trailing missing values.

By default, TRIMID=NONE.

Note that when you do not specify the **START=** and **END=** options in the **ID** statement, the TSMODEL procedure is required to make an additional pass through the primary data set that you specified in the **DATA=** option in order to determine its time index span. This is required so that the total number of accumulation time intervals can be computed using the accumulation frequency that you specified in the **INTERVAL=** option. In this scenario, the TSMODEL procedure will read all rows from the primary data set exactly twice. Moreover, this effect will be captured in the numeric value associated with the row labeled 'Number of rows read' of the time series processing summary table that is generated at the end of the execution of the TSMODEL procedure, as is illustrated in [Figure 29.1](#) and [Output 29.1.1](#).

INSCALARS Statement

INSCALARS *scalar-name-list* ;

INSCALAR *scalar-name-list* ;

The INSCALARS statement specifies which scalar variables to automatically include for your program to use. These variables can be numeric or character. Each variable that you name in an INSCALARS statement must be defined in the table specified in the **INSCALAR=** option in the PROC TSMODEL statement. You must specify at least one INSCALARS statement if you specify the INSCALAR= option in the PROC TSMODEL statement. You can specify multiple INSCALARS statements. For more information, see the **INSCALAR=** option.

OUTARRAYS Statement

OUTARRAYS *array-name-list* ;

OUTARRAY *array-name-list* ;

The OUTARRAYS statement specifies a list of array names; each name specifies a numeric output array variable to be stored in the table that is specified in the **OUTARRAY=** option in the PROC TSMODEL statement. You can specify multiple OUTARRAYS statements.

Your program statements can create and use any number of arrays. The array variables that are specified in the OUTARRAYS statement are created automatically for you, and their lengths are predetermined by the length of the time ID vector for the BY group. The arrays are initialized to missing values at the start of each BY group. Although you can define and use many arrays in your program, only arrays that are listed in the OUTARRAYS statement are included in the table that is specified in the OUTARRAY= option.

OUTSCALARS Statement

OUTSCALARS *scalar-name-list* ;

OUTSCALAR *scalar-name-list* ;

The OUTSCALARS statement specifies a list of scalar names; each name specifies a numeric or character output scalar variable to be stored in the table that is specified in the **OUTSCALAR=** option in the PROC TSMODEL statement. You can specify multiple OUTSCALARS statements.

Your program statements can create and use any number of scalars. Only scalars that are listed in the OUTSCALARS statement are predefined and included in the table that is specified in the OUTSCALAR= option. The scalars are initialized to missing values at the start of each BY group. Each scalar name can specify numeric or character scalar variables which are defined as follows:

var-list defines the variables in the *var-list* as numeric scalars.

var-list \$*n* defines the variables in the *var-list* as character scalars with length *n*.

The following statement defines scalar variables *rc*, *holdout*, and *nchanges* as numeric scalars, and defines *selected* as character with length 32.

```
outscalar rc selected $32 holdout nchanges;
```

PRINT Statement

PRINT *print-options* ;

The PRINT statement allows you to display results that are saved to tables that you specify in the PROC TSMODEL statement options. You can include multiple PRINT statements in your PROC TSMODEL statement block. Each PRINT statement is processed independently in the order that you specify them.

The PRINT OUTLOG statement has the following form:

PRINT OUTLOG < **WHERE** *where-clause* > ;

PRINT OUTLOG displays information from the OUTLOG= table to the SAS LOG. If you specify a BY statement, then the messages that are stored to the OUTLOG table are printed in BY group order followed with a BY line separator. You can specify an optional WHERE clause to select the rows from the OUTLOG table that you want to print. The WHERE clause must reference columns of the OUTLOG= table.

Following are four examples of PRINT OUTLOG statements that you might specify:

```
print outlog;
print outlog where regionName eq 'Region1';
print outlog where _errno_ ne 0;
print outlog where _log_ contains "sale";
```

The first PRINT statement prints all OUTLOG= table rows to the SAS LOG, the second selects rows where the *regionName* column has the value 'Region1', the third selects rows where the *_errno_* column has a non-zero value, and the last selects rows where the *_log_* column text contains the word 'sale'.

VAR Statement

VAR *variable-list* < / *options* > ;

The VAR statement lists the numeric variables in the DATA= table or AUXDATA= tables whose values are to be accumulated to form the time series.

An input table variable can be specified in only one VAR statement. You can specify any number of VAR statements. You can also specify the following *options* in the VAR statements:

ACCUMULATE=*option*

specifies how to accumulate the table observations within each time period for the variables in the *variable-list*. If you do not specify this option, accumulation is determined by the **ACCUMULATE=** option in the ID statement.

SETMISS=*option* | *number*

SETMISSING=*option* | *number*

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series for variables in the *variable-list*. If you do not specify this option, missing values are set based on the **SETMISSING=** option in the ID statement.

REQUIRE Statement

REQUIRE *package-1* < *package-2* ... *package-n* > ;

REQUIRE *package* (*class-1* ... *class-n*) ;

The REQUIRE statement specifies which time series and time frequency analysis packages to make available for your user-defined program. These packages include functions, subroutines, and objects that you can utilize from your program to perform sophisticated time series processing. These packages provide functionality that ranges from a simple function to count missing observations in an array to very sophisticated objects that perform automatic time series modeling and forecasting.

The first form of the REQUIRE statement enables you to specify a list of package names that are available for use. If you use this form, you must specify at least one package name. You can specify multiple packages in the same REQUIRE statement.

The second form of the REQUIRE statement enables you to selectively identify which objects in *package* you intend to use. Packages might contain a number of different objects, and it can be more efficient to register only the ones that you plan to use.

You can specify multiple REQUIRE statements. All packages that are specified in REQUIRE statements are loaded prior to parsing your program statements so that any references are defined at the time your code is parsed. If you specify an invalid package name, then an error is returned prior to parsing your program statements. For more information about the packages that are available in SAS Visual Forecasting, see *SAS Visual Forecasting: Time Series Packages*. For more information about the packages that are available in SAS Econometrics, see *SAS Econometrics: Econometrics Procedures*.

SUBMIT Statement

SUBMIT < FILE= SAS-file-ref | 'File-path' > < submit-options > ;

The SUBMIT statement indicates the start of your user-defined program statements or specifies the location of a file that contains your user-defined program statements.

If the SUBMIT statement does not specify a file, then all lines of code that are specified between the SUBMIT statement and the ENDSUBMIT statement are executed on the time series BY groups that arise from your timestamped input data. Compilation errors are reported back to the SAS log, and no further processing occurs.

Notice the placement of the SUBMIT and ENDSUBMIT in the example in the “[Comparison of the TSMODEL and TIMEDATA Procedures](#)” on page 2068. The use of SUBMIT and ENDSUBMIT statements to delineate of the program block is a major difference between PROC TSMODEL and PROC TIMEDATA.

You can specify the following *submit-options*:

FILE=SAS-file-ref | 'File-path'

specifies a file that contains your user-defined program. If you specify this option, you must not specify the ENDSUBMIT statement.

You can specify the file either by including a *SAS-file-ref* that is defined in a prior SAS FILENAME statement or by including a quoted string that identifies the host-specific file path.

The following rules apply when you include the FILE= option:

- The text file you specify must contain only the program statements that you want to submit and none of the PROC TSMODEL statements. You must not include an ENDSUBMIT statement in the file.
- The text file you specify must not contain any SAS macro statements or macro variable references. The file content is submitted as specified.
- You are free to use any mechanism of your choosing to generate the program statements in the text file. You can use a SAS DATA step with PUT statements, PROC LUA, PROC STREAM, and so on to generate the file from within a SAS program. You can use your favorite editor to generate the text file and then specify it in the SUBMIT statement if the content is static. For more information, see [Getting Started with SAS Viya for Lua](#).

DYNAMICS=(var-1=[number | “string”] < var-2=[number | “string”]. . . >)

defines variables (constants) outside of the scope of the submitted code. These variables function as global arguments to the submitted code. In conjunction with the INSCALARS statement, these values can be used and evaluated within the submitted code. You can specify both numeric and character variable types. The type of the variable is determined by the right-hand side of the variable’s declaration. For example, consider the following DYNAMICS= option:

```
DYNAMICS=(SOF='MAPE' HOLDOUT=10)
```

The variable SOF is defined as a character variable, and its static value is 'MAPE'. The variable HOLDOUT is defined as a numeric variable, and its static value is 10.

ENDSUBMIT Statement

ENDSUBMIT ;

The ENDSUBMIT statement is required to terminate the user-defined program statements in the PROC TSMODEL statement block whenever you specify the SUBMIT statement without the FILE= option. Do not specify the ENDSUBMIT statement if you specify the **FILE=** option in the SUBMIT statement. If you do not specify the FILE= option and you fail to include an ENDSUBMIT statement to accompany your SUBMIT statement, then the PROC TSMODEL step ends with an error.

SAS Programming Statements

Programming statements ;

You can use all the programming statements that are allowed in PROC FCMP except the following:

- CALL RUN_MACRO subroutine
- CALL READ_ARRAY subroutine
- CALL WRITE_ARRAY subroutine

Details: TSMODEL Procedure

The TSMODEL procedure forms time series data from transactional data, which are analyzed according to the following steps if the relevant option listed on the right is specified:

- | | |
|---------------------------------|---|
| 1. accumulation | ACCUMULATE= option in the ID or VAR statement |
| 2. missing value interpretation | SETMISSING= option in the ID or VAR statement |
| 3. program execution | user-defined program statements |
| 4. descriptive statistics | OUTSUM= option |

Accumulation

If you specify the `ACCUMULATE=` option in the `ID` or `VAR` statement, observations in the table that is specified in the `DATA=` option in the `PROC TSMODEL` statement are accumulated within each time period. The frequency (width of each time interval) is specified by the `INTERVAL=` option in the `ID` statement. The `ID` variable contains the time `ID` values. Each time `ID` value is a SAS date or datetime that identifies the time period that contains the date or datetime. Accumulation is useful when the input table contains transactional data, whose observations are not spaced with respect to any particular time interval. The accumulated values form the time series, which is used in subsequent analyses.

For example, suppose an input table contains the following observations:

19MAR1999	10
19MAR1999	30
11MAY1999	50
12MAY1999	20
23MAY1999	20

If `INTERVAL=MONTH` is specified in the `ID` statement, then all the preceding observations fall within a three-month period of time between March 1999 and May 1999. The observations are accumulated within each time period as follows:

- If the `ACCUMULATE=TOTAL` option is specified, the resulting time series is

01MAR1999	40
01APR1999	.
01MAY1999	90

- If the `ACCUMULATE=AVERAGE` option is specified, the resulting time series is

01MAR1999	20
01APR1999	.
01MAY1999	30

- If the `ACCUMULATE=MINIMUM` option is specified, the resulting time series is

01MAR1999	10
01APR1999	.
01MAY1999	20

- If the `ACCUMULATE=MAXIMUM` option is specified, the resulting time series is

01MAR1999	30
01APR1999	.
01MAY1999	50

- If the `ACCUMULATE=STDDEV` option is specified, the resulting time series is

01MAR1999	14.14
01APR1999	.
01MAY1999	17.32

As you can see from the preceding examples, the accumulated time series can have missing values even though the input table observations contain no missing values.

Missing Value Interpretation

Once the data has been accumulated to form a time series based on the `INTERVAL=` and the `ACCUMULATE=` options, missing value interpretation is performed. Sometimes missing values should be interpreted as unknown values. In other situations, missing values are known, such as when missing values are created from accumulation and no observation should be interpreted as no value—that is, 0. In the former case, you can specify the `SETMISSING=` option to interpret how to treat missing values. In the latter case, you can specify `SETMISSING=0` in order to treat missing observations as no (zero) values. In other cases, missing values should be interpreted as global values, such as minimum or maximum values of the accumulated series. The accumulated and interpreted time series is used in subsequent analyses.

Summary Statistics

You can compute summary statistics from the working series by specifying the `OUTSUM=` option in the `PROC TSMODEL` statement.

SAS Programming Statements

The user-defined program for `PROC TSMODEL` can contain most of the SAS programming statements and functions available in the `DATA` step or in the `FCMP` procedure. However, there are a few differences as noted in “[SAS Programming Statements](#)” on page 2093. For more information, see the “`FCMP` Procedure” chapter in the *Base SAS Procedures Guide*.

All variables that are specified in the `ID` and `VAR` statements are assigned as predefined arrays for subsequent processing. In addition, all array names that are specified in the `OUTARRAYS` statements and all the scalars names that are specified in the `OUTSCALARS` statements are assigned as predefined symbols for subsequent processing.

Predefined Symbols

In addition to the predefined arrays that are specified in the OUTARRAYS statements and the predefined scalars that are specified in the OUTSCALARS statements, the TSMODEL procedure creates predefined symbols that are automatically available for use in the programming statements. The name and description of the predefined symbols are shown in the following subsections; these names must not be used as variable names in any input data set.

Predefined Scalar Values

<code>_FORMAT_</code>	time format, which is either implied by the <code>INTERVAL=</code> option or specified in the <code>FORMAT=</code> option in the ID statement
<code>_HORIZON_</code>	the time ID value that is one period beyond <code>_TEND_</code>
<code>_INTERVAL_</code>	time interval, which is specified in the <code>INTERVAL=</code> option in the ID statement
<code>_LEAD_</code>	forecast horizon or lead, which is specified in the <code>LEAD=</code> option in the PROC TSMODEL statement
<code>_LENGTH_</code>	length of the time series that is associated with the current BY group
<code>_SEASONALITY_</code>	length of the seasonal cycle, which is specified in the <code>SEASONALITY=</code> option in the PROC TSMODEL statement or implied by the <code>INTERVAL=</code> option in the ID statement. For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (<i>SAS Visual Forecasting: Time Series Packages</i>).
<code>_SERIES_</code>	series index or BY-group counter. When PROC TSMODEL runs in a parallel processing environment, the index value is relative to the machine that processes the series or BY group. Therefore, values of <code>_SERIES_</code> might not be unique and could also vary for multiple executions of the procedure.
<code>_TEND_</code>	the last time ID value over all the BY groups. <code>_TEND_</code> is independent of the value of the <code>TRIMID=</code> option in the ID statement, but the value of <code>_TEND_</code> is the same as the ending value of the output time ID variable span if you specify <code>TRIMID=NONE</code> .
<code>_TSTART_</code>	the first time ID value over all the BY groups. <code>_TSTART_</code> is independent of the value of the <code>TRIMID=</code> option in the ID statement, but the value of <code>_TSTART_</code> is the same as the starting value of the output time ID variable span if you specify <code>TRIMID=NONE</code> .

Predefined Array Values

<code>_SEASON_</code>	season index values
<code>_CYCLE_</code>	life-cycle index values

Auxiliary Tables

The TSMODEL procedure can use auxiliary tables to contribute input variables to the run of the procedure step. This functionality creates a virtual data source that allows some of the input variables to physically reside in different tables. Some input variables can reside in the primary table, which is specified in the DATA= option, and other input variables can reside in the tables that are specified in one or more AUXDATA= options. This functionality enables sharing of common time series data across multiple projects.

You can specify more than one auxiliary data source to be used to input time series vectors across a particular BY-group hierarchy. To simplify data management, you can isolate variables that have naturally different levels of BY-group qualification into separate tables and use separate AUXDATA= options to supply them.

AUXDATA Functionality

There are two classes of time series table sources:

- a primary table from the DATA= option
- auxiliary data sources from AUXDATA= options

The AUXDATA= option specifies an auxiliary table that provides time series variables that are required for processing but are not included in the table that is specified in the DATA= option.

You can specify multiple AUXDATA= options in the PROC TSMODEL statement. Each AUXDATA= option establishes an auxiliary table source to supply variables that are declared in subsequent statements in the procedure step. If no auxiliary data sources are required, then the AUXDATA= option can be omitted.

Variables referenced in the TSMODEL procedure fall into three classes:

- variables that must be physically present in the primary table, which is specified in the DATA= option
- variables that must be physically present in each auxiliary table that is specified in an AUXDATA= option
- variables that can reside in either the primary or an auxiliary table

The ID variable for PROC TSMODEL must be present in the primary table and all the auxiliary tables that you specify. Variables that you specify in the BY statement must be present in the primary table. The auxiliary tables must contain all of those BY variables or none of them. Partial matching of a leftmost subset of the BY variables is not supported for the auxiliary tables.

The time series variables that you specify in VAR statements can be input from either the primary table or an auxiliary table. Variable resolution proceeds in reverse order from the last AUXDATA= option in the PROC TSMODEL statement to the first. If the variable in question is not found in any of those, the variable must be present in the primary table for the procedure step to be successful.

AUXDATA Alignment across BY Groups

All variables in the BY statement must be physically present in the primary table. However, it is not necessary to have the BY variables present in any of the auxiliary tables. All or none of the BY variables can be present in any auxiliary table.

For example, suppose you have a hierarchy of (REGION, PRODUCT) in the primary table Sales, which holds the time series variables for monthly sales metrics. Suppose you have an auxiliary table called Promotions that has no BY variables and contains analysis variables for promotions, and another table called Returns that contains time series analysis variables for (REGION, PRODUCT) level groupings. In this scenario, each (REGION, PRODUCT) group in the Sales table always includes the time series variables from the Promotions table, and the analysis variables for the matching (REGION, PRODUCT) BY groups from the Returns table. So if ('SOUTH', 'EDSEL') is a BY group from the primary table, any matching rows from the Returns table are used to define the time series variables that are contributed from that table. The time series variables that are contributed by the Promotions table are always included in every BY group from the Sales table.

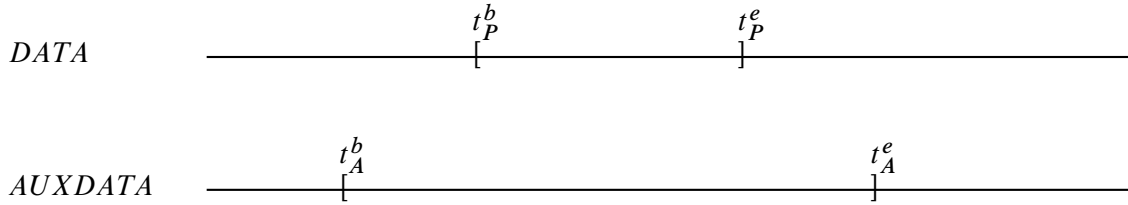
AUXDATA Alignment over the Time Dimension

The series from each BY group of the primary table defines a reference time span for the auxiliary tables. Only the intersection of the time span for each auxiliary series with the reference span is input. Leading or trailing missing values are added to the auxiliary series as required to create a time series that has the same span as the reference span. The leading or trailing missing values, if any, are then interpreted according to the value of the SETMISSING= option. For more information, see the SETMISSING= option in the ID statement and the SETMISSING= option in the VAR statement.

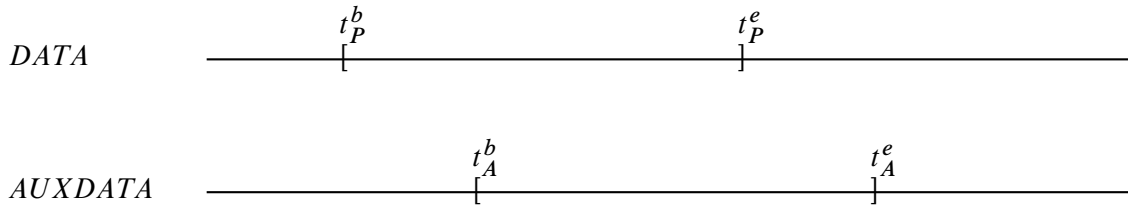
When time series are input from a single primary table, the values for all observations of all the time series are contained in the primary table, and no time series needs to be extended with leading or trailing missing values. However, when time series are input from both the primary table and an auxiliary table, the time series from the auxiliary table are truncated or extended as required if the span of the auxiliary series is not identical to the reference span.

For the preceding (REGION, PRODUCT) example, which includes a primary table and two auxiliary tables, consider how differences in the reference span from each BY group affect the time series input from the auxiliary tables in the following cases. In these cases:

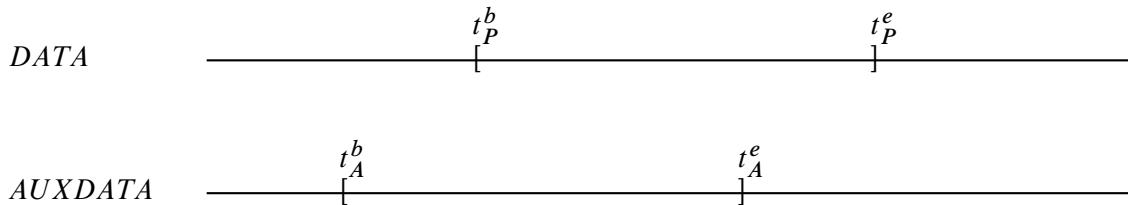
- The DATA line show a series that contains observations in the table specified in the DATA= option.
- The AUXDATA line show a series that contains observations in the table specified in an AUXDATA= option.
- t_P^b denotes the beginning time ID of the primary (DATA) series.
- t_P^e denotes the ending time ID of the primary (DATA) series.
- t_A^b denotes the beginning time ID of the AUXDATA series.
- t_A^e denotes the ending time ID of the AUXDATA series.
- $[t_P^b, t_P^e]$ denotes the time span for the primary (DATA) series (also known as the reference time span).
- $[t_A^b, t_A^e]$ denotes the time span for the AUXDATA series.

Case 1:

In this case, $[t_P^b, t_P^e] \subseteq [t_A^b, t_A^e]$: the auxiliary time span includes the reference span as a subset. Values in the AUXDATA series to the left of t_P^b and values to the right of t_P^e are truncated from the AUXDATA series. AUXDATA series values in $[t_P^b, t_P^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

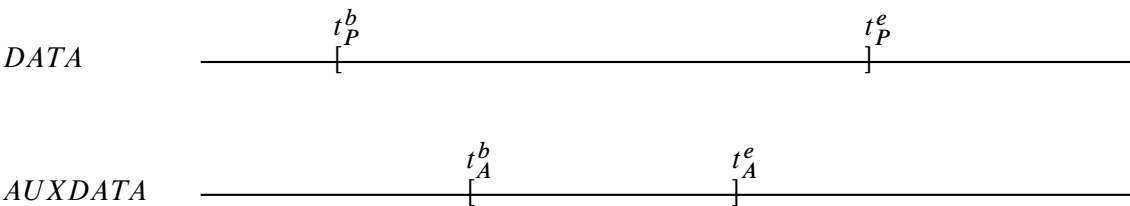
Case 2:

In this case, $[t_P^b, t_P^e] = [t_P^b, t_A^b) \cup [t_A^b, t_P^e]$: the reference time span leads the auxiliary time span with a non-empty intersection. AUXDATA series values in $[t_P^b, t_A^b)$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $[t_A^b, t_P^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_P^e, t_A^e]$ are truncated.

Case 3:

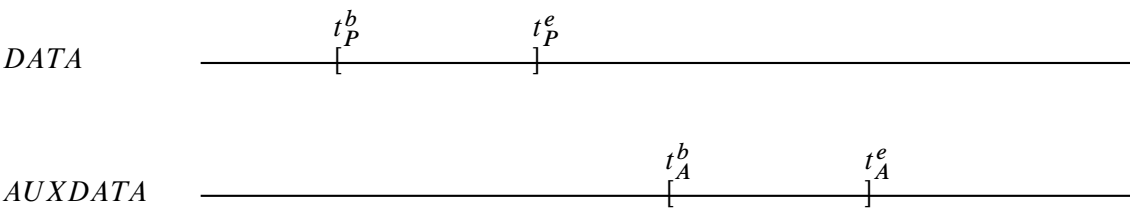
In this case, $[t_P^b, t_P^e] = [t_P^b, t_A^e] \cup (t_A^e, t_P^e]$: the reference time span lags the auxiliary time span with a non-empty intersection. AUXDATA series values in $[t_P^b, t_A^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_A^e, t_P^e]$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_A^b, t_P^b]$ are truncated.

Case 4:



In this case, $[t_A^b, t_A^e] \subset [t_P^b, t_P^e]$: the auxiliary time span is a subset of the reference time span. *AUXDATA* series values in $[t_P^b, t_A^b)$ and values in $(t_A^e, t_P^e]$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. *AUXDATA* series values in $[t_A^b, t_A^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

Case 5:



In this case, $[t_P^b, t_P^e] \cap [t_A^b, t_A^e] = \emptyset$: the auxiliary time span does not intersect the reference time span at all. *AUXDATA* series values in $[t_P^b, t_P^e]$ are set to missing values, and then those missing values are interpreted according to the value of the SETMISSING= option.

Table Output

The TSMODEL procedure can create the **OUT=**, **OUTARRAY=**, **OUTSCALAR=**, **OUTSUM=**, **OUTLOG=**, and **OUTOBJ=** tables. These tables always contain the variables that are specified in the BY statement. If a BY-group analysis step fails, then the values of this step are not recorded or are set to missing in the related output tables. Appropriate error or warning messages (or both) are recorded in the OUTLOG= table, subject to the value of the LOGCONTROL= option.

It is important to note that output tables created by PROC TSMODEL are naturally partitioned by the BY variables because of the way the BY groups are input into CAS session processes to form the time series data frames. The entire time series data for each BY group is input completely in a particular CAS session process. Consequently, all the time series processing for a particular BY group occurs in the context of a single CAS session node. Subsequent use of PROC TSMODEL to process the output tables makes use of this table-level partitioning.

The BY group partitioning of the output tables should not be confused with BY-group sort order or time ID sort order when the output tables are read by the CAS libname engine back into the SAS DATA step code for subsequent processing. If BY-group order is needed, or if time ID sort order within the BY groups is needed, you must specify PROC SORT steps in your SAS code to order the data set. For example, consider

the PROC TSMODEL step in the example “[Comparison of the TSMODEL and TIMEDATA Procedures](#)” on page 2068. In order to process the OUTSUM=, OUTSCALAR=, and OUTARRAY= tables in a SAS DATA step that expects the BY-group rows to be contiguous, you must specify BY Product in the SORT procedure to sort the tables mylib.Pricexsum and mylib.Pricexos. The table mylib.Pricexoa is a time series table, and you must specify BY Product Date in the SORT procedure to sort the data so that they are suitable for normal SAS time series processing. Failure to sort output tables that are generated by PROC TSMODEL can lead to errors when the tables are used as input in other procedures or steps.

OUT= Table

The **OUT=** table contains the variables that are specified in the BY, ID, or VAR statements. The ID variable values are aligned and extended based on the ALIGN=, INTERVAL=, and LEAD= options. The values of the variables specified in the VAR statements are accumulated based on the ACCUMULATE= option, and missing values are interpreted based on the SETMISSING= option.

OUTARRAY= Table

The **OUTARRAY=** table contains the variables that are specified in the BY, ID or VAR statements. If the ID statement is specified, then the ID variable values are aligned and extended based on the ALIGN= and INTERVAL= options. The values of the variables specified in the VAR statements are accumulated based on the ACCUMULATE= option, and missing values are interpreted based on the SETMISSING= option. In addition, the OUTARRAY= table contains the variables that are specified in the OUTARRAYS statements and the following variables:

STATUS	status flag that indicates whether the requested analyses were successful
TIMEID	time ID values
SEASON	season index values
CYCLE	life-cycle index values
Array-Variable-Names	variables that are specified in the OUTARRAYS statement

The OUTARRAY= table contains the arrays that are related to the (accumulated) time series.

OUTSCALAR= Table

The **OUTSCALAR=** table contains the variables that are specified in the BY statement. In addition, the table contains the variables that are specified in the OUTSCALARS statements and the following variables:

STATUS	status flag that indicates whether the requested analyses were successful
Scalar-Variable-Names	variables that are specified in the OUTSCALARS statement

The OUTSCALAR= table contains the scalars that are related to the (accumulated) time series.

OUTSUM= Table

The **OUTSUM=** table contains the variables that are specified in the BY statement and the variables in the following list. This table also records the descriptive statistics for each variable that is specified in a VAR statement. Variables that are related to descriptive statistics are based on the ACCUMULATE= and SETMISSING= options in the ID and VAR statements.

NAME	variable name
STATUS	status flag that indicates whether the requested analyses were successful
START	the starting date of each series
END	the ending date of each series
STARTOBS	the beginning observation number of each series
ENDOBS	the ending observation number of each series
NOBS	number of observations
N	number of nonmissing observations
NMISS	number of missing observations
MINIMUM	minimum value
MAXIMUM	maximum value
AVG	average value
STDDEV	standard deviation

OUTLOG= Table

The **OUTLOG=** table contains the variables that are specified in the BY statement and the variables in the following list. The **OUTLOG=** table records textual messages that arise from the processing of the BY group's time series data. Messages are filtered based on the value of the **LOGCONTROL=** option. In addition to the BY variables, the **OUTLOG=** table includes the following columns:

ERRNO	a numeric variable that stores the _ERRNO_ variable for the BY group. The value of _ERRNO_ might be set by the user-defined program directly or might be set implicitly by calling a function or method that sets the _ERRNO_ value.
LOGLEN	a numeric variable that stores the length of the _LOG_ variable text (byte count).
LOG	a character variable that stores the messages that are logged from the execution of the user-defined program on the BY group's time series data. All messages from the BY group are concatenated into the variable. End-of-line characters separate the individual messages. If PUTTOLOG=YES is specified in the PROC TSMODEL statement, the _LOG_ variable also contains messages from any PUT programming statements that are specified.

STATUS Variable Values

The `_STATUS_` variable that appears in the `OUTARRAY=`, `OUTSCALAR=` and `OUTSUM=` tables contains a value that specifies whether the analysis has been successful or not. The `_STATUS_` variable can take the following values:

0	Analysis was successful.
3000	Accumulation failed.
4000	Missing value interpretation failed.
6000	Series is all missing.
9000	Descriptive statistics could not be computed.

Printed Output

The `TSMODEL` procedure always prints a summary of the processing that is performed on the time series data. This is extremely useful for gauging the work that is performed by the CAS server when it executes `PROC TSMODEL`. Printing of other results is best accomplished by the use of targeted data queries to subset and display the information from the tables that are produced by the `TSMODEL` procedure. For example, you might want to print the `OUTARRAY=` results only for BY groups that have a particular `_STATUS_` value in the `OUTSUM=` table.

Examples: TSMODEL Procedure

Example 29.1: Accumulating Transactional Data into Time Series Data

This example uses the `TSMODEL` procedure to accumulate timestamped transactional data that have been recorded at no particular frequency into time series data at a specific frequency.

Suppose that the input table `mylib.retail` contains variables `Store` and `Timestamp` and numerous other numeric transaction variables. The BY variable `Store` contains values that divide the transactions into groups (BY groups). The time ID variable `Timestamp` contains SAS date values that are recorded at no particular frequency. The other variables, `Item1–Item8`, contain the numeric transaction values to be analyzed. The following statements form monthly time series from the transactional data.

These statements assume that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
proc tsmodel data=mylib.retail
    out=mylib.mseries(replace=yes);
by store;
id timestamp interval=month
    accumulate=avg
    setmiss=0
    start='01jan1998'd
```

```

            end   ='31dec2000'd;
    var item1-item8;
run;

```

The ACCUMULATE=AVG option in the ID statement accumulates the transactions that are recorded with each time period based on the average values. The SETMISS=0 option in the ID statement sets the accumulated time series values for time periods that have no transactions to 0 instead of to missing. The START='01JAN1998'D and END='31DEC2000'D options request that only transactions recorded between the first day of 1998 and the last day of 2000 be considered and, if needed, extended to include this range.

The monthly time series data are stored in the table mylib.mseries. Each BY group that is associated with the BY variable Store contains an observation for each of the 36 months that are associated with the years 1998, 1999, and 2000. Each observation contains the values Store, Timestamp, and each of the analysis variables Item1–Item8 in the input table.

The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in Output 29.1.1.

Output 29.1.1 Summary of Accumulation Processing
The TSMODEL Procedure

Processing Summary	
CAS table	RETAIL
Number of analysis variables	8
Number of rows read	400
Number of groups read	4
Memory for group packages (KB)	1
Time to load groups (seconds)	0.0077979565
Minimum time ID	NOV1998
Maximum time ID	DEC2000
Minimum time periods	36
Maximum time periods	36
Number of nodes run	1
Number of nodes with data	1
Number of nodes with groups	1
Number of threads budgeted	32
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	32
Maximum threads active	32
Number of groups processed by submitted code	4
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0124080181
Number of output table rows produced	144

After each set of transactions has been accumulated to form a corresponding time series, the accumulated time series can be analyzed using various time series analysis techniques. For example, exponentially weighted moving averages can be used to smooth each series.

If the time ID variable Timestamp contains SAS datetime values instead of SAS date values, the INTERVAL=, START=, and END= options must be changed accordingly and the following statements could be used:

```

proc tsmodel data=mylib.retail
  out=mylib.tseries(replace=yes);
  by store;
  id timestamp interval=dtmonth
      accumulate=avg
      setmiss=0
      start='01jan1998:00:00:00'dt
      end   ='31dec2000:00:00:00'dt;
  var item1-item8;
run;

```

The monthly time series data are stored in the table mylib.tseries, and the time ID values use a SAS datetime representation.

Example 29.2: User-Defined Program Statements

The following SAS macro statements create a user-defined subroutine and a user-defined function. Mylog is a subroutine that log-transforms a time series. Mymean is a function that computes the mean of a time series.

```

%macro fcmpcode;
  subroutine mylog(actual[*], transform[*]);
    outargs transform;
    actlen  = DIM(actual);
    do t = 1 to actlen;
      transform[t] = log(actual[t]);
    end;
  endsub;

  function mymean(z[*]);
    nz  = DIM(z);
    sum = 0;
    nnmiss = 0;
    do t = 1 to nz;
      if z[t] ne . then do;
        sum = sum + z[t];
        nnmiss = nnmiss + 1;
      end;
    end;
    if nnmiss eq 0 then return(.);

    return(sum/nnmiss);
  endsub;
%mend;

```

The following DATA step loads data set Sashehelp.Air, which contains airline data into the table mylib.air. These statements assume that your libref is named mylib, but you can substitute any appropriately defined libref.

```
data mylib.air(replace=yes);
  set sashelp.air;
run;
```

The input table mylib.air contains the variables Air and Date. The time series is recorded monthly.

The following statements form quarterly time series from the monthly series based on the median value of the total of the transactions recorded within each month. The OUTARRAYS statement specifies the Logair and Logairc arrays as output. The OUTSCALARS statement specifies the Meanlog scalars as output. The other arrays and scalars are not part of the output. The subsequent program statements create the output arrays and scalars.

```
proc tsmodel data=mylib.air
  outarray=mylib.arrays(replace=yes)
  outscalar=mylib.scalars(replace=yes)
  outsum=mylib.airsum(replace=yes);
  id date interval=qtr acc=total;
  var air;
  outarrays logair logairc;
  outscalars meanlog;
  submit;
    %fcmpcode;
    call mylog(air,logair);
    meanlog = mymean(air);
    do t = 1 to dim(air);
      logairc[t] = logair[t] - meanlog;
    end;
  endsubmit;
quit;
```

The following PROC PRINT step displays the OUTSUM= table, as shown in [Output 29.2.1](#).

```
proc print data=mylib.airsum; run;
```

Output 29.2.1 Summary Statistics for mylib.Air

Obs	_NAME_	NOBS	N	NMISS	MIN	MAX	MEAN	STDDEV
1	AIR	48	48	0	341	1736	840.89583333	356.43144067
2	logair	48	48	0	5.8318824773	7.4593388952	6.6430769906	0.4395319391
3	logairc	48	48	0	-835.0639509	-833.4364944	-834.2527563	0.4395319394

Obs	SUM	_STATUS_	START	END	STARTOBS	ENDOBS
1	40363	0	1949:1	1960:4	1	48
2	318.86769555	0	1949:1	1960:4	1	48
3	-40044.1323	0	1949:1	1960:4	1	48

Example 29.3: Using Auxiliary Tables

This example demonstrates the use of the `AUXDATA=` option in the `PROC TSMODEL` statement. The data set `Sashelp.Gulfoil` contains oil and gas production data from the Gulf of Mexico. The following `DATA` step loads data set `Sashelp.Gulfoil` into the table `mylib.gulfoil`.

These statements assume that your libref is named `mylib`, but you can substitute any appropriately defined libref.

```
data mylib.gulfoil(replace=yes);
    set sashelp.gulfoil;
run;
```

The variable `RegionName` define a time series hierarchy of interest. Suppose you want to generate two new series that contain the region's total share of oil and gas production for each month in the `mylib.gulfoil` table.

You first use `PROC TSMODEL` to perform temporal aggregation (accumulation) of the time series:

```
proc tsmodel data=mylib.gulfoil
    outarray=mylib.allreg(replace=yes);
    id date interval=month accumulate=total;
    var oil gas;
    outarray alloil allgas;
    submit;
        do t=1 to _length_;
            alloil[t]=oil[t];
            allgas[t]=gas[t];
        end;
    endsubmit;
quit;
```

In the preceding `PROC TSMODEL` statements, the Oil and Gas variables are accumulated across all regions because no `BY` statement is specified. In the following `DATA` step, the variables are dropped from the `mylib.allreg` table to avoid conflict with the original time series variables in the `mylib.gulfoil` table:

```
data mylib.allreg;
    set mylib.allreg;
    drop oil gas;
run;
```

You can then use `PROC TSMODEL` with the `AUXDATA=` option to compute the share of oil and gas production that is contributed by each region for each month. `PROC TSMODEL` reads a monthly time series for each `RegionName` group for the variables Oil and Gas from `mylib.Gulfoil`. Two new series are produced in the variables `Oilshare` and `Gasshare`, which contain the region's share of the oil and gas production, respectively. Those share variables are specified in the `OUTARRAY` statement for inclusion in the table (`mylib.shares`), which is specified in the `OUTARRAY=` option. The time series that are acquired for the variables `alloil` and `allgas` are common across all of the `RegionName` `BY` groups.

```
proc tsmodel data=mylib.gulfoil
    auxdata=mylib.allreg
    outarray=mylib.shares;
    by regionname;
    outarray oilshare gasshare;
```

```

var oil gas alloil allgas;
id date interval=month accumulate=total;
submit;
  do i=1 to _length_;
    oilshare[i] = oil[i] / alloil[i];
    gasshare[i] = gas[i] / allgas[i];
  end;
endsubmit;
quit;

```

The following code demonstrates that the computed shares sum to 1 for each time index in the resulting Oilshare and Gasshare series. PROC TSMODEL is used to accumulate the shares for these respective variables from the table mylib.shares, and the accumulated share series at the RegionName level are stored to the table mylib.rshares with variable names Oilsum and Gassum, respectively. The tables mylib.shares and mylib.rshares are shown in [Output 29.3.1](#).

```

proc tsmode data=mylib.shares
  outscalar=mylib.rshares(replace=yes)
  outsum=mylib.rsum(replace=yes);
id date interval=month accumulate=total;
var oilshare gasshare;
outscalar oilsum gassum;
submit;
  oilsum = 0;
  gassum = 0;
  do t=1 to _length_;
    oilsum = oilsum + oilshare[t];
    gassum = gassum + gasshare[t];
  end;
endsubmit;
quit;

proc print data=mylib.rshares; run;
proc print data=mylib.rsum; run;

```

Output 29.3.1 Validation of Oil and Gas Shares by Region

Obs _STATUS_ oilsum gassum													
1	0	123	123										

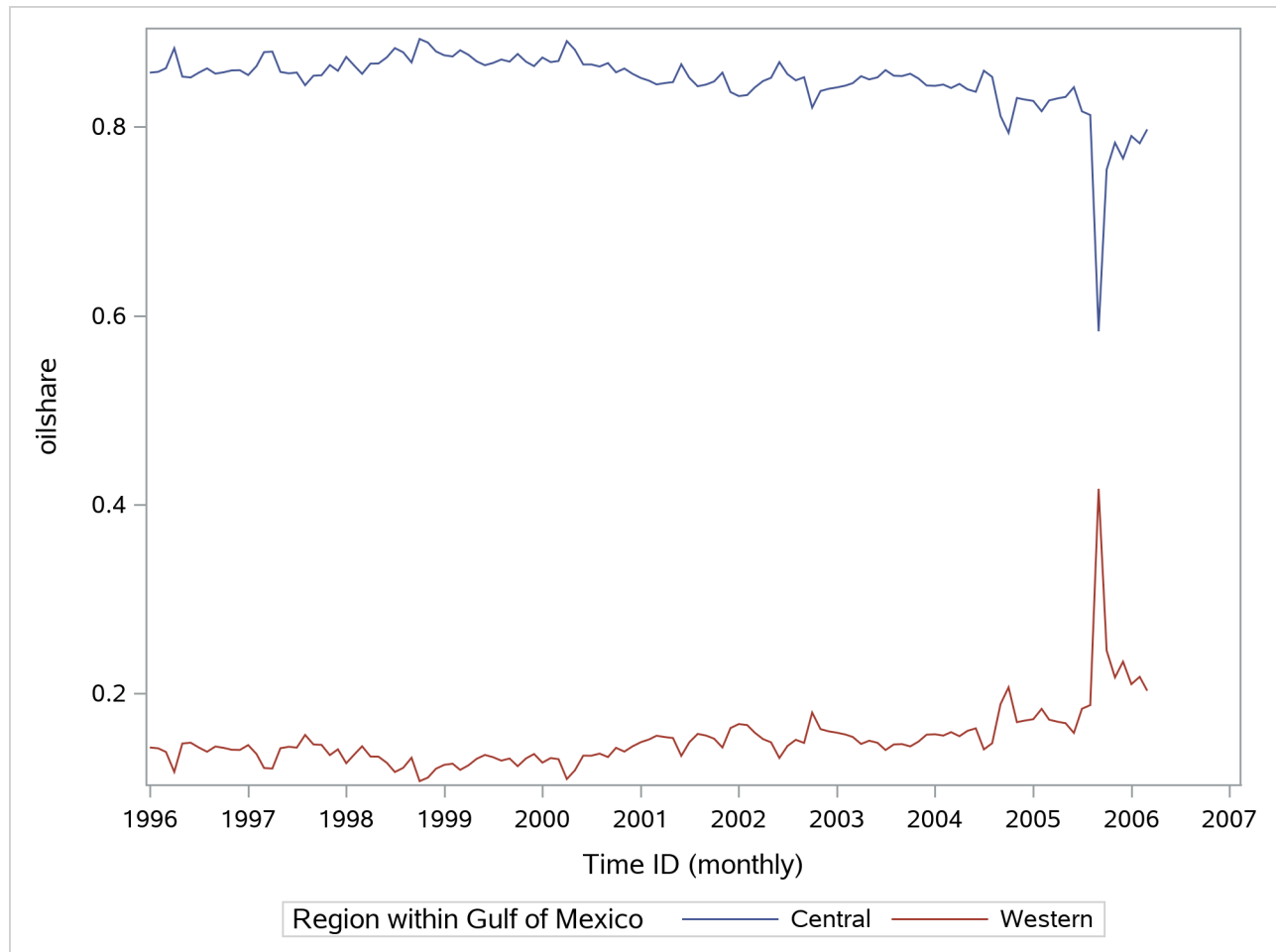
Obs	_NAME_	NOBS	N	NMISS	MIN	MAX	MEAN	STDDEV	SUM	_STATUS_	START	END	STARTOBS	ENDOBS
1	oilshare	123	123	0	1	1	1	0	123	0	JAN1996	MAR2006	1	123
2	gasshare	123	123	0	1	1	1	0	123	0	JAN1996	MAR2006	1	123

You might also want to plot the share series. The following code produces a graph that overlays the region level share series for oil production as shown in [Output 29.3.2](#).

```

proc sgplot data=mylib.shares;
  series x=Date y=OilShare/group=RegionName;
run;

```

Output 29.3.2 Region Share of Oil Production

References

- Keogh, E., Chu, S., Hart, D., and Pazzani, M. (2004). "Segmenting Time Series: A Survey and Novel Approach." In *Data Mining in Time Series Databases*, edited by M. Last, A. Kandel, and H. Bunke, 1–22. London: World Scientific. <https://www.worldscientific.com/worldscibooks/10.1142/5210>.
- National Retail Federation (2007). "National Retail Federation 4-5-4 Calendar." <http://www.nrf.com/>.
- Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). *ISO 8601:2004 Data Elements and Interchange Formats—Information Interchange—Representation of Dates and Times*. 3rd ed. Technical report, International Organization for Standardization.

Chapter 30

UCM Procedure

Contents

Overview: UCM Procedure	2113
PROC UCM in SAS Econometrics Compared with PROC UCM in SAS/ETS	2114
Using CAS Sessions and CAS Engine Librefs	2114
Getting Started: UCM Procedure	2115
A Seasonal Series with Linear Trend	2115
Syntax: UCM Procedure	2123
Functional Summary	2123
PROC UCM Statement	2126
AUTOREG Statement	2129
BLOCKSEASON Statement	2130
BY Statement	2132
CYCLE Statement	2132
DEPLAG Statement	2134
ESTIMATE Statement	2135
FORECAST Statement	2137
ID Statement	2140
IRREGULAR Statement	2140
ARMA Specification	2141
LEVEL Statement	2144
MODEL Statement	2145
NLOPTIONS Statement	2145
OUTLIER Statement	2146
RANDOMREG Statement	2146
SEASON Statement	2147
SLOPE Statement	2150
SPLINEREG Statement	2151
SPLINESEASON Statement	2152
TF Statement	2153
Details: UCM Procedure	2156
An Introduction to Unobserved Component Models	2156
Modeling the Trend	2157
Modeling a Cycle	2157
Modeling Seasons	2158
Modeling an Autoregression	2160
Modeling Regression Effects	2160
Modeling the Irregular Component	2160

The Model Parameters	2161
Model Specification	2161
UCMs as State Space Models	2162
Locally Linear Trend Model	2166
Basic Structural Model	2166
Seasons with Blocked Seasonal Values	2167
Cycles and Autoregression	2168
Incorporating Predictors of Different Types	2169
ARMA Irregular Component	2171
Models with Dependent Lags	2171
Outlier Detection	2172
Missing Values	2173
Parameter Estimation	2173
Parameter Estimation by Profile Likelihood Optimization	2174
<i>t</i> Values	2174
Computational Issues	2175
Convergence Problems	2175
Computer Resource Requirements	2175
Displayed Output	2175
Statistical Graphics	2176
Analysis of Sunspot Data: Illustration of ODS Graphics	2177
ODS Table Names	2186
ODS Graph Names	2190
OUTFOR= Data Set	2193
OUTEST= Data Set	2194
Statistics of Fit	2195
Examples: UCM Procedure	2196
Example 30.1: The Airline Series Revisited	2196
Example 30.2: Variable Star Data	2201
Example 30.3: Modeling Long Seasonal Patterns	2204
Example 30.4: Modeling Time-Varying Regression Effects	2209
Example 30.5: Trend Removal Using the Hodrick-Prescott Filter	2215
Example 30.6: Using Splines to Incorporate Nonlinear Effects	2217
Example 30.7: Detection of Level Shift	2222
Example 30.8: ARIMA Modeling	2226
Example 30.9: Extracting a Business Cycle	2229
Example 30.10: A Transfer-Function Model for the Italian Traffic Accident Data	2234
References	2239

Overview: UCM Procedure

The UCM procedure analyzes and forecasts equally spaced univariate time series data by using an unobserved components model (UCM). The UCMs are also called *structural models* in the time series literature. A UCM decomposes the response series into components such as trend, seasonals, cycles, and the regression effects due to predictor series. The components in the model are supposed to capture the salient features of the series that are useful in explaining and predicting its behavior. Harvey (1989) and Pelagatti (2015) are good references for time series modeling that use the UCMs. Harvey calls the components in a UCM the “stylized facts” about the series under consideration. Traditionally, the ARIMA models and, to some limited extent, the exponential smoothing models have been the main tools in the analysis of this type of time series data. It is fair to say that the UCMs capture the versatility of the ARIMA models while possessing the interpretability of the smoothing models. A thorough discussion of the correspondence between the ARIMA models and the UCMs, and the relative merits of UCM and ARIMA modeling, is given in Harvey (1989). The UCMs are also very similar to another set of models, called the *dynamic models*, that are popular in the Bayesian time series literature (West and Harrison 1999). In SAS Econometrics, you can use PROC CSSM for multivariate and more general univariate UCMs (see Chapter 14, “CSSM Procedure”), PROC ARIMA for ARIMA modeling (see Chapter 7, “ARIMA Procedure” (*SAS/ETS User’s Guide*)), and PROC ESM for exponential smoothing modeling (see Chapter 14, “ESM Procedure” (*SAS/ETS User’s Guide*)).

You can use the UCM procedure to fit a wide range of UCMs that can incorporate complex trend, seasonal, and cyclical patterns and can include multiple predictors. It provides a variety of diagnostic tools to assess the fitted model and to suggest the possible extensions or modifications. The components in the UCM provide a succinct description of the underlying mechanism governing the series. You can print, save, or plot the estimates of these component series. Along with the standard forecast and residual plots, the study of these component plots is an essential part of time series analysis using the UCMs. Once a suitable UCM is found for the series under consideration, it can be used for a variety of purposes. For example, it can be used for the following:

- forecasting the values of the response series and the component series in the model
- obtaining a model-based seasonal decomposition of the series
- obtaining a “denoised” version and interpolating the missing values of the response series in the historical period
- obtaining the full sample or “smoothed” estimates of the component series in the model

PROC UCM in SAS Econometrics Compared with PROC UCM in SAS/ETS

The UCM procedure in SAS Econometrics software is the CAS version of the UCM procedure in SAS/ETS software. This means that it requires SAS Cloud Analytic Services (CAS) in order to run. Except for this difference, both procedures are essentially identical in their modeling capabilities. Because it works in the cloud environment, PROC UCM in SAS Econometrics enables you to efficiently handle large numbers of BY groups by exploiting all available cores and concurrent threads to process them in parallel. These two procedures share virtually identical syntax and should produce virtually identical output. However, some important differences between them arise mainly because CAS tables do not use observation order and because some of the rarely needed and experimental features in PROC UCM in SAS/ETS are excluded from this release of PROC UCM in SAS Econometrics:

- The **ID** statement is required; it is optional in PROC UCM in SAS/ETS.
- All the input and output data sets, such as the input data set in the **DATA=** option in the PROC UCM statement, must be CAS tables.
- In the **NLOPTIONS** statement, you can currently specify only the **TECH=** and **MAXITER=** options; the other options remain at their default settings.
- The InitialParameters ODS table, which displays initial estimates of the free parameters and is part of the default output in PROC UCM in SAS/ETS, is not available. This table is rarely needed.
- The **PERFORMANCE** statement, which is unnecessary in CAS, is not available.
- The **BOOTSTRAP** option in the **FORECAST** statement, which generates the bootstrap prediction standard errors, is not available.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 54 in Chapter 4, “[Shared Concepts](#).”

Getting Started: UCM Procedure

The analysis of time series using the UCMs involves recognizing the salient features present in the series and modeling them suitably. The UCM procedure provides a variety of models for estimating and forecasting the commonly observed features in time series. These models are discussed in detail later in the section “[An Introduction to Unobserved Component Models](#)” on page 2156. First the procedure is illustrated using an example.

A Seasonal Series with Linear Trend

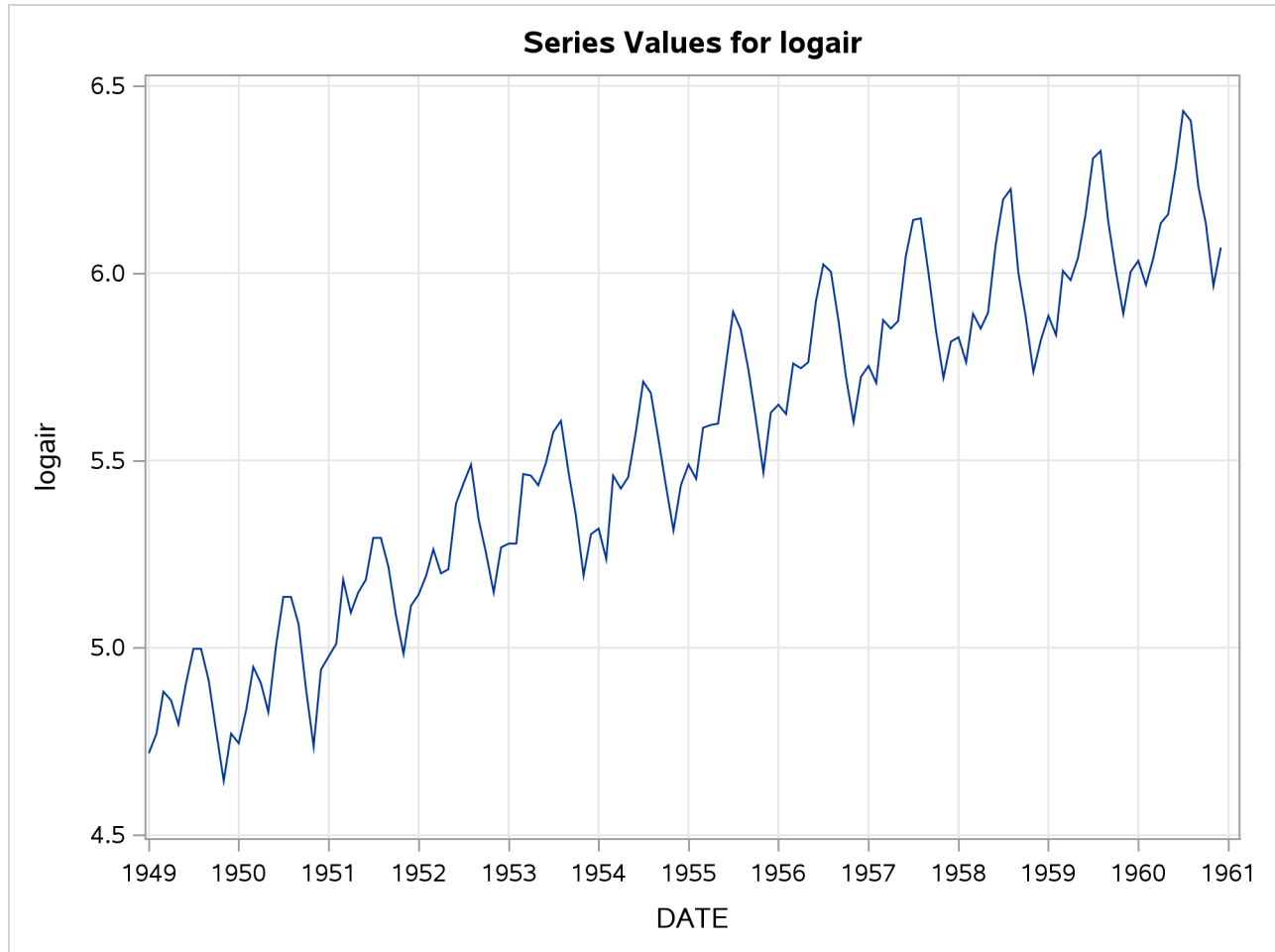
The airline passenger series, given as Series G in Box and Jenkins (1976), is often used in time series literature as an example of a nonstationary seasonal time series. This series is a monthly series consisting of the number of airline passengers who traveled during the years 1949 to 1960. Its main features are a steady rise in the number of passengers from year to year and the seasonal variation in the numbers during any given year. It also exhibits an increase in variability around the trend. A log transformation is used to stabilize this variability. The following DATA step prepares the log-transformed passenger series analyzed in this example:

```
data seriesG;
  set sashelp.air;
  logair = log( air );
run;
```

The following statements produce a time series plot of the series by using the TIMESERIES procedure (see Chapter 38, “TIMESERIES Procedure” (*SAS/ETS User’s Guide*)). The trend and seasonal features of the series are apparent in the plot in [Figure 30.1](#).

```
proc timeseries data=seriesG plot=series;
  id date interval=month;
  var logair;
```

```
run;
```

Figure 30.1 Series Plot of Log-Transformed Airline Passenger Series

In this example this series is modeled using an unobserved component model called the basic structural model (BSM). The BSM models a time series as a sum of three stochastic components: a trend component μ_t , a seasonal component γ_t , and random error ϵ_t . Formally, a BSM for a response series y_t can be described as

$$y_t = \mu_t + \gamma_t + \epsilon_t$$

Each of the stochastic components in the model is modeled separately. The random error ϵ_t , also called the *irregular component*, is modeled simply as a sequence of independent, identically distributed (iid) zero-mean Gaussian random variables. The trend and the seasonal components can be modeled in a few different ways. The model for trend used here is called a *locally linear time trend*. This trend model can be written as follows:

$$\begin{aligned}\mu_t &= \mu_{t-1} + \beta_{t-1} + \eta_t, & \eta_t &\sim \text{iid } N(0, \sigma_\eta^2) \\ \beta_t &= \beta_{t-1} + \xi_t, & \xi_t &\sim \text{iid } N(0, \sigma_\xi^2)\end{aligned}$$

These equations specify a trend where the level μ_t as well as the slope β_t is allowed to vary over time. This variation in slope and level is governed by the variances of the disturbance terms η_t and ξ_t in their respective

equations. Some interesting special cases of this model arise when you manipulate these disturbance variances. For example, if the variance of ξ_t is 0, the slope will be constant (equal to β_0); if the variance of η_t is also 0, μ_t will be a deterministic trend given by the line $\mu_0 + \beta_0 t$. The seasonal model that this example uses is called a trigonometric seasonal. The stochastic equations that govern a trigonometric seasonal are explained in the section “[Modeling Seasons](#)” on page 2158. However, it is worth noting here that this seasonal model reduces to the familiar regression with deterministic seasonal dummies if the variance of the disturbance terms in its equations is 0. The following statements specify a BSM that has these three components:

```
/* Load data into mylib */
data mylib.seriesG;
    set seriesG;
run;

proc ucm data=mylib.seriesG;
    id date interval=month;
    model logair;
    irregular;
    level;
    slope;
    season length=12 type=trig print=smooth;
    estimate;
    forecast lead=24 print=decomp;
run;
```

The PROC UCM statement signifies the start of the UCM procedure; the input SAS table, mylib.seriesG, which contains the dependent series, is specified there. The ID statement is used to specify a date, datetime, or time identification variable, which is called date in this example, to label the observations. The INTERVAL=MONTH option in the ID statement indicates that the measurements were collected on a monthly basis. The model specification begins with the MODEL statement, where the response series is specified (logair in this case). Then the components in the model are specified using separate statements that enable you to control their individual properties. The irregular component ϵ_t is specified using the IRREGULAR statement, and the trend component μ_t is specified using the LEVEL and SLOPE statements. The seasonal component γ_t is specified using the SEASON statement. The specifics of the seasonal characteristics, such as the season length, its stochastic evolution properties, and so on, are specified using the options in the SEASON statement. The seasonal component that this example uses has a season length of 12, corresponding to monthly seasonality, and is of the trigonometric type. The various types of seasonal components are explained in the section “[Modeling Seasons](#)” on page 2158.

The parameters of this model are the variances of the disturbance terms in the evolution equations of μ_t , β_t , and γ_t and the variance of the irregular component ϵ_t . These parameters are estimated by maximizing the likelihood of the data. The ESTIMATE statement options can be used to specify the span of data that are used in parameter estimation and to display and save the results of the estimation step and the model diagnostics. You can use the estimated model to obtain the forecasts of the series as well as the components. The options in the individual component statements can be used to display the component forecasts; for example, the PRINT=SMOOTH option in the SEASON statement displays smoothed forecasts of the seasonal component γ_t . The series forecasts and forecasts of the sum of components can be produced using the FORECAST statement. The PRINT=DECOMP option in the FORECAST statement prints the smoothed trend μ_t and the trend plus seasonal component ($\mu_t + \gamma_t$).

The parameter estimates for this model are displayed in [Figure 30.2](#).

Figure 30.2 BSM for the Logair Series**The UCM Procedure**

Final Estimates of the Free Parameters					
Component	Parameter	Estimate	Approx Std Error	t Value	Approx Pr > t
Irregular	Error Variance	0.00023436	0.0001	2.17	0.0298
Level	Error Variance	0.00029828	0.0001	2.82	0.0048
Slope	Error Variance	8.47913E-13	0.0000	0.00	0.9989
Season	Error Variance	0.00000356	0.0000	2.69	0.0072

The estimates suggest that except for the slope component, the disturbance variances of all the components are significant—that is, all these components are *stochastic*. The slope component, however, appears to be deterministic because its error variance is quite insignificant. It might then be useful to check if the slope component can be dropped from the model—that is, if $\beta_0 = 0$. This can be checked by examining the significance analysis table of the components given in Figure 30.3.

Figure 30.3 Component Significance Analysis for the Logair Series

Significance Analysis of Components (Based on the Final State)			
Component	DF	Chi-Square	Pr > ChiSq
Irregular	1	0.08	0.7747
Level	1	117867	<.0001
Slope	1	43.78	<.0001
Season	11	507.75	<.0001

This table provides the significance of the components in the model at the end of the estimation span. If a component is deterministic, this analysis is equivalent to checking whether the corresponding regression effect is significant. However, if a component is stochastic, then this analysis pertains only to the portion of the series near the end of the estimation span. In this example the slope appears quite significant and should be retained in the model, possibly as a deterministic component. Note that, on the basis of this table, the irregular component's contribution appears insignificant toward the end of the estimation span; however, since it is a stochastic component, it cannot be dropped from the model on the basis of this analysis alone. The slope component can be made deterministic by holding the value of its error variance fixed at 0. This is done by modifying the SLOPE statement as follows:

```
slope variance=0 noest;
```

After a tentative model is fit, its adequacy can be checked by examining different goodness-of-fit measures and other diagnostic tests and plots that are based on the model residuals. Once the model appears satisfactory, it can be used for forecasting. An interesting feature of the UCM procedure is that, apart from the series forecasts, you can request the forecasts of the individual components in the model. The plots of component forecasts can be useful in understanding their contributions to the series. The following statements illustrate some of these features:


```

proc ucm data=mylib.seriesG;
  id date interval = month;
  model logair;
  irregular;
  level plot=smooth;
  slope variance=0 noest;
  season length=12 type=trig
    plot=smooth;
  estimate;
  forecast lead=24 plot=decomp;
run;

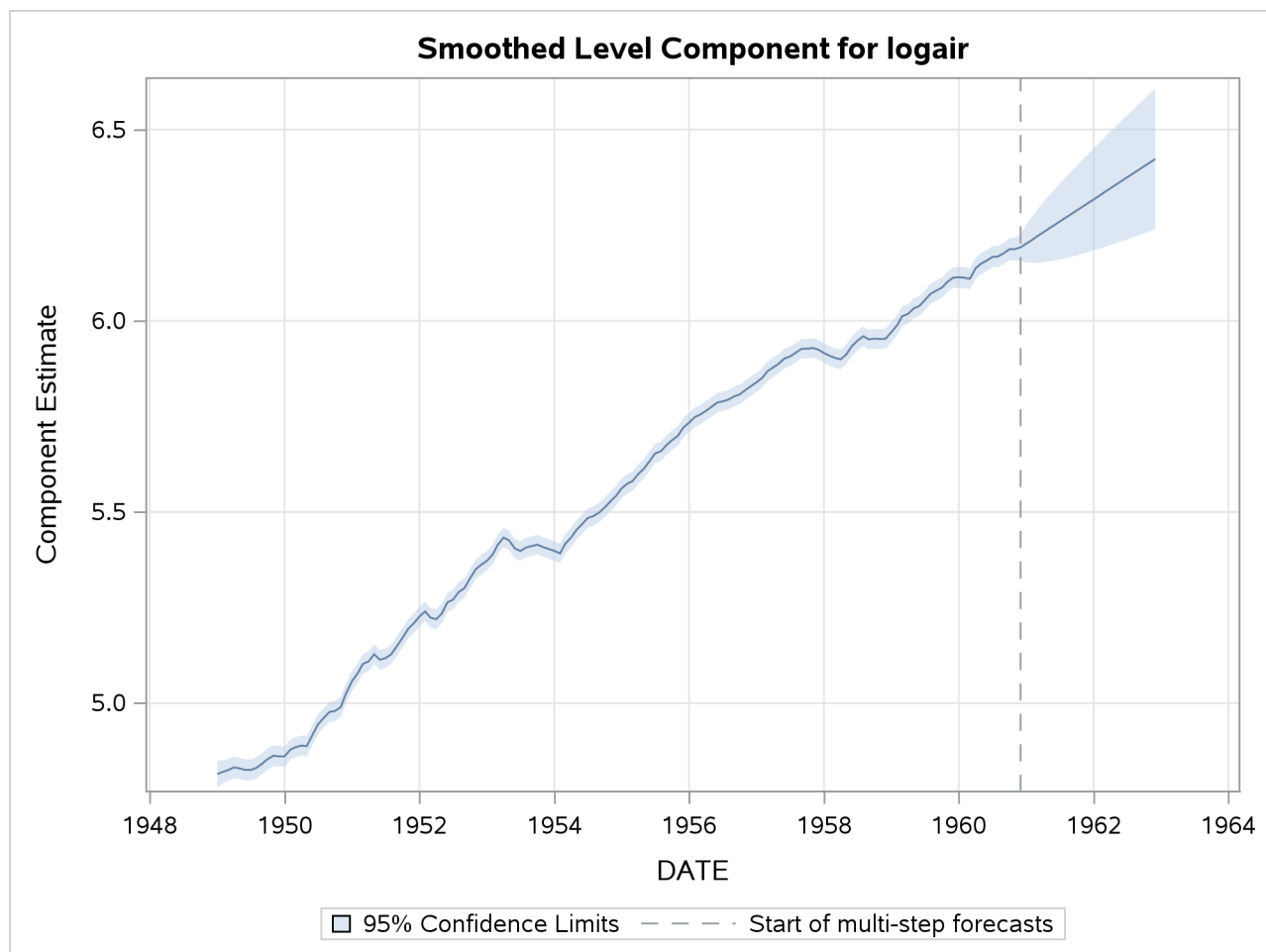
```

The table given in [Figure 30.4](#) shows the goodness-of-fit statistics that are computed by using the one-step-ahead prediction errors (see the section “[Statistics of Fit](#)” on page 2195). These measures indicate a good agreement between the model and the data. Additional diagnostic measures are also printed by default but are not shown here.

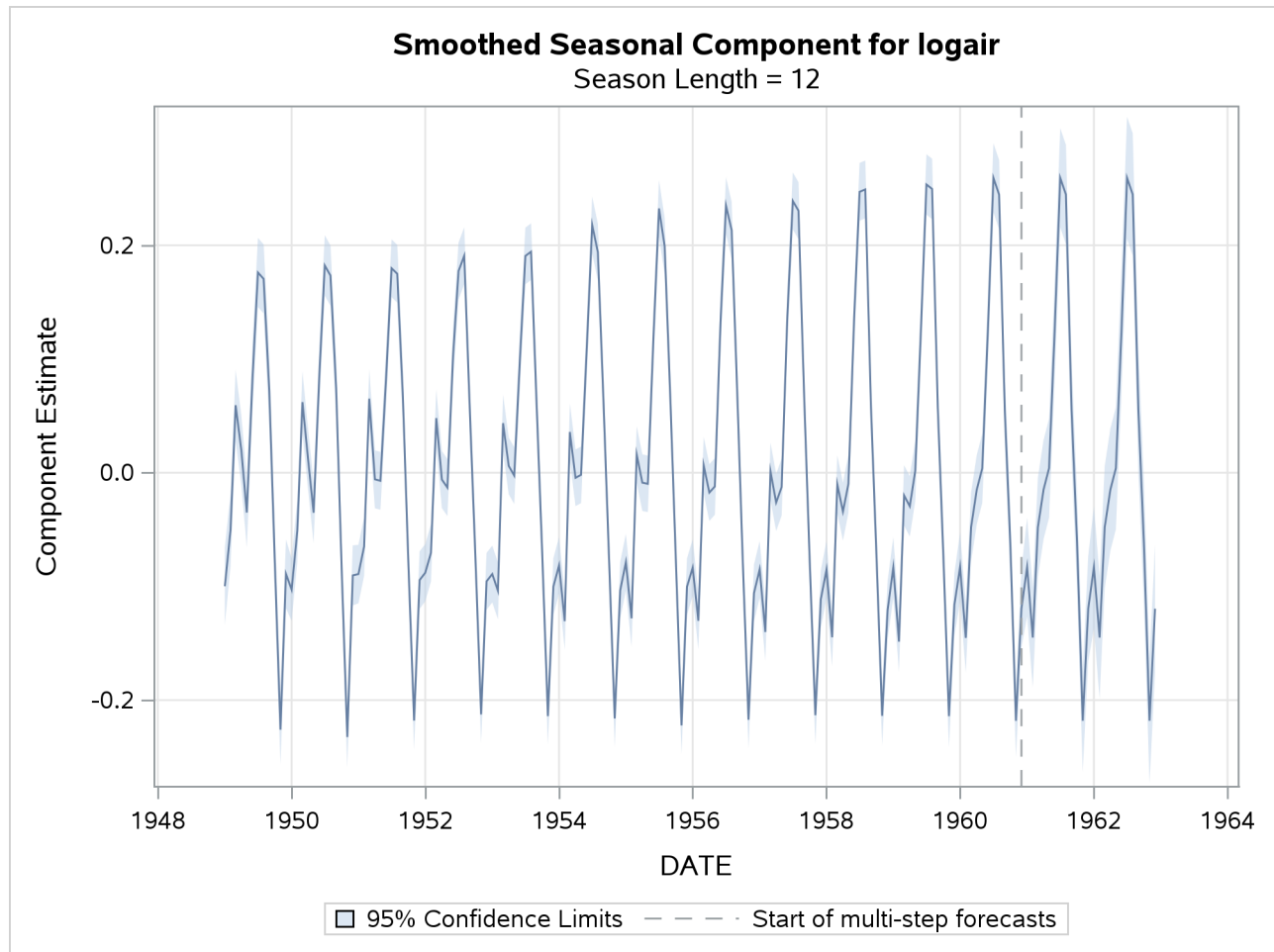
Figure 30.4 Fit Statistics for the Logair Series
The UCM Procedure

Fit Statistics Based on Residuals	
Mean Squared Error	0.00147
Root Mean Squared Error	0.03830
Mean Absolute Percentage Error	0.54132
Maximum Percent Error	2.19097
R-Square	0.99061
Adjusted R-Square	0.99046
Random Walk R-Square	0.87288
Amemiya's Adjusted R-Square	0.99017
Number of non-missing residuals used for computing the fit statistics = 131	

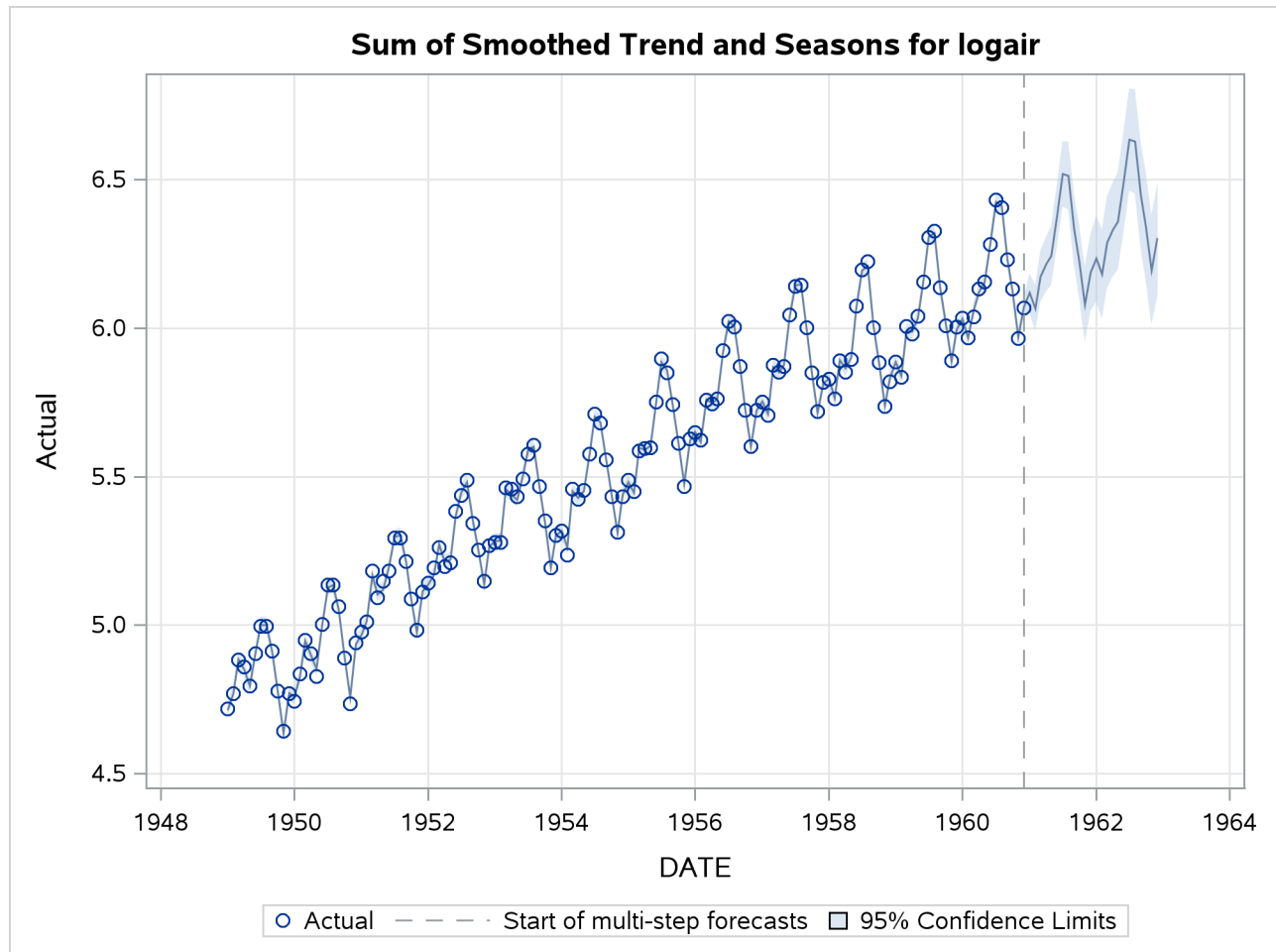
The first plot, shown in [Figure 30.5](#), is produced by the PLOT=SMOOTH option in the LEVEL statement, it shows the smoothed level of the series.

Figure 30.5 Smoothed Trend in the Logair Series

The second plot (Figure 30.6), produced by the PLOT=SMOOTH option in the SEASON statement, shows the smoothed seasonal component by itself.

Figure 30.6 Smoothed Seasonal in the Logair Series

The plot of the sum of the trend and seasonal component, produced by the PLOT=DECOMP option in the FORECAST statement, is shown in [Figure 30.7](#). You can see that, at least visually, the model seems to fit the data well. In all these decomposition plots the component estimates are extrapolated for two years in the future based on the LEAD=24 option specified in the FORECAST statement.

Figure 30.7 Smoothed Trend plus Seasonal in the Logair Series

Syntax: UCM Procedure

The UCM procedure uses the following statements:

```

PROC UCM < options > ;
  AUTOREG < options > ;
  BLOCKSEASON options ;
  BY variables ;
  CYCLE < options > ;
  DEPLAG options ;
  ESTIMATE < options > ;
  FORECAST < options > ;
  ID variable options ;
  IRREGULAR < options > ;
  LEVEL < options > ;
  MODEL dependent variable < = regressors > ;
  NLOPTIONS options ;
  OUTLIER options ;
  RANDOMREG regressors < / options > ;
  SEASON options ;
  SLOPE < options > ;
  SPLINEREG regressor < options > ;
  SPLINESEASON options ;
  TF regressor < options > ;

```

The **PROC UCM**, **ID**, and **MODEL** statements are required. In addition, the model must contain at least one component with nonzero disturbance variance.

Functional Summary

The statements and options controlling the UCM procedure are summarized in [Table 30.1](#). Most commonly needed scenarios are listed; see the individual statements for additional details. You can use the **PRINT=** and **PLOT=** options in the individual component statements for printing and plotting the corresponding component forecasts.

Table 30.1 Functional Summary

Description	Statement	Option
Data Set Options		
Specify the input SAS table	PROC UCM	DATA=
Write parameter estimates to an output SAS table	ESTIMATE	OUTEST=
Write series and component forecasts to an output SAS table	FORECAST	OUTFOR=

Table 30.1 *continued*

Description	Statement	Option
Model Specification		
Specify the dependent variable and simple predictors	MODEL	
Specify predictors with time-varying coefficients	RANDOMREG	
Specify a nonlinear predictor	SPLINEREG	
Specify the irregular component	IRREGULAR	
Specify the random walk trend	LEVEL	
Specify the locally linear trend	LEVEL and SLOPE	
Specify a cycle component	CYCLE	
Specify a dummy seasonal component	SEASON	TYPE=DUMMY
Specify a trigonometric seasonal component	SEASON	TYPE=TRIG
Drop some harmonics from a trigonometric seasonal component	SEASON	DROPH=
Specify a list of harmonics to keep in a trigonometric seasonal component	SEASON	KEEPH=
Specify a spline-season component	SPLINESEASON	
Specify a block-season component	BLOCKSEASON	
Specify an autoreg component	AUTOREG	
Specify the lags of the dependent variable	DEPLAG	
Specify a transfer function component	TF	
Controlling the Likelihood Optimization Process		
Request optimization of the profile likelihood	ESTIMATE	PROFILE
Request optimization of the usual likelihood	ESTIMATE	NOPROFILE
Specify the optimization technique	NLOPTIONS	TECH=
Limit the number of iterations	NLOPTIONS	MAXITER=
Outlier Detection		
Turn on the search for additive outliers		Default
Turn on the search for level shifts	LEVEL	CHECKBREAK
Specify the significance level for outlier tests	OUTLIER	ALPHA=
Limit the number of outliers	OUTLIER	MAXNUM=
Limit the number of outliers to a percentage of the series length	OUTLIER	MAXPCT=
Controlling the Series Span		
Exclude some initial observations from analysis during the parameter estimation	ESTIMATE	SKIPFIRST=
Exclude some observations at the end from analysis during the parameter estimation	ESTIMATE	BACK=
Exclude some initial observations from analysis during forecasting	FORECAST	SKIPFIRST=

Table 30.1 *continued*

Description	Statement	Option
Exclude some observations at the end from analysis during forecasting	FORECAST	BACK=
Graphical Residual Analysis		
Get a panel of plots consisting of residual autocorrelation plots and residual normality plots	ESTIMATE	PLOT=PANEL
Get the residual CUSUM plot	ESTIMATE	PLOT=CUSUM
Get the residual cumulative sum of squares plot	ESTIMATE	PLOT=CUSUMSQ
Get a plot of p -values for the portmanteau white-noise test	ESTIMATE	PLOT=WN
Get a time series plot of residuals with overlaid loess smoother	ESTIMATE	PLOT=LOESS
Series Decomposition and Forecasting		
Specify the number of periods to forecast in the future	FORECAST	LEAD=
Specify the significance level of the forecast confidence interval	FORECAST	ALPHA=
Prints smoothed series decomposition	FORECAST	PRINT=DECOMP
Prints one-step-ahead and multistep-ahead forecasts	FORECAST	PRINT=FORECASTS
Prints smoothed series decomposition	FORECAST	PLOT=DECOMP
Plots one-step-ahead and multistep-ahead forecasts	FORECAST	PLOT=FORECASTS
BY Groups		
Specify BY-group processing	BY	
Global Printing and Plotting Options		
Turn off all the printing for the procedure	PROC UCM	NOPRINT
Turn on all the printing options for the procedure	PROC UCM	PRINTALL
Turn off all the plotting for the procedure	PROC UCM	PLOTS=NONE
Turn on all the plotting options for the procedure	PROC UCM	PLOTS=ALL
Turn on a variety of plotting options for the procedure	PROC UCM	PLOTS=
ID		
Specify a variable that provides the time index for the series values	ID	

PROC UCM Statement

PROC UCM <options> ;

The PROC UCM statement invokes the UCM procedure. You can specify the following options:

DATA=*libref.data-table*

specifies the name of the SAS table that contains the time series. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section “Using CAS Sessions and CAS Engine Librefs” on page 2114.

NOPRINT

turns off all printing for the procedure. Subsequent print options in the procedure are ignored.

PLOTS<(global-plot-options)> <= plot-request <(options)>>

PLOTS<(global-plot-options)> <= (plot-request <(options)> <...plot-request <(options)>>>

controls the plots that are produced using ODS Graphics. When you specify only one *plot-request*, you can omit the parentheses around it.

Here are some examples:

```
plots=none
plots=all
plots=residuals(acf loess)
plots(noclm)=(smooth(decomp) residual(panel loess))
```

For general information about ODS Graphics, see Chapter 24, “Statistical Graphics Using ODS” (*SAS/STAT User’s Guide*).

```
proc ucm;
  model y = x;
  irregular;
  level;
run;

proc ucm plots=all;
  model y = x;
  irregular;
  level;
run;
```

The first PROC UCM step does not specify the PLOTS= option, so the default plot that displays the series forecasts in the forecast region is produced. The PLOTS=ALL option in the second PROC UCM step produces all the plots that are appropriate for the specified model.

In addition to the PLOTS= option in the PROC UCM statement, you can produce plots by using the PLOT= option in other statements of the UCM procedure. This way of producing plots provides finer

control over the plot production. If you do not specify any specific plot request, then PROC UCM produces the plot of series forecasts in the forecast horizon by default.

Global Plot Options

The *global-plot-options* apply to all relevant plots generated by the UCM procedure. You can specify the following *global-plot-option*:

NOCLM

suppresses the confidence limits in all the component and forecast plots.

Specific Plot Options

The following list describes the specific plots that you can specify and their options:

ALL

produces all plots appropriate for the particular analysis.

FILTER (< filter-plot-options >)

produces time series plots of the filtered component estimates. You can specify the following *filter-plot-options*:

ALL

produces all the filtered component estimate plots appropriate for the particular analysis.

CYCLE

produces time series plots of the filtered cycle component estimates for all cycle components in the model, if there are any.

DECOMP

produces time series plots of the filtered estimates of the series decomposition.

LEVEL

produces a time series plot of the filtered level component estimate, provided that the model contains the level component.

SEASON

produces time series plots of the filtered season component estimates for all seasonal components in the model, if there are any.

SLOPE

produces a time series plot of the filtered slope component estimate, provided that the model contains the slope component.

RESIDUAL (< residual-plot-options >)

produces the residuals plots. You can specify the following *residual-plot-options*:

ALL

produces all the residual diagnostics plots appropriate for the particular analysis.

ACF

produces the autocorrelation plot of residuals.

CUSUM

produces the plot of cumulative residuals against time.

CUSUMSQ

produces the plot of cumulative squared residuals against time.

HISTOGRAM

produces the histogram of residuals.

LOESS

produces a scatter plot of residuals against time, which has an overlaid loess fit.

PACF

produces the partial-autocorrelation plot of residuals.

PANEL

produces a summary panel of the residual diagnostics that consists of the following:

- histogram of residuals
- normal quantile plot of residuals
- residual-autocorrelation plot
- residual-partial-autocorrelation plot

QQ

produces a normal quantile plot of residuals.

RESIDUAL

produces a needle plot of residuals against time.

WN

produces the plot of Ljung-Box white-noise test p -values at different lags (in log scale).

SMOOTH (< smooth-plot-options >)

produces time series plots of the smoothed component estimates. You can specify the following *smooth-plot-options*:

ALL

produces all the smoothed component estimate plots appropriate for the particular analysis.

CYCLE

produces time series plots of the smoothed cycle component estimates for all cycle components in the model, if there are any.

DECOMP

produces time series plots of the smoothed estimates of the series decomposition.

LEVEL

produces a time series plot of the smoothed level component estimate, provided that the model contains the level component.

SEASON

produces time series plots of the smoothed season component estimates for all season components in the model, if there are any.

SLOPE

produces a time series plot of the smoothed slope component estimate, provided that the model contains the slope component.

NONE

suppresses all plots.

PRINTALL

turns on all the printing options for the procedure. Any subsequent NOPRINT options in the procedure are ignored.

AUTOREG Statement

AUTOREG <options> ;

The AUTOREG statement specifies an autoregressive component in the model. An autoregressive component is a special case of cycle that corresponds to the frequency of 0 or π . It is modeled separately for easier interpretation. A stochastic equation for an autoregressive component r_t can be written as follows:

$$r_t = \rho r_{t-1} + v_t, \quad v_t \sim \text{iid } N(0, \sigma_v^2)$$

The damping factor ρ can take any value in the interval $(-1, 1)$, including -1 but excluding 1 . If $\rho = 1$, the autoregressive component cannot be distinguished from the random walk level component. If $\rho = -1$, the autoregressive component corresponds to a seasonal component with a season length of 2, or a nonstationary cycle with period 2. If $|\rho| < 1$, then the autoregressive component is stationary. The following example illustrates the AUTOREG statement. This statement includes an autoregressive component in the model. The damping factor ρ and the disturbance variance σ_v^2 are estimated from the data.

autoreg;

You can specify the following *options*:

NOEST=RHO**NOEST=VARIANCE****NOEST=(RHO VARIANCE)**fixes the values of ρ and σ_v^2 to those specified in the **RHO=** and **VARIANCE=** options.**PLOT=FILTER****PLOT=SMOOTH****PLOT=(< FILTER > < SMOOTH >)**

plots the filtered or smoothed estimate of the autoreg component.

PRINT=FILTER**PRINT=SMOOTH****PRINT=(< FILTER > < SMOOTH >)**

prints the filtered or smoothed estimate of the autoreg component.

RHO=valuespecifies an initial value for the damping factor ρ during the parameter estimation process. The value of ρ must be in the interval $(-1, 1)$, including -1 but excluding 1 .**VARIANCE=value**specifies an initial value for the disturbance variance σ_v^2 during the parameter estimation process. Any nonnegative value, including 0 , is an acceptable starting value.

BLOCKSEASON Statement

BLOCKSEASON NBLOCKS=integer BLOCKSIZE=integer < options > ;

The **BLOCKSEASON** or **BLOCKSEASONAL** statement is used to specify a seasonal component γ_t that has a special block structure. The seasonal γ_t is called a *block seasonal* of block size m and number of blocks k if its season length, s , can be factored as $s = m * k$ and its seasonal effects have a block form—that is, the first m seasonal effects are all equal to some number τ_1 , the next m effects are all equal to some number τ_2 , and so on.

This type of seasonal structure can be appropriate in some cases; for example, consider a series that is recorded on an hourly basis. Further assume that, in this particular case, the hour-of-the-day effect and the day-of-the-week effect are additive. In this situation the hour-of-the-week seasonality, having a season length of 168 , can be modeled as a sum of two components. The hour-of-the-day effect is modeled using a simple seasonal of season length 24 , while the day-of-the-week is modeled as a block seasonal component that has the days of the week as blocks. This day-of-the-week block seasonal component has seven blocks, each of size 24 .

A block seasonal specification requires, at the minimum, the block size m and the number of blocks in the seasonal k . These are specified using the **BLOCKSIZE=** and **NBLOCKS=** option, respectively. In addition, you might need to specify the position of the first observation of the series by using the **OFFSET=** option if it is not at the beginning of one of the blocks. In the example just considered, this corresponds to a situation where the first series measurement is not at the start of the day. Suppose that the first measurement of the series corresponds to the hour between 6:00 and 7:00 a.m., which is the seventh hour within that day or at the seventh position within that block. This is specified as **OFFSET=7**.

The other options in this statement are very similar to the options in the SEASON statement; for example, a block seasonal can also be of one of the two types, DUMMY and TRIG. There can be more than one block seasonal component in the model, each specified using a separate BLOCKSEASON statement. No two block seasonals in the model can have the same NBLOCKS= and BLOCKSIZE= specifications. The following example illustrates the use of the BLOCKSEASON statement to specify the additive, hour-of-the-week seasonal model:

```
season length=24 type=trig;
blockseason nblocks=7 blocksize=24;
```

You must specify the following *options*:

BLOCKSIZE=integer

specifies the block size, m . The block size can be any integer greater than or equal to 2. Typical examples of block sizes are 24, corresponding to the hours of the day when a day is being used as a block in hourly data, or 60, corresponding to the minutes in an hour when an hour is being used as a block in data recorded by minutes, and so on.

NBLOCKS=integer

specifies the number of blocks, k . The number of blocks can be any integer greater than or equal to 2.

You can also specify the following *options*:

NOEST

fixes the value of the disturbance variance parameter to the value specified in the **VARIANCE=** option.

OFFSET=integer

specifies the position of the first measurement within the block, if the first measurement is not at the start of a block. The OFFSET= value must be between 1 and the block size. By default, OFFSET=1. The first measurement refers to the start of the estimation span and the forecast span. If these spans differ, their starting measurements must be separated by an integer multiple of the block size.

PLOT=FILTER

PLOT=SMOOTH

PLOT=F_ ANNUAL

PLOT=S_ ANNUAL

PLOT=(< plot-request > ... < plot-request >)

plots the season component. When you specify only one *plot-request*, you can omit the parentheses around it. You can use the FILTER and SMOOTH options to plot the filtered and smoothed estimates of the season component γ_t . You can use the F_ ANNUAL and S_ ANNUAL options to get the plots of “annual” variation in the filtered and smoothed estimates of γ_t . The annual plots are useful to see the change in the contribution of a particular month over the span of years. Here “month” and “year” are generic terms that change appropriately with the interval type being used to label the observations and the season length. For example, for monthly data with a season length of 12, the usual meaning applies, while for daily data with a season length of 7, the days of the week serve as months and the weeks serve as years. The first period in each block is plotted over the years.

PRINT=FILTER**PRINT=SMOOTH****PRINT=(< FILTER > < SMOOTH >)**prints the filtered or smoothed estimate of the block seasonal component γ_t .**TYPE=DUMMY | TRIG**

specifies the type of the block seasonal component. By default, TYPE=DUMMY.

VARIANCE=valuespecifies an initial value for the disturbance variance, σ_ω^2 , in the γ_t equation at the start of the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

BY Statement

BY variables ;

You can use a BY statement in the UCM procedure to process a data set in groups of observations defined by the BY variables. The model that you specify using the MODEL statement and other component statements is applied to all the groups defined by the BY variables. The variables are one or more variables in the input SAS table.

CYCLE Statement

CYCLE < options > ;

The CYCLE statement is used to specify a cycle component, ψ_t , in the model. The stochastic equation governing a cycle component of period p and damping factor ρ is

$$\begin{bmatrix} \psi_t \\ \psi_t^* \end{bmatrix} = \rho \begin{bmatrix} \cos \lambda & \sin \lambda \\ -\sin \lambda & \cos \lambda \end{bmatrix} \begin{bmatrix} \psi_{t-1} \\ \psi_{t-1}^* \end{bmatrix} + \begin{bmatrix} \nu_t \\ \nu_t^* \end{bmatrix}$$

where ν_t and ν_t^* are independent, zero-mean, Gaussian disturbances with variance σ_ν^2 and $\lambda = 2 * \pi / p$ is the angular frequency of the cycle. Any p strictly greater than 2 is an admissible value for the period, and the damping factor ρ can be any value in the interval (0, 1), including 1 but excluding 0. The cycles with frequency 0 and π , which correspond to the periods equal to infinity and 2, respectively, can be specified using the AUTOREG statement. The values of ρ less than 1 give rise to a stationary cycle, while $\rho = 1$ gives rise to a nonstationary cycle. As a default, values of ρ , p , and σ_ν^2 are estimated from the data. However, if necessary, you can fix the values of some or all of these parameters.

There can be multiple cycles in a model, each specified using a separate CYCLE statement. The examples that follow illustrate the use of the CYCLE statement.

The following statements add two cycles in the model. The parameters of each of these cycles are estimated from the data.

```
cycle;
cycle;
```

The following statement adds a nonstationary cycle in the model. The cycle period p and the disturbance variance σ_v^2 are estimated from the data.

```
cycle rho=1 noest=rho;
```

In the following statement, a nonstationary cycle with a fixed period of 12 is specified. Moreover, a starting value is supplied for σ_v^2 .

```
cycle period=12 rho=1 variance=4 noest=(rho period);
```

You can specify the following *options*:

NOEST=PERIOD

NOEST=RHO

NOEST=VARIANCE

NOEST=(< RHO > < PERIOD > < VARIANCE >)

fixes the values of the component parameters to those specified in the **RHO=**, **PERIOD=**, and **VARIANCE=** options. This option enables you to fix any combination of parameter values.

ORDER=integer

enables you to specify a higher-order cycle. A higher-order cycle (a cycle whose order is greater than 1) is a generalization of the stochastic cycle described at the beginning of this section, which can be thought of as a first-order cycle. Higher-order cycles are well explained in Trimbur (2005) and Pelagatti (2015, sect. 3.3.3). A cycle whose order is greater than 2 is rarely needed, and specifying cycles of large orders (for example, an order greater than 4) can lead to computational instability. See [Example 30.9](#) for an example of the use of higher-order cycles.

PERIOD=value

specifies an initial value for the cycle period during the parameter estimation process. Period value must be strictly greater than 2.

PLOT=FILTER

PLOT=SMOOTH

PLOT=(< FILTER > < SMOOTH >)

plots the filtered or smoothed estimate of the cycle component.

PRINT=FILTER

PRINT=SMOOTH

PRINT=(< FILTER > < SMOOTH >)

prints a filtered or smoothed estimate of the cycle component ψ_t .

RHO=value

specifies an initial value for the damping factor in this component during the parameter estimation process. Any value in the interval (0, 1), including 1 but excluding 0, is an acceptable initial value for the damping factor.

VARIANCE=value

specifies an initial value for the disturbance variance parameter, σ_v^2 , to be used during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

DEPLAG Statement

DEPLAG LAGS=order < PHI=value ... > NOEST > ;

The DEPLAG statement is used to specify the lags of the dependent variable to be included as predictors in the model. The following examples illustrate the use of the DEPLAG statement.

If the dependent series is denoted by y_t , the following statement specifies the inclusion of $\phi_1 y_{t-1} + \phi_2 y_{t-2}$ in the model. The parameters ϕ_1 and ϕ_2 are estimated from the data.

```
deplag lags=2;
```

The following statement includes $\phi_1 y_{t-1} + \phi_2 y_{t-4} - \phi_1 \phi_2 y_{t-5}$ in the model. The values of ϕ_1 and ϕ_2 are fixed at 0.8 and -1.2, respectively.

```
deplag lags=(1) (4) phi=0.8 -1.2 noest;
```

The dependent lag parameters are not constrained to lie in any particular region. In particular, this implies that a UCM that contains only an irregular component and dependent lags, resulting in a traditional autoregressive model, is not constrained to be a stationary model. In the DEPLAG statement, if an initial value is supplied for any one of the parameters, the initial values must also be supplied for all other parameters.

You must specify the following option:

LAGS=order

LAGS=(lag, ..., lag) ... (lag, ..., lag)

specifies lags of the dependent variable. $\text{LAGS}=(l_1, l_2, \dots, l_k)$ defines a model with specified lags of the dependent variable included as predictors. LAGS=order is equivalent to $\text{LAGS}=(1, 2, \dots, \text{order})$.

A concatenation of parenthesized lists specifies a factored model. For example, $\text{LAGS}=(1)(12)$ specifies that the lag values, 1, 12, and 13, corresponding to the following polynomial in the backward shift operator, be included in the model:

$$(1 - \phi_{1,1}B)(1 - \phi_{2,1}B^{12})$$

Note that, in this case, the coefficient of the 13th lag is constrained to be the product of the coefficients of the 1st and 12th lags.

You can also specify the following *options*:

NOEST

fixes the values of the parameters to those specified in **PHI=** option.

PHI=value ...

lists starting values for the coefficients of the lagged dependent variable. The order of the values listed corresponds with the order of the lags specified in the **LAGS=** option.

ESTIMATE Statement

ESTIMATE <options> ;

The ESTIMATE statement is an optional statement used to control the overall model-fitting environment. Using this statement, you can control the span of observations used to fit the model by using the **SKIPFIRST=** and **BACK=** options. This can be useful in model diagnostics. You can request a variety of goodness-of-fit statistics and other model diagnostic information including different residual diagnostic plots. Note that the ESTIMATE statement is not used to control the nonlinear optimization process itself. That is done using the **NLOPTIONS** statement, where you can control the number of iterations, choose between the different optimization techniques, and so on. You can save the estimated parameters and other related information in a SAS table by using the **OUTEST=** option. You can request the optimization of the profile likelihood, the likelihood obtained by concentrating out a disturbance variance, for parameter estimation by using the **PROFILE** option. The following example illustrates the use of this statement:

```
estimate skipfirst=12 back=24;
```

This statement requests that the initial 12 measurements and the last 24 measurements be excluded during the model-fitting process. The actual observation span used to fit the model is decided as follows: Suppose that n_0 and n_1 are the observation numbers of the first and last nonmissing values of the response variable, respectively. As a result of specifying **SKIPFIRST=12** and **BACK=24**, the measurements between observation numbers $n_0 + 12$ and $n_1 - 24$ form the estimation span. Of course, the model fitting might not take place if there are insufficient data in the resulting span. The model fitting does not take place if there are regressors in the model that have missing values in the estimation span.

You can specify the following *options*:

BACK=integer

SKIPLAST=integer

indicates that some ending part of the data needs to be ignored during the parameter estimation. This can be useful when you want to study the forecasting performance of a model on the observed data. When **BACK=10**, the last 10 measurements of the response series are skipped during the parameter estimation. By default, **BACK=0**.

EXTRADIFFUSE=k

enables continuation of the diffuse filtering iterations for k additional iterations beyond the first instance where the initialization of the diffuse state would have otherwise taken place. If the specified k is larger than the sample size, the diffuse iterations continue until the end of the sample. Note that one-step-ahead residuals are produced only after the diffuse state is initialized. Delaying the initialization leads to a reduction in the number of one-step-ahead residuals available for computing the residual diagnostic measures. This option is useful when you want to ignore the first few one-step-ahead residuals that often have large variance.

LIKE=DIFFUSE | MARGINAL

specifies the type of likelihood to use for parameter estimation. You can specify the following values:

DIFFUSE	uses diffuse likelihood.
MARGINAL	uses marginal likelihood.

By default, LIKE=DIFFUSE. For more information about likelihood types, see the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769 in Chapter 14, “[CSSM Procedure](#).” For an example of the use of LIKE=MARGINAL option, see [Example 30.10](#).

NOPROFILE

requests that the usual likelihood be optimized for parameter estimation. For more information, see the section “[Parameter Estimation by Profile Likelihood Optimization](#)” on page 2174.

OUTEST=libref.data-table

specifies an output SAS table for the estimated parameters. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 2114.

PLOT=plot-request**PLOT=(<plot-request> ... <plot-request>)**

produces residual diagnostic plots. You can specify the following *plot-requests*:

ACF

produces a residual-autocorrelation plot.

CUSUM

produces a plot of cumulative residuals against time.

CUSUMSQ

produces a plot of cumulative squared residuals against time.

HISTOGRAM

produces a histogram of residuals.

LOESS

produces a scatter plot of residuals against time, which has an overlaid loess fit.

MODEL

produces a plot of one-step-ahead forecasts in the estimation span.

PACF

produces a residual-partial-autocorrelation plot.

PANEL

produces a summary panel of the residual diagnostics that consists of the following:

- histogram of residuals
- normal quantile plot of residuals
- residual-autocorrelation plot
- residual-partial-autocorrelation plot

QQ

produces a normal quantile plot of residuals.

RESIDUAL

produces a needle plot of residuals against time.

WN

produces a plot of p -values, in log scale, at different lags for the Ljung-Box portmanteau white-noise test statistics.

PRINT=NONE

suppresses all the printed output related to the model fitting, such as the parameter estimates, the goodness-of-fit statistics, and so on.

PROFILE

requests that the profile likelihood, obtained by concentrating out one of the disturbance variances from the likelihood, be optimized for parameter estimation. By default, the profile likelihood is not optimized if any of the disturbance variance parameters is held fixed to a nonzero value. For more information see the section “[Parameter Estimation by Profile Likelihood Optimization](#)” on page 2174.

SKIPFIRST=integer

indicates that some early part of the data needs to be ignored during the parameter estimation. This can be useful if there is a reason to believe that the model being estimated is not appropriate for this portion of the data. When SKIPFIRST=10, the first 10 measurements of the response series are skipped during the parameter estimation. By default, SKIPFIRST=0.

FORECAST Statement

FORECAST <options> ;

The FORECAST statement is an optional statement that is used to specify the overall forecasting environment for the specified model. It can be used to specify the span of observations, the historical period, to be used to compute the forecasts of the future observations. This is done using the SKIPFIRST= and BACK= options. The number of periods to forecast beyond the historical period, and the significance level of the forecast confidence interval, is specified using the LEAD= and ALPHA= options. You can request one-step-ahead series and component forecasts by using the PRINT= option. You can save the series forecasts, and the model-based decomposition of the series, in a SAS table by using the OUTFOR= option. The following example illustrates the use of this statement:

```
forecast skipfirst=12 back=24 lead=30;
```

This statement requests that the initial 12 and the last 24 response values be excluded during the forecast computations. The forecast horizon, specified using the LEAD= option, is 30 periods; that is, multistep forecasting begins at the end of the historical period and continues for 30 periods. The actual observation span that is used to compute the multistep forecasting is determined as follows: Suppose that n_0 and n_1 are the observation numbers of the first and the last nonmissing values of the response variable, respectively. As a result of specifying SKIPFIRST=12 and BACK=24, the historical period, or the forecast span, begins at $n_0 + 12$ and ends at $n_1 - 24$. Multistep forecasts are produced for the next 30 periods—that is, for the

observation numbers $n_1 - 23$ to $n_1 + 6$. Of course, the forecast computations can fail if the model has regressor variables that have missing values in the forecast span. If the regressors contain missing values in the forecast horizon—that is, between the observations $n_1 - 23$ and $n_1 + 6$ —the forecast horizon is reduced accordingly.

You can specify the following *options*:

ALPHA=*value*

specifies the significance level of the forecast confidence intervals; for example, ALPHA=0.05, which is the default, results in a 95% confidence interval.

BACK=*integer*

SKIPLAST=*integer*

specifies the holdout sample for the evaluation of the forecasting performance of the model. For example, BACK=10 results in treating the last 10 observed values of the response series as unobserved. A post-sample-prediction-analysis table is produced for comparing the predicted values with the actual values in the holdout period. By default, BACK=0.

EXTRADIFFUSE=*k*

enables continuation of the diffuse filtering iterations for *k* additional iterations beyond the first instance where the initialization of the diffuse state would have otherwise taken place. If the specified *k* is larger than the sample size, the diffuse iterations continue until the end of the sample. Note that one-step-ahead forecasts are produced only after the diffuse state is initialized. Delaying the initialization leads to reduction in the number of one-step-ahead forecasts. This option is useful when you want to ignore the first few one-step-ahead forecasts that often have large variance.

LEAD=*integer*

specifies the number of periods to forecast beyond the historical period defined by the SKIPFIRST= and BACK= options; for example, LEAD=10 results in the forecasting of 10 future values of the response series. By default, LEAD=12.

OUTFOR=*libref.data-table*

specifies an output SAS table for the forecasts. The output SAS table contains the ID variable, the response and predictor series, the one-step-ahead and out-of-sample response series forecasts, the forecast confidence intervals, the smoothed values of the response series, and the smoothed forecasts produced as a result of the model-based decomposition of the series. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 2114.

PLOT=*plot-request*

PLOT=(*< plot-request > ... < plot-request >*)

produces forecast and model decomposition plots. You can specify the following *plot-requests*:

DECOMP

produces smoothed decomposition plots.

DECOMPVAR

produces plots of the variance of smoothed decomposition.

FDECOMP

produces filtered decomposition plots.

FDECOMPVAR

produces plots of the variance of filtered decomposition.

FORECASTS

produces a plot of series forecasts.

TREND

produces a plot of smoothed trend.

By default, a plot of multistep-ahead series forecasts in the post-sample region is produced.

PRINT=*print-request*

PRINT=(*< print-request > ... < print-request >*)

controls the printing of the series forecasts and the printing of smoothed model decomposition estimates. By default, the series forecasts are printed only for the forecast horizon that you specify in the **LEAD**= option; that is, the one-step-ahead predicted values are not printed. You can specify the following *print-requests*:

DECOMP

prints smoothed estimates of the trend, the trend plus regression, the trend plus regression plus cycle, and the sum of all components except the irregular.

FDECOMP

prints filtered estimates of the trend, the trend plus regression, the trend plus regression plus cycle, and the sum of all components except the irregular.

FORECASTS

prints forecasts for the entire forecast span.

NONE

suppress the printing of all the forecast output.

By default, a table of multistep-ahead series forecasts in the post-sample region is produced.

SKIPFIRST=*integer*

specifies that an early portion of the data needs to be ignored during the forecasting calculations. This can be useful if there is a reason to believe that the model that you are using for forecasting is not appropriate for this portion of the data. If you specify an *integer* value of 10, the first 10 measurements of the response series are skipped during the forecasting calculations. By default, SKIPFIRST=0.

ID Statement

ID *variable* INTERVAL=*value* < ALIGN=*value* > ;

The ID statement names a numeric variable that identifies observations in the input and output data sets. This statement is required. The values of the ID *variable* are assumed to be SAS date, time, or datetime values, or the ID *variable* values can signify the observation numbers within a sequence, such as 5 to indicate the fifth observation (for an example of an observation index as an ID variable, see [Example 30.2](#)). In addition, the ID statement specifies the frequency that is associated with the time series. The options also specify how to align the observations to form the time series. The INTERVAL= option is required. The values of the ID *variable* are extrapolated for the forecast observations on the basis of the values that you specify in the INTERVAL= option.

You must specify the following option:

INTERVAL=*value*

specifies the time interval between observations. This option is used in conjunction with the ID *variable* to verify that the input data are in the proper order and have no gaps. The option is also used to extrapolate the ID values past the end of the input data. For an ID variable that signifies the observation number within a sequence, specify INTERVAL=OBS. For more information about the supported intervals, see Chapter 4, “Date Intervals, Formats, and Functions” (*SAS/ETS User’s Guide*).

You can also specify the following option:

ALIGN=*value*

controls the alignment of SAS dates that are used to identify output observations. You can specify the following values: BEGINNING | BEG | B, MIDDLE | MID | M, and ENDING | END | E. The default is BEGINNING. The ALIGN= option is used to align the ID variable with the beginning, middle, or end of the time ID interval that you specify in the INTERVAL= option.

IRREGULAR Statement

IRREGULAR < *options* > ;

The IRREGULAR statement includes an irregular component in the model. There can be at most one IRREGULAR statement in the model specification. The irregular component corresponds to the overall random error ϵ_t in the model. By default, the irregular component is modeled as white noise—that is, as a sequence of independent, identically distributed, zero-mean, Gaussian random variables. However, you can also model it as an autoregressive moving average (ARMA) process. The options for specifying an ARMA model for the irregular component are given in the section “[ARMA Specification](#)” on page 2141.

The options in this statement enable you to specify the model for the irregular component and to output its estimates. Two examples of the IRREGULAR statement follow. In the first example, the statement is in its simplest form, resulting in the inclusion of an irregular component that is white noise with unknown variance:

```
irregular;
```

In the second example, the following statement provides a starting value for the white-noise variance σ_ϵ^2 to be used in the nonlinear parameter estimation process. It also prints smoothed estimates of ϵ_t . The smoothed irregulars are useful in model diagnostics.

```
irregular variance=4 print=smooth;
```

You can specify the following *options*:

NOEST

sets the value of σ_ϵ^2 to the value that you specify in the **VARIANCE=** option. Also see the **NOEST=** option in the section “[ARMA Specification](#)” on page 2141.

PLOT=FILTER

PLOT=SMOOTH

PLOT=(< FILTER > < SMOOTH >)

plots the filtered or smoothed estimate of the irregular component.

PRINT=FILTER

PRINT=SMOOTH

PRINT=(< FILTER > < SMOOTH >)

prints the filtered or smoothed estimate of the irregular component.

VARIANCE=value

specifies an initial value for σ_ϵ^2 during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

ARMA Specification

This section details the options for specifying an ARMA model for the irregular component. The specification of ARMA models requires some notation, which is explained first.

Let B denote the backshift operator—that is, for any sequence ϵ_t , $B\epsilon_t = \epsilon_{t-1}$. The higher powers of B represent larger shifts (for example, $B^3\epsilon_t = \epsilon_{t-3}$). A random sequence ϵ_t follows a zero-mean $\text{ARMA}(p,q) \times (P,Q)_s$ model with nonseasonal autoregressive order p , seasonal autoregressive order P , nonseasonal moving average order q , and seasonal moving average order Q , if it satisfies the following difference equation specified in terms of the polynomials in the backshift operator, where a_t is a white-noise sequence and s is the season length:

$$\phi(B)\Phi(B^s)\epsilon_t = \theta(B)\Theta(B^s)a_t$$

The polynomials ϕ , Φ , θ , and Θ are of orders p , P , q , and Q , respectively, which can be any nonnegative integers. The season length s must be a positive integer. For example, ϵ_t satisfies an $\text{ARMA}(1,1)$ model (that

is, $p = 1, q = 1, P = 0$, and $Q = 0$) if

$$\epsilon_t = \phi_1 \epsilon_{t-1} + a_t - \theta_1 a_{t-1}$$

for some coefficients ϕ_1 and θ_1 and a white-noise sequence a_t . Similarly, ϵ_t satisfies an $\text{ARMA}(1,1) \times (1,1)_{12}$ model if

$$\epsilon_t = \phi_1 \epsilon_{t-1} + \Phi_1 \epsilon_{t-12} - \phi_1 \Phi_1 \epsilon_{t-13} + a_t - \theta_1 a_{t-1} - \Theta_1 a_{t-12} + \theta_1 \Theta_1 a_{t-13}$$

for some coefficients ϕ_1, Φ_1, θ_1 , and Θ_1 and a white-noise sequence a_t . The ARMA process is stationary and invertible if the defining polynomials ϕ, Φ, θ , and Θ have all their roots outside the unit circle—that is, their absolute values are strictly larger than 1.0. It is assumed that the ARMA model specified for the irregular component is stationary and invertible—that is, the coefficients of the polynomials ϕ, Φ, θ , and Θ are constrained so that the stationarity and invertibility conditions are satisfied. The unknown coefficients of these polynomials become part of the model parameter vector that is estimated using the data.

The notation for a closely related class of models, autoregressive integrated moving average (ARIMA) models, is also given here. A random sequence y_t is said to follow an $\text{ARIMA}(p,d,q) \times (P,D,Q)_s$ model if, for some nonnegative integers d and D , the differenced series $\epsilon_t = (1 - B)^d (1 - B^s)^D y_t$ follows an $\text{ARMA}(p,q) \times (P,Q)_s$ model. The integers d and D are called nonseasonal and seasonal differencing orders, respectively. You can specify ARIMA models by using the **DEPLAG** statement for specifying the differencing orders and by using the **IRREGULAR** statement for the ARMA specification. For an example of $\text{ARIMA}(0,1,1) \times (0,1,1)_{12}$ model specification, see [Example 30.8](#). Brockwell and Davis (1991) can be consulted for additional information about ARIMA models.

You can use options of the **IRREGULAR** statement to specify the desired ARMA model and to request printed and graphical output. A few examples of the **IRREGULAR** statement are given next.

The following statement specifies an irregular component that is modeled as an $\text{ARMA}(1,1)$ process. It also requests plotting its smoothed estimate.

```
irregular p=1 q=1 plot=smooth;
```

The following statement specifies an $\text{ARMA}(1,1) \times (1,1)_{12}$ model. It also fixes the coefficient of the first-order seasonal moving average polynomial to 0.1. The other coefficients and the white-noise variance are estimated using the data.

```
irregular p=1 sp=1 q=1 sq=1 s=12 sma=0.1 noest=(sma);
```

You can specify the following ARMA model options:

AR= $\phi_1 \phi_2 \dots \phi_p$

lists the starting values of the coefficients of the nonseasonal autoregressive polynomial

$$\phi(B) = 1 - \phi_1 B - \dots - \phi_p B^p$$

where the order p is specified in the **P=** option. The coefficients ϕ_i must define a stationary autoregressive polynomial.

MA= $\theta_1 \theta_2 \dots \theta_q$

lists the starting values of the coefficients of the nonseasonal moving average polynomial

$$\theta(B) = 1 - \theta_1 B - \dots - \theta_q B^q$$

where the order q is specified in the **Q=** option. The coefficients θ_i must define an invertible moving average polynomial.

NOEST=(**<VARIANCE>** **<AR>** **<SAR>** **<MA>** **<SMA>**)

fixes the values of the ARMA parameters and the value of the white-noise variance to those specified in the **AR=**, **SAR=**, **MA=**, **SMA=**, or **VARIANCE=** option.

P=integer

specifies the order of the nonseasonal autoregressive polynomial. The order can be any nonnegative integer; by default, $P=0$. In practice, the order is a small integer such as 1, 2, or 3.

Q=integer

specifies the order of the nonseasonal moving average polynomial. The order can be any nonnegative integer; by default, $Q=0$. In practice, the order is a small integer such as 1, 2, or 3.

S=integer

specifies the season length used during the specification of the seasonal autoregressive or seasonal moving average polynomial. The season length can be any positive integer; for example, $S=4$ might be an appropriate value for a quarterly series. By default, $S=1$.

SAR= $\Phi_1 \Phi_2 \dots \Phi_P$

lists the starting values of the coefficients of the seasonal autoregressive polynomial

$$\Phi(B^S) = 1 - \Phi_1 B^S - \dots - \Phi_P B^{SP}$$

where the order P is specified in the **SP=** option and the season length s is specified in the **S=** option. The coefficients Φ_i must define a stationary autoregressive polynomial.

SMA= $\Theta_1 \Theta_2 \dots \Theta_Q$

lists the starting values of the coefficients of the seasonal moving average polynomial

$$\Theta(B^S) = 1 - \Theta_1 B^S - \dots - \Theta_Q B^{SQ}$$

where the order Q is specified in the **SQ=** option and the season length s is specified in the **S=** option. The coefficients Θ_i must define an invertible moving average polynomial.

SP=integer

specifies the order of the seasonal autoregressive polynomial. The order can be any nonnegative integer; by default, $SP=0$. In practice, the order is a small integer such as 1 or 2.

SQ=integer

specifies the order of the seasonal moving average polynomial. The order can be any nonnegative integer; by default, $SQ=0$. In practice, the order is a small integer such as 1 or 2.

LEVEL Statement

LEVEL <options> ;

The LEVEL statement is used to include a level component in the model. The level component, either by itself or together with a slope component (see the **SLOPE** statement), forms the trend component, μ_t , of the model. If the slope component is absent, the resulting trend is a random walk (RW) specified by the following equations:

$$\mu_t = \mu_{t-1} + \eta_t, \quad \eta_t \sim \text{iid } N(0, \sigma_\eta^2)$$

If the slope component is present, signified by the presence of a **SLOPE** statement, a locally linear trend (LLT) is obtained. The equations of LLT are as follows:

$$\begin{aligned} \mu_t &= \mu_{t-1} + \beta_{t-1} + \eta_t, & \eta_t &\sim \text{iid } N(0, \sigma_\eta^2) \\ \beta_t &= \beta_{t-1} + \xi_t, & \xi_t &\sim \text{iid } N(0, \sigma_\xi^2) \end{aligned}$$

In either case, the options in the LEVEL statement are used to specify the value of σ_η^2 and to request forecasts of μ_t . The SLOPE statement is used for similar purposes in the case of slope β_t . The following examples illustrate the use of the LEVEL statement. Assuming that a SLOPE statement is not added subsequently, a simple random walk trend is specified by the following statement:

```
level;
```

The following statements specify a locally linear trend with value of σ_η^2 fixed at 4. It also requests printing of filtered values of μ_t . The value of σ_ξ^2 , the disturbance variance in the slope equation, is estimated from the data.

```
level variance=4 noest print=filter;
slope;
```

You can specify the following *options*:

CHECKBREAK

turns on the checking of breaks in the level component.

NOEST

fixes the value of σ_η^2 to the value specified in the **VARIANCE=** option.

PLOT=FILTER

PLOT=SMOOTH

PLOT=(<FILTER> <SMOOTH>)

plots the filtered or smoothed estimate of the level component.

PRINT=FILTER**PRINT=SMOOTH****PRINT=(< FILTER > < SMOOTH >)**

prints the filtered or smoothed estimate of the level component.

VARIANCE=*value*specifies an initial value for σ_{η}^2 , the disturbance variance in the μ_t equation at the start of the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

MODEL Statement

MODEL *dependent variable* <= *regressors* > ;

The MODEL statement specifies the response variable and, optionally, the predictor or regressor variables for the UCM model. This is a required statement in the UCM procedure. The predictors specified in the MODEL statement are assumed to have a linear and time-invariant relationship with the response. The predictors that have time-varying regression coefficients are specified separately in the [RANDOMREG](#) statement. Similarly, the predictors that have a nonlinear effect on the response variable are specified separately in the [SPLINEREG](#) statement. Only one MODEL statement can be specified.

NLOPTIONS Statement

NLOPTIONS < *options* > ;

PROC UCM uses the nonlinear optimization (NLO) subsystem to perform the nonlinear optimization of the likelihood function during the estimation of model parameters. You can use the NLOPTIONS statement to control different aspects of this optimization process. For most problems, the default settings of the optimization process are adequate. However, in some cases it might be useful to change the optimization technique or to change the maximum number of iterations. You can do this as follows by using the TECH= and MAXITER= options in the NLOPTIONS statement:

```
nloptions tech=dbldog maxiter=200;
```

This statement sets the maximum number of iterations to 200 and changes the optimization technique to DBLDOG rather than the default technique, TRUREG, that PROC UCM uses. For more information about the options that you can specify in the NLOPTIONS statement, see Chapter 6, “Nonlinear Optimization Methods” (*SAS/ETS User’s Guide*). In this version of PROC UCM, the user can specify only the TECH= and MAXITER= options; all other options remain at their default settings.

OUTLIER Statement

OUTLIER < options > ;

The OUTLIER statement enables you to control the reporting of the additive outliers (AO) and level shifts (LS) in the response series. The AOs are searched by default. You can search for LSs by using the [CHECKBREAK](#) option in the LEVEL statement.

You can specify the following *options*:

ALPHA=*significance-level*

specifies the significance level for reporting the outliers. By default, ALPHA=0.05.

MAXNUM=*number*

specifies the number of outliers to search for. By default, MAXNUM=5.

MAXPCT=*number*

is similar to the MAXNUM= option. The MAXPCT= option enables you to limit the number of outliers to search for according to a percentage of the series length. By default, MAXPCT=1.

NOTE: The number of outliers that are searched for is always the lower of the two numbers implied by the MAXNUM= and MAXPCT= options, regardless of whether they are explicitly specified or set to their defaults.

PRINT=SHORT | DETAIL

controls the printed output of the outlier search. The PRINT=SHORT option, which is the default, produces an outlier summary table that contains the most significant outliers, either AO or LS, that the outlier search finds. The PRINT=DETAIL option produces, in addition to the outlier summary table, separate tables that contain the AO and LS structural break chi-square statistics that are computed at each time point in the estimation span.

RANDOMREG Statement

RANDOMREG *regressors* < / options > ;

The RANDOMREG statement is used to specify regressors with time-varying regression coefficients. Each regression coefficient—for example, β_t —is assumed to evolve as a random walk:

$$\beta_t = \beta_{t-1} + \eta_t, \quad \eta_t \sim \text{iid } N(0, \sigma^2)$$

Of course, if the random walk disturbance variance σ^2 is 0, then the regression coefficient is not time-varying, and it reduces to the standard regression setting. There can be multiple RANDOMREG statements, and each statement can contain one or more regressors. The regressors in a given RANDOMREG statement form a group that is assumed to share the same disturbance variance parameter. The random walks associated with different regressors are assumed to be independent. For an example of using this statement, see [Example 30.4](#). For additional information about how estimates are reported for this type of regressors, see the section “Reporting Parameter Estimates for Random Regressors” on page 2170.

You can specify the following *options*:

NOEST

fixes the value of σ^2 to the value specified in the **VARIANCE=** option.

PLOT=FILTER**PLOT=SMOOTH****PLOT=(< FILTER > < SMOOTH >)**

plots the filtered or smoothed estimate of the time-varying regression coefficient.

PRINT=FILTER**PRINT=SMOOTH****PRINT=(< FILTER > < SMOOTH >)**

prints the filtered or smoothed estimate of the time-varying regression coefficient.

VARIANCE=*value*

specifies an initial value for σ^2 during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

SEASON Statement

SEASON LENGTH=*integer* < *options* > ;

The SEASON or SEASONAL statement is used to specify a seasonal component, γ_t , in the model. A seasonal component can be one of the two types, DUMMY or TRIG. A DUMMY seasonal with season length s satisfies the following stochastic equation:

$$\sum_{i=0}^{s-1} \gamma_{t-i} = \omega_t, \quad \omega_t \sim \text{iid } N(0, \sigma_\omega^2)$$

The equations for a TRIG (short for trigonometric) seasonal component are as follows

$$\gamma_t = \sum_{j=1}^{[s/2]} \gamma_{j,t}$$

where $[s/2]$ equals $s/2$ if s is even and $(s-1)/2$ if it is odd. The sinusoids, also called harmonics, $\gamma_{j,t}$ have frequencies $\lambda_j = 2\pi j/s$ and are specified by the matrix equation

$$\begin{bmatrix} \gamma_{j,t} \\ \gamma_{j,t}^* \end{bmatrix} = \begin{bmatrix} \cos \lambda_j & \sin \lambda_j \\ -\sin \lambda_j & \cos \lambda_j \end{bmatrix} \begin{bmatrix} \gamma_{j,t-1} \\ \gamma_{j,t-1}^* \end{bmatrix} + \begin{bmatrix} \omega_{j,t} \\ \omega_{j,t}^* \end{bmatrix}$$

where the disturbances $\omega_{j,t}$ and $\omega_{j,t}^*$ are assumed to be independent and, for fixed j , $\omega_{j,t}$ and $\omega_{j,t}^* \sim N(0, \sigma_\omega^2)$. If s is even, then the equation for $\gamma_{s/2,t}^*$ is not needed and $\gamma_{s/2,t}$ is given by

$$\gamma_{s/2,t} = -\gamma_{s/2,t-1} + \omega_{s/2,t}$$

In the TRIG seasonal case, the option **KEEPPH=** or **DROPH=** can be used to obtain *subset trigonometric* seasonals that contain only a subset of the full set of harmonics $\gamma_{j,t}$, $j = 1, 2, \dots, [s/2]$. This is particularly useful when the season length s is large and the seasonal pattern is relatively smooth.

Note that whether the seasonal type is DUMMY or TRIG, there is only one parameter, the disturbance variance σ_ω^2 , in the seasonal model.

There can be more than one seasonal component in the model, necessarily with different season lengths if the seasons are full. You can have multiple *subset* season components with the same season length, if you need to use separate disturbance variances for different sets of harmonics. Each seasonal component is specified using a separate SEASON statement. A model with multiple seasonal components can easily become quite complex and might need a large amount of data and computing resources for its estimation and forecasting. The examples that follow illustrate the use of SEASON statement.

The following statement specifies a DUMMY type (default) seasonal component with a season length of four, corresponding to the quarterly seasonality. The disturbance variance σ_ω^2 is estimated from the data.

```
season length=4;
```

The following statement specifies a trigonometric seasonal with monthly seasonality. It also provides a starting value for σ_ω^2 .

```
season length=12 type=trig variance=4;
```

You must specify the following *option*:

LENGTH=*integer*

specifies the season length, s . The season length can be any integer greater than or equal to 2. Typical examples of season lengths are 12, corresponding to the monthly seasonality, or 4, corresponding to the quarterly seasonality.

You can also specify the following *options*:

DROPHARMONICS | **DROPH=***number-list* | **n TO m BY p**

enables you to drop some harmonics $\gamma_{j,t}$ from the full set of harmonics used to obtain a trigonometric seasonal. The drop list can include any integer between 1 and $[s/2]$, where s is the season length. For example, the following specification results in a specification of a trigonometric seasonal with a season length 12 that consists of only the first four harmonics $\gamma_{j,t}$, $j = 1, 2, 3, 4$:

```
season length=12 type=trig DROPH=5 6;
```

The last two *high*-frequency harmonics are dropped. The **DROPH=** option cannot be used with the **KEEPH=** option.

KEEPHARMONICS | **KEEPH=***number-list* | **n TO m BY p**

enables you to keep only the harmonics $\gamma_{j,t}$ listed in the option to obtain a trigonometric seasonal. The keep list can include any integer between 1 and $[s/2]$, where s is the season length. For example, the following specification results in a specification of a trigonometric seasonal with a season length of 12 that consists of all six harmonics $\gamma_{j,t}$, $j = 1, \dots, 6$:

```
season length=12 type=trig KEEPH=1 to 3;
season length=12 type=trig KEEPH=4 to 6;
```

However, these six harmonics are grouped into two groups, each having its own disturbance variance parameter. The **DROPH=** option cannot be used with the **KEEPH=** option.

NOEST

fixes the value of the disturbance variance parameter to the value specified in the **VARIANCE=** option.

PLOT=FILTER

PLOT=SMOOTH

PLOT=F_ ANNUAL

PLOT=S_ ANNUAL

PLOT=(<plot-request> ... <plot-request>)

plots the season component. When you specify only one *plot-request*, you can omit the parentheses around it. You can use the **FILTER** and **SMOOTH** options to plot the filtered and smoothed estimates of the season component γ_t . You can use the **F_ ANNUAL** and **S_ ANNUAL** options to produce the plots of “annual” variation in the filtered and smoothed estimates of γ_t . The annual plots are useful to see the change in the contribution of a particular month over the span of years. Here “month” and “year” are generic terms that change appropriately with the interval type being used to label the observations and the season length. For example, for monthly data with a season length of 12, the usual meaning applies, while for daily data with a season length of 7, the days of the week serve as months and the weeks serve as years.

PRINT=HARMONICS

prints the summary of harmonics present in the seasonal component. This option is valid only for the trigonometric seasonal component.

PRINT=FILTER

PRINT=SMOOTH

PRINT=(<print-request> ... <print-request>)

prints the filtered or smoothed estimate of the seasonal component γ_t .

TYPE=DUMMY | TRIG

specifies the type of the seasonal component. By default, **TYPE=DUMMY**.

VARIANCE=value

specifies an initial value for the disturbance variance, σ_ω^2 , in the γ_t equation at the start of the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

SLOPE Statement

SLOPE <options> ;

The SLOPE statement is used to include a slope component in the model. The slope component cannot be used without the level component (see the **LEVEL** statement). The level and slope specifications jointly define the trend component of the model. A SLOPE statement without the accompanying LEVEL statement is ignored. The equations of the trend, defined jointly by the level μ_t and slope β_t , are as follows:

$$\begin{aligned}\mu_t &= \mu_{t-1} + \beta_{t-1} + \eta_t, & \eta_t &\sim \text{iid } N(0, \sigma_\eta^2) \\ \beta_t &= \beta_{t-1} + \xi_t, & \xi_t &\sim \text{iid } N(0, \sigma_\xi^2)\end{aligned}$$

The SLOPE statement is used to specify the value of the disturbance variance, σ_ξ^2 , in the slope equation, and to request forecasts of β_t . The following examples illustrate this statement:

```
level;
slope;
```

The preceding statements fit a model with a locally linear trend. The disturbance variances σ_η^2 and σ_ξ^2 are estimated from the data. You can fit a locally linear trend with fixed slope by using the following statements:

```
level;
slope variance=0 noest;
```

You can specify the following *options*:

NOEST

fixes the value of the disturbance variance, σ_ξ^2 , to the value specified in the **VARIANCE=** option.

PLOT=FILTER

PLOT=SMOOTH

PLOT=(<FILTER> <SMOOTH>)

plots the filtered or smoothed estimate of the slope component.

PRINT=FILTER

PRINT=SMOOTH

PRINT=(<FILTER> <SMOOTH>)

prints the filtered or smoothed estimate of the slope component β_t .

VARIANCE=value

specifies an initial value for the disturbance variance, σ_ξ^2 , in the β_t equation at the start of the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

SPLINEREG Statement

SPLINEREG *regressor* < *options* > ;

The SPLINEREG statement is used to specify a regressor that has a nonlinear relationship with the dependent series that can be approximated by a given B-spline. If the specified spline has degree d and is based on n internal knots, then it is known that it can be written as a linear combination of $(n + d + 1)$ regressors that are derived from the original regressor. The span of these $(n + d + 1)$ -derived regressors includes a constant; therefore, to avoid multicollinearity with the level component, one of these regressors is dropped. Specifying the SPLINEREG statement is equivalent to specifying a RANDOMREG statement with these derived regressors. You can specify multiple SPLINEREG statements. You must specify at least one interior knot, using either the NKNOTS= option or the KNOTS= option. For more information about splines, see Chapter 126, “The TRANSREG Procedure” (*SAS/STAT User’s Guide*). For an example of using this statement, see [Example 30.6](#). For additional information about how parameter estimates are reported for this type of regressors, see the section “[Reporting Parameter Estimates for Random Regressors](#)” on page 2170.

You can specify the following *options*:

DEGREE=*integer*

specifies the degree of the spline. It can be any integer larger than or equal to 0. By default, DEGREE=3. The polynomial degree should be a small integer, usually 0, 1, 2, or 3. Larger values are rarely useful. If you have any doubt as to what degree to specify, use the default.

KNOTS=*number-list* | *n* **TO** *m* **BY** *p*

specifies the interior knots or break points. The values in the knot list must be nondecreasing and must lie between the minimum and the maximum of the spline regressor values in the input data set. The first time you specify a value in the knot list, it indicates a discontinuity in the n th (from DEGREE= n) derivative of the transformation function at the value of the knot. The second mention of a value indicates a discontinuity in the $(n - 1)$ th derivative of the transformation function at the value of the knot. Knots can be repeated any number of times for decreasing smoothness at the break points, but the values in the knot list can never decrease.

You cannot use the [KNOTS=](#) option with the [NKNOTS=](#) option. You should keep the number of knots small.

NKNOTS=*m*

creates m knots, the first at the $100/(m + 1)$ percentile, the second at the $200/(m + 1)$ percentile, and so on. Knots are always placed at data values; there is no interpolation. For example, if NKNOTS=3, knots are placed at the 25th percentile, the median, and the 75th percentile. The value specified for the NKNOTS= option must be ≥ 1 . You cannot use the [NKNOTS=](#) option with the [KNOTS=](#) option.

NOTE: Specifying knots by using the NKNOTS= option can result in different sets of knots in the estimation and forecast stages if the distributions of regressor values in the estimation and forecast spans differ. The estimation span is based on the BACK= and SKIPFIRST= options in the [ESTIMATE](#) statement, and the forecast span is based on the BACK= and SKIPFIRST= options in the [FORECAST](#) statement.

NOEST

fixes the value of the regression coefficient random walk disturbance variance to the value specified in the **VARIANCE=** option.

PLOT=FILTER**PLOT=SMOOTH****PLOT=(< FILTER > < SMOOTH >)**

plots the filtered or smoothed estimate of the time-varying regression coefficient.

PRINT=FILTER**PRINT=SMOOTH****PRINT=(< FILTER > < SMOOTH >)**

prints the filtered or smoothed estimate of the time-varying regression coefficient.

VARIANCE=*value*

specifies an initial value for the regression coefficient random walk disturbance variance during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

SPLINESEASON Statement

SPLINESEASON LENGTH=*integer* **KNOTS=***integer*₁ *integer*₂ ... **< options >** ;

The SPLINESEASON statement is used to specify a seasonal pattern that is to be approximated by a given B-spline. If the specified spline has degree d and is based on n internal knots, then it can be written as a linear combination of $(n + d)$ regressors that are derived from the seasonal dummy regressors. The SPLINESEASON specification is equivalent to specifying a RANDOMREG specification with these derived regressors. Such approximation is useful only if the season length is relatively large, at least larger than $(n + d)$. For additional information about splines, see Chapter 126, “The TRANSREG Procedure” (*SAS/STAT User's Guide*). For an example of using this statement, see [Example 30.3](#).

You must specify the following *options*:

LENGTH=*integer*

specifies the season length, s . The length can be any integer greater than or equal to 3.

KNOTS=*integer*₁ *integer*₂ ...

lists the *internal* knots. This list of values must be a nondecreasing sequence of integers within the range of 2 to $(s - 1)$, where s is the season length specified in the **LENGTH=** option.

You can also specify the following *options*:

DEGREE=*integer*

specifies the degree of the spline. It can be any integer greater than or equal to 0. By default, **DEGREE=3**.

NOEST

fixes the value of the regression coefficient random walk disturbance variance to the value specified in the **VARIANCE=** option.

OFFSET=integer

specifies the position of the first measurement within the season, if the first measurement is not at the start of the season. The **OFFSET=** value must be between one and the season length. The default value is one. The first measurement refers to the start of the estimation span and the forecast span. If these spans differ, their starting measurements must be separated by an integer multiple of the season length.

PLOT=FILTER**PLOT=SMOOTH****PLOT=(< FILTER > < SMOOTH >)**

plots the season component. When you specify only one plot request, you can omit the parentheses around the plot request. You can use the **FILTER** and **SMOOTH** options to plot the filtered and smoothed estimates of the season component.

PRINT=FILTER**PRINT=SMOOTH****PRINT=(< FILTER > < SMOOTH >)**

prints the filtered or smoothed estimate of the spline season component.

RKNOTS=(knot, ..., knot) ... (knot, ..., knot)

specifies a grouping of knots such that the knots within the same group have identical seasonal values. The knots specified in this option must already be present in the list specified by the **KNOTS=** option. The knot groups must be non-overlapping and without any repeated knots.

VARIANCE=value

specifies an initial value for the regression coefficient random walk disturbance variance during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value.

TF Statement

TF regressor < options > ;

The TF statement specifies a regressor that has a transfer-function relationship with the dependent series. A transfer function is useful for capturing the contributions from lagged values of the regressor. Box and Jenkins (1976) popularized ARIMA models that have transfer-function inputs. In the UCM procedure, you can specify a transfer function of the following type (assuming the regression variable is x):

$$\frac{(\gamma_0 + \gamma_1 B^{l1} + \gamma_2 B^{l2} + \dots) B^d}{(1 - \delta_1 B - \delta_2 B^2 - \dots - \delta_m B^m)(1 - \omega_1 B^s - \omega_2 B^{2s} - \dots - \omega_n B^{ns})} x_t$$

This transfer function is specified by using the ratio of polynomials in the backshift operator B . The numerator polynomial orders ($l1, l2, \dots$) are positive integers, possibly with gaps (for example, 1, 3). The numerator term B^d signifies the *delay* of order d . The denominator polynomial can have two factors: a nonseasonal factor, $(1 - \delta_1 B - \delta_2 B^2 - \dots - \delta_m B^m)$, and a seasonal factor whose season length is s , $(1 - \omega_1 B^s - \omega_2 B^{2s} - \dots - \omega_n B^{ns})$. The orders of the terms in the denominator factors cannot have gaps;

that is, if 5 is the maximum order of the nonseasonal factor, then all terms of orders 1 through 5 are present. By design, the denominator factors are restricted to be stable polynomials (their roots are strictly larger than 1 in absolute value). As an example, consider the following transfer function specification:

$$\frac{(\gamma_0 + \gamma_1 B^1 + \gamma_2 B^2) B^3}{(1 - \delta_1 B - \delta_2 B^2)(1 - \omega_1 B^4)} x_t$$

You can specify this transfer function as follows:

```
tf x num=(1 2) den=2 sden=1 s=4 delay=3;
```

Since the numerator polynomial orders do not have any gaps, the following simpler specification is also available:

```
tf x num=2 den=2 sden=1 s=4 delay=3;
```

Because the denominator factors do not permit gaps in their orders, only the maximum orders need to be provided in their specification.

A state space representation of a transfer-function relationship is described in the section “[State Space Form of a Transfer Function Relationship](#)” on page 2170. You can specify multiple TF statements, each one with a separate regressor. A regressor that is specified in any transfer function specification must not appear in any other regression specifications, such as in the right-hand side of the MODEL statement or in the RANDOMREG and SPLINEREG statements.

NOTE: The mathematical form of the transfer function considered by PROC UCM is similar to the one considered in the ARIMA procedure (Chapter 7, “ARIMA Procedure” ([SAS/ETS User’s Guide](#))). However, there are some differences:

- The sign convention of the coefficients of the nonzero-order terms in the numerator polynomial in the UCM procedure is opposite to that of the ARIMA procedure.
- The ARIMA procedure permits multiple polynomial factors in both the numerator and the denominator. The UCM procedure permits only one numerator factor and at most two denominator factors.
- The ARIMA procedure permits full control over the terms present in each of the polynomial factors. The UCM procedure does not permit such fine control over the terms in the polynomials.
- In the UCM procedure, you cannot fix the coefficients of the numerator polynomial. They are always estimated from the data.
- In the UCM procedure, if both nonseasonal and seasonal factors are present in the denominator, you must specify starting values for their coefficients either for both factors or for neither.

You can specify the following *options* in the TF statement:

DELAY=integer

specifies the delay order, which must be a positive integer. By default, DELAY=0.

DEN=integer

specifies the maximum order of the nonseasonal factor of the denominator polynomial. By default, DEN=0.

DENVAL=val1 val2 ...

specifies the starting values of the coefficients of the nonseasonal factor of the denominator polynomial. The number of values supplied in the DENVAL= option must match the value of the DEN= option. Moreover, the resulting polynomial must be stable.

NOEST

fixes the values of the denominator polynomial coefficients to those specified in the DENVAL= and SDENVAL= specifications.

NUM=argument

specifies the positive orders of the terms in the numerator polynomial. You can specify the *argument* in either of the following forms:

integer includes all orders from 1 to *integer*.

(*lag1*, *lag2*, ...) specifies a more general list of orders.

PLOT=FILTER**PLOT=SMOOTH****PLOT=(< FILTER > < SMOOTH >)**

plots the transfer-function component. When you specify only one plot request, you can omit the parentheses around it. You can use the FILTER and SMOOTH options to plot the filtered and smoothed estimates of the transfer-function component.

PRINT=FILTER**PRINT=SMOOTH****PRINT=(< FILTER > < SMOOTH >)**

prints the filtered or smoothed estimate of the transfer-function component. When you specify only one print request, you can omit the parentheses around it. You can use the FILTER and SMOOTH options to print the filtered and smoothed estimates of the transfer-function component.

S=integer

specifies the season length that is used in the specification of the seasonal factor of the denominator polynomial. The season length can be any positive integer; for example, S=4 might be an appropriate value for a quarterly series. By default, S=1.

SDEN=integer

specifies the maximum order of the seasonal factor of the denominator polynomial. By default, SDEN=0.

SDENVAL=*val1 val2 ...*

specifies the starting values of the coefficients of the seasonal factor of the denominator polynomial. The number of values supplied in this option must match the value of the SDEN= option. Moreover, the resulting polynomial must be stable.

TFSTART=*value*

specifies the value of the transfer function at the start of the sample (the first time ID). By default, the value of this option is a missing value that is estimated from the data. This option is often used when the past values of the transfer function can be inferred because of the structure of the problem or when it is useful to set these values (usually to 0) to achieve identifiability of the overall model. For more information, see the section “[State Space Form of a Transfer Function Relationship](#)” on page 2170. See [Example 30.10](#) for an example of the use of this option.

Details: UCM Procedure

An Introduction to Unobserved Component Models

A UCM decomposes the response series into components such as trend, seasons, cycles, and the regression effects due to predictor series. The following model shows a possible scenario:

$$y_t = \mu_t + \gamma_t + \psi_t + \sum_{j=1}^m \beta_j x_{jt} + \epsilon_t$$

$$\epsilon_t \sim \text{iid } N(0, \sigma_\epsilon^2)$$

The terms μ_t , γ_t , and ψ_t represent the trend, seasonal, and cyclical components, respectively. In fact the model can contain multiple seasons and cycles, and the seasons can be of different types. For simplicity of discussion the preceding model contains only one of each of these components. The regression term, $\sum_{j=1}^m \beta_j x_{jt}$, includes contribution of regression variables with *fixed* regression coefficients. A model can also contain regression variables that have *time-varying* regression coefficients or that have a nonlinear or a transfer-function relationship with the dependent series (see “[Incorporating Predictors of Different Types](#)” on page 2169). The disturbance term ϵ_t , also called the *irregular* component, is usually assumed to be Gaussian white noise. In some cases it is useful to model the irregular component as a stationary ARMA process. For additional information, see the section “[Modeling the Irregular Component](#)” on page 2160.

By controlling the presence or absence of various terms and by choosing the proper flavor of the included terms, the UCMs can generate a rich variety of time series patterns. A UCM can be applied to variables after transforming them by transforms such as *log* and *difference*.

The components μ_t , γ_t , and ψ_t model structurally different aspects of the time series. For example, the trend μ_t models the natural tendency of the series in the absence of any other perturbing effects such as seasonality, cyclical components, and the effects of exogenous variables, while the seasonal component γ_t models the correction to the level due to the seasonal effects. These components are assumed to be statistically independent of each other and independent of the irregular component. All of the component models can be thought of as stochastic generalizations of the relevant deterministic patterns in time. This

way the deterministic cases emerge as special cases of the stochastic models. The different models available for these unobserved components are discussed next.

Modeling the Trend

As mentioned earlier, the trend in a series can be loosely defined as the natural tendency of the series in the absence of any other perturbing effects. The UCM procedure offers two ways to model the trend component μ_t . The first model, called the random walk (RW) model, implies that the trend remains roughly constant throughout the life of the series without any persistent upward or downward drift. In the second model the trend is modeled as a locally linear time trend (LLT). The RW model can be described as

$$\mu_t = \mu_{t-1} + \eta_t, \quad \eta_t \sim \text{iid } N(0, \sigma_\eta^2)$$

Note that if $\sigma_\eta^2 = 0$, then the model becomes $\mu_t = \text{constant}$. In the LLT model the trend is locally linear, consisting of both the *level* and *slope*. The LLT model is

$$\begin{aligned} \mu_t &= \mu_{t-1} + \beta_{t-1} + \eta_t, & \eta_t &\sim \text{iid } N(0, \sigma_\eta^2) \\ \beta_t &= \beta_{t-1} + \xi_t, & \xi_t &\sim \text{iid } N(0, \sigma_\xi^2) \end{aligned}$$

The disturbances η_t and ξ_t are assumed to be independent. There are some interesting special cases of this model obtained by setting one or both of the disturbance variances σ_η^2 and σ_ξ^2 equal to 0. If σ_ξ^2 is set equal to 0, then you get a linear trend model with fixed slope. If σ_η^2 is set to 0, then the resulting model usually has a smoother trend. If both the variances are set to 0, then the resulting model is the deterministic linear time trend: $\mu_t = \mu_0 + \beta_0 t$.

You can incorporate these trend patterns in your model by using the **LEVEL** and **SLOPE** statements.

Modeling a Cycle

A deterministic cycle ψ_t with frequency λ , $0 < \lambda < \pi$, can be written as

$$\psi_t = \alpha \cos(\lambda t) + \beta \sin(\lambda t)$$

If the argument t is measured on a continuous scale, then ψ_t is a periodic function with period $2\pi/\lambda$, amplitude $\gamma = (\alpha^2 + \beta^2)^{1/2}$, and phase $\phi = \tan^{-1}(\beta/\alpha)$. Equivalently, the cycle can be written in terms of the amplitude and phase as

$$\psi_t = \gamma \cos(\lambda t - \phi)$$

Note that when ψ_t is measured only at the integer values, it is not exactly periodic, unless $\lambda = (2\pi j)/k$ for some integers j and k . The cycles in their pure form are not used very often in practice. However, they are very useful as building blocks for more complex periodic patterns. It is well known that the periodic pattern of any complexity can be written as a sum of pure cycles of different frequencies and amplitudes. In time series situations it is useful to generalize this simple cyclical pattern to a stochastic cycle that has a fixed expected period but time-varying amplitude and phase. The stochastic cycle considered here is motivated by the following recursive formula for computing ψ_t ,

$$\begin{bmatrix} \psi_t \\ \psi_t^* \end{bmatrix} = \begin{bmatrix} \cos \lambda & \sin \lambda \\ -\sin \lambda & \cos \lambda \end{bmatrix} \begin{bmatrix} \psi_{t-1} \\ \psi_{t-1}^* \end{bmatrix}$$

starting with $\psi_0 = \alpha$ and $\psi_0^* = \beta$. Note that ψ_t and ψ_t^* satisfy the relation

$$\psi_t^2 + \psi_t^{*2} = \alpha^2 + \beta^2 \quad \text{for all } t$$

A stochastic generalization of the cycle ψ_t can be obtained by adding random noise to this recursion and by introducing a damping factor, ρ , for additional modeling flexibility. This model can be described as follows,

$$\begin{bmatrix} \psi_t \\ \psi_t^* \end{bmatrix} = \rho \begin{bmatrix} \cos \lambda & \sin \lambda \\ -\sin \lambda & \cos \lambda \end{bmatrix} \begin{bmatrix} \psi_{t-1} \\ \psi_{t-1}^* \end{bmatrix} + \begin{bmatrix} \nu_t \\ \nu_t^* \end{bmatrix}$$

where $0 \leq \rho \leq 1$, and the disturbances ν_t and ν_t^* are independent $N(0, \sigma_\nu^2)$ variables. The resulting stochastic cycle has a fixed expected period but time-varying amplitude and phase. The stationarity properties of the random sequence ψ_t depend on the damping factor ρ . If $\rho < 1$, ψ_t has a stationary distribution with mean 0 and variance $\sigma_\nu^2/(1 - \rho^2)$. If $\rho = 1$, ψ_t is nonstationary.

You can incorporate a cycle in a UCM by specifying a **CYCLE** statement. You can include multiple cycles in the model by using separate **CYCLE** statements for each included cycle.

As mentioned before, the cycles are very useful as building blocks for constructing more complex periodic patterns. Periodic patterns of almost any complexity can be created by superimposing cycles of different periods and amplitudes. In particular, the seasonal patterns, general periodic patterns with integer periods, can be constructed as sums of cycles. This important topic of modeling the seasonal components is considered next.

Modeling Seasons

Seasonal fluctuations are a common source of variation in time series data. These fluctuations arise because of the regular changes in seasons or some other periodic events. The seasonal effects are regarded as corrections to the general trend of the series due to the seasonal variations, and these effects sum to 0 when summed over the full season cycle. Therefore the seasonal component γ_t is modeled as a stochastic periodic pattern of an integer period s such that the sum $\sum_{i=0}^{s-1} \gamma_{t-i}$ is always 0 in the mean. The period s is called the season length. Two different models for the seasonal component are considered here. The first model is called the *dummy* variable form of the seasonal component. It is described by the equation

$$\sum_{i=0}^{s-1} \gamma_{t-i} = \omega_t, \quad \omega_t \sim \text{iid } N(0, \sigma_\omega^2)$$

The other model is called the *trigonometric* form of the seasonal component. In this case γ_t is modeled as a sum of cycles of different frequencies. This model is given by

$$\gamma_t = \sum_{j=1}^{[s/2]} \gamma_{j,t}$$

where $[s/2]$ equals $s/2$ if s is even and $(s-1)/2$ if it is odd. The cycles $\gamma_{j,t}$ have frequencies $\lambda_j = 2\pi j/s$ and are specified by the matrix equation

$$\begin{bmatrix} \gamma_{j,t} \\ \gamma_{j,t}^* \end{bmatrix} = \begin{bmatrix} \cos \lambda_j & \sin \lambda_j \\ -\sin \lambda_j & \cos \lambda_j \end{bmatrix} \begin{bmatrix} \gamma_{j,t-1} \\ \gamma_{j,t-1}^* \end{bmatrix} + \begin{bmatrix} \omega_{j,t} \\ \omega_{j,t}^* \end{bmatrix}$$

where the disturbances $\omega_{j,t}$ and $\omega_{j,t}^*$ are assumed to be independent and, for fixed j , $\omega_{j,t}$ and $\omega_{j,t}^* \sim N(0, \sigma_\omega^2)$. If s is even, then the equation for $\gamma_{s/2,t}^*$ is not needed and $\gamma_{s/2,t}$ is given by

$$\gamma_{s/2,t} = -\gamma_{s/2,t-1} + \omega_{s/2,t}$$

The cycles $\gamma_{j,t}$ are called *harmonics*. If the seasonal component is deterministic, the decomposition of the seasonal effects into these harmonics is identical to its Fourier decomposition. In this case the sum of squares of the seasonal factors equals the sum of squares of the amplitudes of these harmonics. In many practical situations, the contribution of the high-frequency harmonics is negligible and can be ignored, giving rise to a simpler description of the seasonal. In the case of stochastic seasonals, the situation might not be so transparent; however, similar considerations still apply. Note that if the disturbance variance $\sigma_\omega^2 = 0$, then both the dummy and the trigonometric forms of seasonal components reduce to constant seasonal effects. That is, the seasonal component reduces to a deterministic function that is completely determined by its first $s - 1$ values.

In the UCM procedure you can specify a seasonal component in a variety of ways, the **SEASON** statement being the simplest of these. The dummy and the trigonometric seasonal components discussed so far can be considered as *saturated* seasonal components that put no restrictions on the $s - 1$ seasonal values. In some cases a more parsimonious representation of the seasonal might be more appropriate. This is particularly useful for seasonal components with large season lengths. In the UCM procedure you can obtain parsimonious representations of the seasonal components by one of the following ways:

- Use a *subset* trigonometric seasonal component obtained by deleting a few of the $[s/2]$ harmonics used in its sum. For example, a slightly smoother seasonal component of length 12, corresponding to the monthly seasonality, can be obtained by deleting the highest-frequency harmonic of period 2. That is, such a seasonal component will be a sum of five stochastic cycles that have periods 12, 6, 4, 3, and 2.4. You can specify such subset seasonal components by using the **KEEPH=** or **DROPH=** option in the **SEASON** statement.
- Approximate the seasonal pattern by a suitable spline approximation. You can do this by using the **SPLINESEASON** statement.
- A *block-seasonal* pattern is a seasonal pattern where the pattern is divided into a few blocks of equal length such that the season values within a block are the same—for example, a monthly seasonal pattern that has only four different values, one for each quarter. In some situations a long seasonal pattern can be approximated by the sum of block season and a simple season, the length of the simple season being equal to the block length of the block season. You can obtain such approximation by using a combination of **BLOCKSEASON** and **SEASON** statements.
- Consider a seasonal component of a large season length as a sum of two or more seasonal components that are each of much smaller season lengths. This can be done by specifying more than one **SEASON** statements.

Note that the preceding techniques of obtaining parsimonious seasonal components can also enable you to specify seasonal components that are more *general* than the simple saturated seasonal components. For example, you can specify a saturated trigonometric seasonal component that has some of its harmonics evolving according to one disturbance variance parameter while the others evolve with another disturbance variance parameter.

Modeling an Autoregression

An autoregression of order one can be thought of as a special case of a cycle when the frequency λ is either 0 or π . Modeling this special case separately helps interpretation and parameter estimation. The autoregression component r_t is modeled as

$$r_t = \rho r_{t-1} + v_t, \quad v_t \sim \text{iid } N(0, \sigma_v^2)$$

where $-1 \leq \rho < 1$. An autoregression can also provide an alternative to the **IRREGULAR** component when the model errors show some autocorrelation. You can incorporate an autoregression in your model by using the **AUTOREG** statement.

Modeling Regression Effects

A predictor variable can affect the response variable in a variety of ways. The UCM procedure enables you to model several different types of predictor-response relationships:

- The predictor-response relationship is *linear*, and the regression coefficient does not change with time. This is the simplest kind of relationship and such predictors are specified in the **MODEL** statement.
- The predictor-response relationship is *linear*, but the regression coefficient does change with time. Such predictors are specified in the **RANDOMREG** statement. Here the regression coefficient is assumed to evolve as a random walk.
- The predictor-response relationship is *nonlinear* and the relationship can change with time. This type of relationship can be approximated by an appropriate time-varying spline. Such predictors are specified in the **SPLINEREG** statement.
- The response depends on contemporaneous and lagged values of the predictor. This type of relationship is called transfer-function relationship, which can be specified in the **TF** statement.

A response variable can depend on its own past values—that is, lagged dependent values. Such a relationship can be specified in the **DEPLAG** statement.

Modeling the Irregular Component

The components—such as trend, seasonal and regression effects, and nonstationary cycles—are used to capture the structural dynamics of a response series. In contrast, the stationary cycles and the autoregression are used to capture the transient aspects of the response series that are important for its short-range prediction but have little impact on its long-term forecasts. The irregular component represents the residual variation remaining in the response series that is modeled using an appropriate selection of structural and transient effects. In most cases, the irregular component can be assumed to be simply Gaussian white noise. In some other cases, however, the residual variation can be more complicated. In such situations, it might be necessary to model the irregular component as a stationary ARMA process. Moreover, you can use the ARMA irregular component together with the dependent lag specification (see the **DEPLAG** statement) to specify an $\text{ARIMA}(p, d, q) \times (P, D, Q)_s$ model for the response series. For an explanation of the ARIMA notation, see the **IRREGULAR** statement. For an example of modeling a series by using an $\text{ARIMA}(0, 1, 1) \times (0, 1, 1)_{12}$ model, see [Example 30.8](#).

The Model Parameters

The parameter vector in a UCM consists of the variances of the disturbance terms of the unobserved components, the damping coefficients and frequencies in the cycles, the damping coefficient in the autoregression, and the regression coefficients in the regression terms. These parameters are estimated by maximizing the likelihood. It is possible to restrict the values of the model parameters to user-specified values.

Model Specification

A UCM is specified by describing the components in the model. For example, consider the model

$$y_t = \mu_t + \gamma_t + \epsilon_t$$

consisting of the irregular, level, slope, and seasonal components. This model is called the basic structural model (BSM) by Harvey (1989). The syntax for a BSM with monthly seasonality of trigonometric type is as follows:

```
model y;
irregular;
level;
slope;
season length=12 type=trig;
```

Similarly, the following syntax specifies a BSM with a response variable y , a regressor x , and dummy-type monthly seasonality:

```
model y = x;
irregular;
level;
slope variance=0 noest;
season length=12 type=dummy;
```

Moreover, the disturbance variance of the slope component is restricted to 0, giving rise to a local linear trend with fixed slope.

A model can contain multiple cycle and seasonal components. In such cases the model syntax contains a separate statement for each of these multiple cycle or seasonal components; for example, the syntax for a model containing irregular and level components along with two cycle components could be as follows:

```
model y = x;
irregular;
level;
cycle;
cycle;
```

UCMs as State Space Models

The UCMs considered in PROC UCM are special cases of more general models, called (linear) state space models (SSMs). The section “[State Space Model and Notation](#)” on page 760 in Chapter 14, “[CSSM Procedure](#),” provides an elaborate notation for such models. However, for most of the UCMs considered in PROC UCM, much simpler notation suffices. This section describes a treatment of UCMs in terms of this simplified notation. At times the description and mathematical treatment (such as the expressions of likelihood) of state space models in PROC UCM and PROC SSM can appear different. However, these differences are only notational and the underlying mathematical quantities coincide. For example, the diffuse Kalman filter (DKF) described in this section is called the *exact initial* Kalman filter whereas the DKF described in the section “[Filtering, Smoothing, Likelihood, and Structural Break Detection](#)” on page 767 in Chapter 14, “[CSSM Procedure](#),” is called the *augmented* Kalman filter. Both of these algorithms produce the same final output (see Durbin and Koopman (2012, chap. 5) for more information).

An SSM can be described as follows:

$$\begin{aligned} y_t &= Z_t \alpha_t \\ \alpha_{t+1} &= T_t \alpha_t + \zeta_{t+1}, \quad \zeta_t \sim N(0, Q_t) \\ \alpha_1 &\sim N(0, P) \end{aligned}$$

The first equation, called the *observation equation*, relates the response series y_t to a state vector α_t that is usually unobserved. The second equation, called the *state equation*, describes the evolution of the state vector in time. The system matrices Z_t and T_t are of appropriate dimensions and are known, except possibly for some unknown elements that become part of the parameter vector of the model. The noise series ζ_t consists of independent, zero-mean, Gaussian vectors with covariance matrices Q_t . For most of the UCMs considered here, the system matrices Z_t and T_t , and the noise covariances Q_t , are time invariant—that is, they do not depend on time. In a few cases, however, some or all of them can depend on time. The initial state vector α_1 is assumed to be independent of the noise series, and its covariance matrix P can be partially diffuse. A random vector has a partially diffuse covariance matrix if it can be partitioned such that one part of the vector has a properly defined probability distribution, while the covariance matrix of the other part is infinite—that is, you have no prior information about this part of the vector. The covariance of the initial state α_1 is assumed to have the form

$$P = P_* + \kappa P_\infty$$

where P_* and P_∞ are nonnegative definite, symmetric matrices and κ is a constant that is assumed to be close to ∞ . In the case of UCMs considered here, P_∞ is always a diagonal matrix that consists of zeros and ones, and, if a particular diagonal element of P_∞ is one, then the corresponding row and column in P_* are 0.

The state space formulation of a UCM has many computational advantages. In this formulation there are convenient algorithms for estimating and forecasting the unobserved states $\{\alpha_t\}$ by using the observed series $\{y_t\}$. These algorithms also yield the in-sample and out-of-sample forecasts and the likelihood of $\{y_t\}$. The state space representation of a UCM does not need to be unique. In the representation used here, the unobserved components in the UCM often appear as elements of the state vector. This makes the elements of the state interpretable and, more important, the sample estimates and forecasts of these unobserved components are easily obtained. For additional information about the computational aspects of the state space modeling, see Durbin and Koopman (2012). Next, some notation is developed to describe the essential quantities computed during the analysis of the state space models.

Let $\{y_t, t = 1, \dots, n\}$ be the observed sample from a series that satisfies a state space model. Next, for $1 \leq t \leq n$, let the one-step-ahead forecasts of the series, the states, and their variances be defined as follows, using the usual notation to denote the conditional expectation and conditional variance:

$$\begin{aligned}\hat{\alpha}_t &= E(\alpha_t | y_1, y_2, \dots, y_{t-1}) \\ \Gamma_t &= \text{Var}(\alpha_t | y_1, y_2, \dots, y_{t-1}) \\ \hat{y}_t &= E(y_t | y_1, y_2, \dots, y_{t-1}) \\ F_t &= \text{Var}(y_t | y_1, y_2, \dots, y_{t-1})\end{aligned}$$

These are also called the *filtered* estimates of the series and the states. Similarly, for $t \geq 1$, let the following denote the full-sample estimates of the series and the state values at time t :

$$\begin{aligned}\tilde{\alpha}_t &= E(\alpha_t | y_1, y_2, \dots, y_n) \\ \Delta_t &= \text{Var}(\alpha_t | y_1, y_2, \dots, y_n) \\ \tilde{y}_t &= E(y_t | y_1, y_2, \dots, y_n) \\ G_t &= \text{Var}(y_t | y_1, y_2, \dots, y_n)\end{aligned}$$

If the time t is in the historical period—that is, if $1 \leq t \leq n$ —then the full-sample estimates are called the *smoothed* estimates, and if t lies in the future then they are called out-of-sample forecasts. Note that if $1 \leq t \leq n$, then $\tilde{y}_t = y_t$ and $G_t = 0$, unless y_t is missing.

All the filtered and smoothed estimates ($\hat{\alpha}_t, \tilde{\alpha}_t, \dots, G_t$, and so on) are computed by using the Kalman filtering and smoothing (KFS) algorithm, which is an iterative process. If the initial state is diffuse, as is often the case for the UCMs, its treatment requires modification of the traditional KFS, which is called the diffuse KFS (DKFS). The details of DKFS implemented in the UCM procedure can be found in De Jong and Chu-Chun-Lin (2003). Additional information on the state space models can be found in Durbin and Koopman (2012). The likelihood formulas described in this section are taken from the latter reference.

In the case of diffuse initial condition, the effect of the improper prior distribution of α_1 manifests itself in the first few filtering iterations. During these initial filtering iterations the distribution of the filtered quantities remains diffuse; that is, during these iterations the one-step-ahead series and state forecast variances F_t and Γ_t have the following form:

$$\begin{aligned}F_t &= F_{*t} + \kappa F_{\infty t} \\ \Gamma_t &= \Gamma_{*t} + \kappa \Gamma_{\infty t}\end{aligned}$$

The actual number of iterations—for example, I —affected by this improper prior depends on the nature of the vectors Z_t , the number of nonzero diagonal elements of P_∞ , and the pattern of missing values in the dependent series. After I iterations, $\Gamma_{\infty t}$ and $F_{\infty t}$ become 0 and the one-step-ahead series and state forecasts have proper distributions. These first I iterations constitute the *initialization* phase of the DKFS algorithm. The post-initialization phase of the DKFS and the traditional KFS is the same. In the state space modeling literature the pre-initialization and post-initialization phases are sometimes called *pre-collapse* and *post-collapse* phases of the diffuse Kalman filtering. In certain missing value patterns it is possible for I to exceed the sample size; that is, the sample information can be insufficient to create a proper prior for the filtering process. In these cases, parameter estimation and forecasting is done on the basis of this improper prior, and some or all of the series and component forecasts can have infinite variances (or zero precision).

The forecasts that have infinite variance are set to missing. The same situation can occur if the specified model contains components that are essentially multicollinear. In these situations no residual analysis is possible; in particular, no residuals-based goodness-of-fit statistics are produced.

The log likelihood of the sample (L_∞), which takes account of this diffuse initialization step, is computed by using the one-step-ahead series forecasts as follows,

$$L_\infty(y_1, \dots, y_n) = -\frac{(n-d)}{2} \log 2\pi - \frac{1}{2} \sum_{t=1}^I w_t - \frac{1}{2} \sum_{t=I+1}^n (\log F_t + \frac{v_t^2}{F_t})$$

where d is the number of diffuse elements in the initial state α_1 , $v_t = y_t - Z_t \hat{\alpha}_t$ are the one-step-ahead residuals, and

$$\begin{aligned} w_t &= \log F_{\infty t} && \text{if } F_{\infty t} > 0 \\ &= \log F_{*t} + \frac{v_t^2}{F_{*t}} && \text{if } F_{\infty t} = 0 \end{aligned}$$

If y_t is missing at some time t , then the corresponding summand in the log likelihood expression is deleted, and the constant term is adjusted suitably. Moreover, if the initialization step does not complete—that is, if I exceeds the sample size—then the value of d is reduced to the number of diffuse states that are successfully initialized.

The portion of the log likelihood that corresponds to the post-initialization period is called the nondiffuse log likelihood (L_0). The nondiffuse log likelihood is given by

$$L_0(y_1, \dots, y_n) = -\frac{1}{2} \sum_{t=I+1}^n (\log F_t + \frac{v_t^2}{F_t})$$

In the case of UCMs considered in PROC UCM, it often happens that the diffuse part of the likelihood, $\sum_{t=1}^I w_t$, does not depend on the model parameters, and in these cases the maximization of nondiffuse and diffuse likelihoods is equivalent. However, in some cases, such as when the model consists of dependent lags, the diffuse part does depend on the model parameters. In these cases the maximization of the diffuse and nondiffuse likelihood can produce different parameter estimates.

In some situations it is convenient to reparameterize the nondiffuse initial state covariance P_* as $\sigma^2 P_*$ and the state noise covariance Q_t as $\sigma^2 Q_t$ for some common scalar parameter σ^2 . In this case the preceding log-likelihood expression, up to a constant, can be written as

$$L_\infty(y_1, \dots, y_n) = -\frac{1}{2} \sum_{t=1}^I w_t - \frac{1}{2} \sum_{t=I+1}^n \log F_t - \frac{1}{2\sigma^2} \sum_{t=I+1}^n \frac{v_t^2}{F_t} - \frac{(n-d)}{2} \log \sigma^2$$

Solving analytically for the optimum, the maximum likelihood estimate of σ^2 can be shown to be

$$\hat{\sigma}^2 = \frac{1}{(n-d)} \sum_{t=I+1}^n \frac{v_t^2}{F_t}$$

When this expression of σ^2 is substituted back into the likelihood formula, an expression called the *profile likelihood* (L_{profile}) of the data is obtained:

$$-2L_{\text{profile}}(y_1, \dots, y_n) = \sum_{t=1}^I w_t + \sum_{t=I+1}^n \log F_t + (n-d) \log \left(\sum_{t=I+1}^n \frac{v_t^2}{F_t} \right)$$

In some situations the parameter estimation is done by optimizing the profile likelihood (see the section “[Parameter Estimation by Profile Likelihood Optimization](#)” on page 2174 and the [PROFILE](#) option in the ESTIMATE statement).

You can also request that parameter estimation be based on an alternate form of the likelihood, called the marginal likelihood ($L_m(\mathbf{Y}, \boldsymbol{\theta})$). You can switch to the marginal-likelihood-based parameter estimation by specifying [LIKE=MARGINAL](#) in the ESTIMATE statement. This alternate likelihood and two additional likelihoods are described in the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769 in Chapter 14, “[CSSM Procedure](#).” The diffuse likelihood, L_∞ , described in this section is equivalent to the diffuse likelihood, $L_d(\mathbf{Y}, \boldsymbol{\theta})$, described in that section. However, do not confuse the profile likelihood, L_{profile} , described in this section with the profile likelihood, $L_p(\mathbf{Y}, \boldsymbol{\theta})$, described in that section. The profiling in $L_p(\mathbf{Y}, \boldsymbol{\theta})$ refers to the profiling of the diffuse effects, whereas the profiling in L_{profile} refers to the profiling of a common scalar parameter σ^2 . For each of the three likelihoods—diffuse, marginal and profile—that are described in that section, it is possible to profile out (also called concentrate out) a common scalar parameter σ^2 and obtain expressions similar to the L_{profile} likelihood that is described in this section. In fact, when you request that parameter estimation be based on the marginal likelihood by specifying [LIKE=MARGINAL](#) in the ESTIMATE statement, the profile version of marginal likelihood ($L_m(\mathbf{Y}, \boldsymbol{\theta})$) is used if the [PROFILE](#) option is in effect (by default or when the [PROFILE](#) option is specified). The discussion in the section “[Parameter Estimation by Profile Likelihood Optimization](#)” on page 2174 also applies to marginal likelihood. As explained in the section “[Likelihood Computation and Model-Fitting Phase](#)” on page 769 in Chapter 14, “[CSSM Procedure](#),” the estimates that are based on marginal likelihood and the estimates that are based on diffuse likelihood coincide in many cases. In PROC UCM, estimates that are based on marginal likelihood and diffuse likelihood will differ only if at least one of the following conditions holds:

- The [DEPLAG](#) statement is present and the NOEST option is not specified.
- In a [TF](#) statement, at least one denominator factor is present and the NOEST option is not specified.
- In a [CYCLE](#) statement, RHO is fixed at 1 and the period is to be estimated—that is, $\text{RHO}=1$ and $\text{NOEST}=\text{RHO}$ or $\text{NOEST}=(\text{RHO VARIANCE})$.

Whenever you specify [LIKE=MARGINAL](#) in the ESTIMATE statement, the FitSummary table that displays the likelihood-based fit statistics includes fit statistics and information criteria that are based on the marginal likelihood in addition to fit statistics that are based on diffuse likelihood.

In the remainder of this section, the state space formulation of UCMs is further explained by using some particular UCMs as examples. The examples show that the state space formulation of the UCMs depends on the components in the model in a simple fashion; for example, the system matrix T is usually a block diagonal matrix with blocks that correspond to the components in the model. The only exception to this pattern is the UCMs that consist of the lags of dependent variable. This case is considered at the end of the section.

In what follows, $\text{Diag}[a, b, \dots]$ denotes a diagonal matrix with diagonal entries $[a, b, \dots]$, and the transpose of a matrix T is denoted as T' .

Locally Linear Trend Model

Recall that the dynamics of the locally linear trend model are

$$\begin{aligned} y_t &= \mu_t + \epsilon_t \\ \mu_t &= \mu_{t-1} + \beta_{t-1} + \eta_t \\ \beta_t &= \beta_{t-1} + \xi_t \end{aligned}$$

Here y_t is the response series and ϵ_t , η_t , and ξ_t are independent, zero-mean Gaussian disturbance sequences with variances σ_ϵ^2 , σ_η^2 , and σ_ξ^2 , respectively. This model can be formulated as a state space model where the state vector $\alpha_t = [\epsilon_t \mu_t \beta_t]'$ and the state noise $\zeta_t = [\epsilon_t \eta_t \xi_t]'$. Note that the elements of the state vector are precisely the unobserved components in the model. The system matrices T and Z and the noise covariance Q corresponding to this choice of state and state noise vectors can be seen to be time invariant and are given by

$$Z = [1 \ 1 \ 0], \quad T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad Q = \text{Diag}[\sigma_\epsilon^2, \sigma_\eta^2, \sigma_\xi^2]$$

The distribution of the initial state vector α_1 is diffuse, with $P_* = \text{Diag}[\sigma_\epsilon^2, 0, 0]$ and $P_\infty = \text{Diag}[0, 1, 1]$. The parameter vector θ consists of all the disturbance variances—that is, $\theta = (\sigma_\epsilon^2, \sigma_\eta^2, \sigma_\xi^2)$.

Basic Structural Model

The basic structural model (BSM) is obtained by adding a seasonal component, γ_t , to the local level model. In order to economize on the space, the state space formulation of a BSM with a relatively short season length, season length = 4 (quarterly seasonality), is considered here. The pattern for longer season lengths such as 12 (monthly) and 52 (weekly) is easy to see.

Let us first consider the dummy form of seasonality. In this case the state and state noise vectors are $\alpha_t = [\epsilon_t \mu_t \beta_t \gamma_{1,t} \gamma_{2,t} \gamma_{3,t}]'$ and $\zeta_t = [\epsilon_t \eta_t \xi_t \omega_t 0 0]'$, respectively. The first three elements of the state vector are the irregular, level, and slope components, respectively. The remaining elements, $\gamma_{i,t}$, are lagged versions of the seasonal component γ_t . $\gamma_{1,t}$ corresponds to lag zero—that is, the same as γ_t , $\gamma_{2,t}$ to lag 1 and $\gamma_{3,t}$ to lag 2. The system matrices are

$$Z = [1 \ 1 \ 0 \ 1 \ 0 \ 0], \quad T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

and $Q = \text{Diag}[\sigma_\epsilon^2, \sigma_\eta^2, \sigma_\xi^2, \sigma_\omega^2, 0, 0]$. The distribution of the initial state vector α_1 is diffuse, with $P_* = \text{Diag}[\sigma_\epsilon^2, 0, 0, 0, 0, 0]$ and $P_\infty = \text{Diag}[0, 1, 1, 1, 1, 1]$.

In the case of the trigonometric type of seasonality, $\alpha_t = [\epsilon_t \mu_t \beta_t \gamma_{1,t} \gamma_{1,t}^* \gamma_{2,t}]'$ and $\zeta_t = [\epsilon_t \eta_t \xi_t \omega_{1,t} \omega_{1,t}^* \omega_{2,t}]'$. The disturbance sequences, $\omega_{j,t}$, $1 \leq j \leq 2$, and $\omega_{1,t}^*$, are independent, zero-

mean, Gaussian sequences with variance σ_ω^2 . The system matrices are

$$Z = [1 \ 1 \ 0 \ 1 \ 0 \ 1], \quad T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos \lambda_1 & \sin \lambda_1 & 0 \\ 0 & 0 & 0 & -\sin \lambda_1 & \cos \lambda_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cos \lambda_2 \end{bmatrix}$$

and $Q = \text{Diag}[\sigma_\epsilon^2, \sigma_\eta^2, \sigma_\xi^2, \sigma_\omega^2, \sigma_\omega^2, \sigma_\omega^2]$. Here $\lambda_j = (2\pi j)/4$. The distribution of the initial state vector α_1 is diffuse, with $P_* = \text{Diag}[\sigma_\epsilon^2, 0, 0, 0, 0, 0]$ and $P_\infty = \text{Diag}[0, 1, 1, 1, 1, 1]$. The parameter vector in both the cases is $\theta = (\sigma_\epsilon^2, \sigma_\eta^2, \sigma_\xi^2, \sigma_\omega^2)$.

Seasons with Blocked Seasonal Values

Block seasonals are special seasonal components that impose a special block structure on the seasonal effects. Let us consider a BSM with monthly seasonality that has a quarterly block structure—that is, months within the same quarter are assumed to have identical effects except for some random perturbation. Such a seasonal component is a block seasonal with block size m equal to 3 and the number of blocks k equal to 4. The state space structure for such a model with dummy-type seasonality is as follows: The state and state noise vectors are $\alpha_t = [\epsilon_t \ \mu_t \ \beta_t \ \gamma_{1,t} \ \gamma_{2,t} \ \gamma_{3,t}]'$ and $\zeta_t = [\epsilon_t \ \eta_t \ \xi_t \ \omega_t \ 0 \ 0]'$, respectively. The first three elements of the state vector are the irregular, level, and slope components, respectively. The remaining elements, $\gamma_{i,t}$, are lagged versions of the seasonal component γ_t . $\gamma_{1,t}$ corresponds to lag zero—that is, the same as γ_t , $\gamma_{2,t}$ to lag m and $\gamma_{3,t}$ to lag $2m$. All the system matrices are time invariant, except the matrix T . They can be seen to be $Z = [1 \ 1 \ 0 \ 1 \ 0 \ 0]$, $Q = \text{Diag}[\sigma_\epsilon^2, \sigma_\eta^2, \sigma_\xi^2, \sigma_\omega^2, 0, 0]$, and

$$T_t = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

when t is a multiple of the block size m , and

$$T_t = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

otherwise. Note that when t is not a multiple of m , the portion of the T_t matrix corresponding to the seasonal is identity. The distribution of the initial state vector α_1 is diffuse, with $P_* = \text{Diag}[\sigma_\epsilon^2, 0, 0, 0, 0, 0]$ and $P_\infty = \text{Diag}[0, 1, 1, 1, 1, 1]$.

Similarly, in the case of the trigonometric form of seasonality, $\alpha_t = [\epsilon_t \ \mu_t \ \beta_t \ \gamma_{1,t} \ \gamma_{1,t}^* \ \gamma_{2,t}]'$ and $\zeta_t = [\epsilon_t \ \eta_t \ \xi_t \ \omega_{1,t} \ \omega_{1,t}^* \ \omega_{2,t}]'$. The disturbance sequences, $\omega_{j,t}$, $1 \leq j \leq 2$, and $\omega_{1,t}^*$, are independent, zero-mean, Gaussian sequences with variance σ_ω^2 . $Z = [1 \ 1 \ 0 \ 1 \ 0 \ 1]$, $Q = \text{Diag}[\sigma_\epsilon^2, \sigma_\eta^2, \sigma_\xi^2, \sigma_\omega^2, \sigma_\omega^2, \sigma_\omega^2]$,

and

$$T_t = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos \lambda_1 & \sin \lambda_1 & 0 \\ 0 & 0 & 0 & -\sin \lambda_1 & \cos \lambda_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cos \lambda_2 \end{bmatrix}$$

when t is a multiple of the block size m , and

$$T_t = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

otherwise. As before, when t is not a multiple of m , the portion of the T_t matrix corresponding to the seasonal is identity. Here $\lambda_j = (2\pi j)/4$. The distribution of the initial state vector α_1 is diffuse, with $P_* = \text{Diag}[\sigma_\epsilon^2, 0, 0, 0, 0, 0]$ and $P_\infty = \text{Diag}[0, 1, 1, 1, 1, 1]$. The parameter vector in both the cases is $\theta = (\sigma_\epsilon^2, \sigma_\eta^2, \sigma_\xi^2, \sigma_\omega^2)$.

Cycles and Autoregression

The preceding examples have illustrated how to build a state space model corresponding to a UCM that includes components such as irregular, trend, and seasonal. There you can see that the state vector and the system matrices have a simple block structure with blocks corresponding to the components in the model. Therefore, here only a simple model consisting of a single cycle and an irregular component is considered. The state space form for more complex UCMs consisting of multiple cycles and other components can be easily deduced from this example.

Recall that a stochastic cycle ψ_t with frequency λ , $0 < \lambda < \pi$, and damping coefficient ρ can be modeled as

$$\begin{bmatrix} \psi_t \\ \psi_t^* \end{bmatrix} = \rho \begin{bmatrix} \cos \lambda & \sin \lambda \\ -\sin \lambda & \cos \lambda \end{bmatrix} \begin{bmatrix} \psi_{t-1} \\ \psi_{t-1}^* \end{bmatrix} + \begin{bmatrix} \nu_t \\ \nu_t^* \end{bmatrix}$$

where ν_t and ν_t^* are independent, zero-mean, Gaussian disturbances with variance σ_ν^2 . In what follows, a state space form for a model consisting of such a stochastic cycle and an irregular component is given.

The state vector $\alpha_t = [\epsilon_t \ \psi_t \ \psi_t^*]'$, and the state noise vector $\zeta_t = [\epsilon_t \ \nu_t \ \nu_t^*]'$. The system matrices are

$$Z = [1 \ 1 \ 0] \quad T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \rho \cos \lambda & \rho \sin \lambda \\ 0 & -\rho \sin \lambda & \rho \cos \lambda \end{bmatrix} \quad Q = \text{Diag}[\sigma_\epsilon^2, \sigma_\nu^2, \sigma_\nu^2]$$

The distribution of the initial state vector α_1 is proper, with $P_* = \text{Diag}[\sigma_\epsilon^2, \sigma_\psi^2, \sigma_\psi^2]$, where $\sigma_\psi^2 = \sigma_\nu^2(1 - \rho^2)^{-1}$. The parameter vector $\theta = (\sigma_\epsilon^2, \rho, \lambda, \sigma_\nu^2)$.

An autoregression r_t can be considered as a special case of cycle with frequency λ equal to 0 or π . In this case the equation for ψ_t^* is not needed. Therefore, for a UCM consisting of an autoregressive component and an irregular component, the state space model simplifies to the following form.

The state vector $\alpha_t = [\epsilon_t \ r_t]'$, and the state noise vector $\zeta_t = [\epsilon_t \ v_t]'$. The system matrices are

$$Z = [1 \ 1], \quad T = \begin{bmatrix} 0 & 0 \\ 0 & \rho \end{bmatrix} \quad \text{and} \quad Q = \text{Diag}[\sigma_\epsilon^2, \sigma_v^2]$$

The distribution of the initial state vector α_1 is proper, with $P_* = \text{Diag}[\sigma_\epsilon^2, \sigma_r^2]$, where $\sigma_r^2 = \sigma_v^2(1 - \rho^2)^{-1}$. The parameter vector $\theta = (\sigma_\epsilon^2, \rho, \sigma_v^2)$.

Incorporating Predictors of Different Types

In the UCM procedure, you can incorporate predictors in a UCM in a variety of ways: you can specify simple time-invariant linear predictors in the **MODEL** statement, you can specify predictors that have time-varying coefficients in the **RANDOMREG** statement, and you can specify predictors that have a nonlinear relationship with the response variable in the **SPLINEREG** statement. You can also specify a transfer-function relationship by using the **TF** statement. As with earlier examples, the first part of this section uses a simple special case to show how to obtain a state space form of a UCM that consists of a variety of predictors (except the transfer-function relationship). The state space form that is associated with a transfer-function relationship is described in the section “**State Space Form of a Transfer Function Relationship**” on page 2170.

Consider a random walk trend model that has predictors x , u_1 , u_2 , and v . Assume that x is a simple regressor that is specified in the **MODEL** statement, u_1 and u_2 are random regressors with time-varying regression coefficients that are specified in the same **RANDOMREG** statement, and v is a nonlinear regressor that is specified in a **SPLINEREG** statement. Further assume that the spline that is associated with v has degree one and is based on two internal knots. As explained in the section “**SPLINEREG Statement**” on page 2151, using v is equivalent to using $(n \text{ knots} + \text{degree}) = (2 + 1) = 3$ derived (random) regressors: for example, s_1, s_2, s_3 . There are $(1 + 2 + 3) = 6$ regressors in all, the first one being a simple regressor and the others being time-varying coefficient regressors. The time-varying regressors are in two groups: the first group consists of u_1 and u_2 , and the other group consists of s_1, s_2 , and s_3 . The dynamics of this model are as follows:

$$\begin{aligned} y_t &= \mu_t + \beta x_t + \kappa_{1t} u_{1t} + \kappa_{2t} u_{2t} + \sum_{i=1}^3 \gamma_{it} s_{it} + \epsilon_t \\ \mu_t &= \mu_{t-1} + \eta_t \\ \kappa_{1t} &= \kappa_{1(t-1)} + \xi_{1t} \\ \kappa_{2t} &= \kappa_{2(t-1)} + \xi_{2t} \\ \gamma_{1t} &= \gamma_{1(t-1)} + \zeta_{1t} \\ \gamma_{2t} &= \gamma_{2(t-1)} + \zeta_{2t} \\ \gamma_{3t} &= \gamma_{3(t-1)} + \zeta_{3t} \end{aligned}$$

All the disturbances $\epsilon_t, \eta_t, \xi_{1t}, \xi_{2t}, \zeta_{1t}, \zeta_{2t}$, and ζ_{3t} are independent, zero-mean, Gaussian variables, where ξ_{1t}, ξ_{2t} share a common variance parameter σ_ξ^2 and $\zeta_{1t}, \zeta_{2t}, \zeta_{3t}$ share a common variance σ_ζ^2 . These dynamics can be captured in the state space form by taking state $\alpha_t = [\epsilon_t \ \mu_t \ \beta \ \kappa_{1t} \ \kappa_{2t} \ \gamma_{1t} \ \gamma_{2t} \ \gamma_{3t}]'$, state disturbance $\zeta_t = [\epsilon_t \ \eta_t \ 0 \ \xi_{1t} \ \xi_{2t} \ \zeta_{1t} \ \zeta_{2t} \ \zeta_{3t}]'$, and the system matrices

$$\begin{aligned} Z_t &= [1 \ 1 \ x_t \ u_{1t} \ u_{2t} \ s_{1t} \ s_{2t} \ s_{3t}] \\ T &= \text{Diag}[0, 1, 1, 1, 1, 1, 1, 1] \\ Q &= \text{Diag}[\sigma_\epsilon^2, \sigma_\eta^2, 0, \sigma_\xi^2, \sigma_\xi^2, \sigma_\zeta^2, \sigma_\zeta^2, \sigma_\zeta^2] \end{aligned}$$

Note that the regression coefficients are elements of the state vector and that the system vector Z_t is not time-invariant. The distribution of the initial state vector α_1 is diffuse, with $P_* = \text{Diag}[\sigma_\epsilon^2, 0, 0, 0, 0, 0, 0, 0]$ and $P_\infty = \text{Diag}[0, 1, 1, 1, 1, 1, 1, 1]$. The parameters of this model are the disturbance variances, σ_ϵ^2 , σ_η^2 , σ_ξ^2 , and σ_ζ^2 , which are estimated by maximizing the likelihood. The regression coefficients, time-invariant β , and time-varying κ_{1t} , κ_{2t} , γ_{1t} , γ_{2t} and γ_{3t} are implicitly estimated during the state estimation (smoothing).

State Space Form of a Transfer Function Relationship

This section illustrates the state space form of a simple transfer-function relationship. The state space form of more complicated transfer-function relationships can be deduced using the same logic. Suppose that a predictor x enters the model for a response variable y as

$$\begin{aligned} y_t &= f_t + \epsilon_t \\ f_t &= \frac{(\gamma_0 + \gamma_1 B)}{(1 - \delta_1 B - \delta_2 B^2)} x_t \end{aligned}$$

where f_t is the transfer-function component and ϵ_t is a sequence of independent, zero-mean, Gaussian variables. In this description, the transfer-function component is described using the backward shift operator B . Alternatively, it can be described as follows:

$$f_t = \delta_1 f_{t-1} + \delta_2 f_{t-2} + \gamma_0 x_t + \gamma_1 x_{t-1}$$

This model can be easily put in a state space form by taking state $\alpha_t = (\epsilon_t \ f_t \ f_{t-1} \ \gamma_0 \ \gamma_1)'$, state disturbance $\zeta_t = (\epsilon_t \ 0 \ 0 \ 0 \ 0)'$, the system matrices $Z = [1 \ 1 \ 0 \ 0 \ 0]$, $Q = \text{Diag}[\sigma_\epsilon^2 \ 0 \ 0 \ 0 \ 0]$, and

$$T_t = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \delta_1 & \delta_2 & x_{t+1} & x_t \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The initial state α_1 is partially diffuse. The precise form of the initial state depends on the value of the **TFSTART=** option in the **TF** statement. If the **TFSTART=** option is not specified, all elements of α_1 except for the first element (ϵ_1) are treated as diffuse. On the other hand, if a value is specified in the **TFSTART=** option, the initial transfer function values (f_1 and f_0) in α_1 are fixed at that specified value. In this formulation of the model, the numerator coefficients of the transfer-function relationship (γ_0 and γ_1) are part of the state. They are implicitly estimated during the state estimation (smoothing). On the other hand, the denominator coefficients (δ_1 and δ_2) and the noise variance (σ_ϵ^2) are estimated by maximizing the likelihood.

Reporting Parameter Estimates for Random Regressors

If the random walk disturbance variance that is associated with a random regressor is held fixed at 0, then its coefficient is no longer time-varying. In the UCM procedure, the random regressor parameter estimates are reported differently if the random walk disturbance variance that is associated with a random regressor is held fixed at 0. The following points explain how the parameter estimates are reported in the parameter estimates table and in the **OUTEST=** data set:

- If the random walk disturbance variance that is associated with a random regressor is not held fixed, then its estimate is reported in the parameter estimates table and in the **OUTEST=** data set.

- If more than one random regressor is specified in a **RANDOMREG** statement, then the first regressor in the list is used as a representative of the list when the corresponding common variance parameter estimate is reported.
- If the random walk disturbance variance is held fixed at 0, then the parameter estimates table and the OUTEST= data set contain the corresponding regression parameter estimate rather than the variance parameter estimate.
- Similar considerations apply in the case of the derived random regressors that are associated with a spline regressor.

Forecasting with Predictor Variables

If regression effects are included in the model (in a **MODEL** statement or in one or more of the **RANDOMREG**, **SPLINEREG**, and **TF** statements) and the **FORECAST** statement is used to compute multistep forecasts, then future values of the predictor variables must be included in the **DATA=** data set for the forecast horizon that is defined by the **BACK=** and **LEAD=** options in the **FORECAST** statement. For more information about how the forecast horizon is defined, see the **FORECAST** statement.

ARMA Irregular Component

The state space form for the irregular component that follows an $\text{ARMA}(p, q) \times (P, Q)_s$ model is described in this section. The notation for ARMA models is explained in the **IRREGULAR** statement. A number of alternate state space forms are possible in this case; the one given here is based on Jones (1980). With slight abuse of notation, let $p = p + sP$ denote the effective autoregressive order and $q = q + sQ$ denote the effective moving average order of the model. Similarly, let ϕ be the effective autoregressive polynomial and θ be the effective moving average polynomial in the backshift operator with coefficients $\phi_1, \dots, \phi_p$ and $\theta_1, \dots, \theta_q$, obtained by multiplying the respective nonseasonal and seasonal factors. Then, a random sequence ϵ_t that follows an $\text{ARMA}(p, q) \times (P, Q)_s$ model with a white-noise sequence a_t has a state space form with state vector of size $m = \max(p, q + 1)$. The system matrices, which are time invariant, are as follows: $Z = [1 \ 0 \ \dots \ 0]$. The state transition matrix T , in a blocked form, is given by

$$T = \begin{bmatrix} 0 & I_{m-1} \\ \phi_m & \dots & \phi_1 \end{bmatrix}$$

where $\phi_i = 0$ if $i > p$ and I_{m-1} is an $(m - 1)$ dimensional identity matrix. The covariance of the state disturbance matrix $Q = \sigma^2 \psi \psi'$ where σ^2 is the variance of the white-noise sequence a_t and the vector $\psi = [\psi_0 \ \dots \ \psi_{m-1}]'$ contains the first m values of the impulse response function—that is, the first m coefficients in the expansion of the ratio θ/ϕ . Since ϵ_t is a stationary sequence, the initial state is nondiffuse and $P_\infty = 0$. The description of P_* , the covariance matrix of the initial state, is a little involved; the details are given in Jones (1980).

Models with Dependent Lags

The state space form of a UCM consisting of the lags of the dependent variable is quite different from the state space forms considered so far. Let us consider an example to illustrate this situation. Consider a model that has random walk trend, two simple time-invariant regressors, and that also includes a few—for example,

k —lags of the dependent variable. That is,

$$\begin{aligned} y_t &= \sum_{i=1}^k \phi_i y_{t-i} + \mu_t + \beta_1 x_{1t} + \beta_2 x_{2t} + \epsilon_t \\ \mu_t &= \mu_{t-1} + \eta_t \end{aligned}$$

The state space form of this augmented model can be described in terms of the state space form of a model that has random walk trend with two simple time-invariant regressors. A superscript dagger ($\dagger$) has been added to distinguish the augmented model state space entities from the corresponding entities of the state space form of the random walk with predictors model. With this notation, the state vector of the augmented model $\alpha_t^\dagger = [\alpha_t' y_t y_{t-1} \dots y_{t-k+1}]'$ and the new state noise vector $\zeta_t^\dagger = [\zeta_t' u_t 0 \dots 0]'$, where u_t is the matrix product $Z_t \zeta_t$. Note that the length of the new state vector is $k + \text{length}(\alpha_t) = k + 4$. The new system matrices, in block form, are

$$Z_t^\dagger = [0 \ 0 \ 0 \ 0 \ 1 \ \dots \ 0], \quad T_t^\dagger = \begin{bmatrix} T_t & 0 & \dots & 0 \\ Z_{t+1} T_t & \phi_1 & \dots & \phi_k \\ 0 & I_{k-1, k-1} & & 0 \end{bmatrix}$$

where $I_{k-1, k-1}$ is the $k - 1$ dimensional identity matrix and

$$Q_t^\dagger = \begin{bmatrix} Q_t & Q_t Z_t' & 0 \\ Z_t Q_t & Z_t Q_t Z_t' & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Note that the T and Q matrices of the random walk with predictors model are time invariant, and in the expressions above their time indices are kept because they illustrate the pattern for more general models. The initial state vector is diffuse, with

$$P_*^\dagger = \begin{bmatrix} P_* & 0 \\ 0 & 0 \end{bmatrix}, \quad P_\infty^\dagger = \begin{bmatrix} P_\infty & 0 \\ 0 & I_{k, k} \end{bmatrix}$$

The parameters of this model are the disturbance variances σ_ϵ^2 and σ_η^2 , the lag coefficients $\phi_1, \phi_2, \dots, \phi_k$, and the regression coefficients β_1 and β_2 . As before, the regression coefficients get estimated during the state smoothing, and the other parameters are estimated by maximizing the likelihood.

Outlier Detection

In time series analysis it is often useful to detect changes over time in the characteristics of the response series. In the UCM procedure you can search for two types of changes, additive outliers (AO) and level shifts (LS). An additive outlier is an unusual value in the series, the cause of which might be a data recording error or a temporary shock to the series generation process. A level shift represents a permanent shift, either up or down, in the level of the series. You can control different aspects of the outlier search, such as the significance level of the reported outliers, by choosing different options in the **OUTLIER** statement. The search for AOs is done by default, whereas the **CHECKBREAK** option in the **LEVEL** statement must be used to turn on the search for LSs.

The outlier detection process implemented in the UCM procedure is based on De Jong and Penzer (1998). In this approach the fitted model is taken to be the *null* model, and the series values and level shifts that are

not adequately accounted for by the null model are flagged as outliers. The unusualness of a response series value at a particular time point t_0 , with respect to the fitted model, can be judged by estimating its value based on the rest of the data (that is, the series obtained by *deleting* the series value at t_0) and comparing the estimated value to the observed value. If the difference between the estimated and observed values is statistically significant, then such value can be regarded as an AO. Note that this difference between the estimated and observed values is also the regression coefficient of a *dummy* regressor that takes the value 1.0 at t_0 and is 0.0 elsewhere, assuming such a regressor is added to the null model. In this way the series value at t_0 is regarded as AO if the regression coefficient of this dummy regressor is significant. Similarly, you can say that a level shift has occurred at a time point t_0 if the regression coefficient of a regressor, which is 0.0 before t_0 and 1.0 at t_0 and thereafter, is statistically significant. De Jong and Penzer (1998) provide an efficient way to compute such AO and LS regression coefficients and their standard errors at all time points in the series. The outlier summary table, which is produced by default, simply lists the most statistically significant candidates among these.

Missing Values

Embedded missing values in the dependent variable usually cause no problems in UCM modeling. However, no missing values are allowed in the predictor variables. Certain patterns of missing values in the dependent variable can lead to failure of the initialization step of the diffuse Kalman filtering for some models. For example, if in a monthly series all values are missing for a certain month—such as May—then a BSM with monthly seasonality leads to such a situation. However, in this case the initialization step can complete successfully for a nonseasonal model such as local linear model.

Parameter Estimation

The parameter vector in a UCM consists of the variances of the disturbance terms of the unobserved components, the damping coefficients and frequencies in the cycles, the damping coefficient in the autoregression, the lag coefficients of the dependent lags, and the regression coefficients in the regression terms. The regression coefficients are always part of the state vector and are estimated by state smoothing. The remaining parameters are estimated by maximizing either the full diffuse likelihood or the nondiffuse likelihood. The decision to use the full diffuse likelihood or the nondiffuse likelihood depends on the presence or absence of the dependent lag coefficients in the parameter vector. If the parameter vector does not contain any dependent lag coefficients, then the full diffuse likelihood is used. If, on the other hand, the parameter vector does contain some dependent lag coefficients, then the parameters are estimated by maximizing the nondiffuse likelihood. The optimization of the full diffuse likelihood is often unstable when the parameter vector contains dependent lag coefficients. In this sense, when the parameter vector contains dependent lag coefficients, the parameter estimates are not true maximum likelihood estimates.

The optimization of the likelihood, either full or nondiffuse, is carried out using one of several nonlinear optimization algorithms. The user can control many aspects of the optimization process by using the **NLOPTIONS** statement and by providing the starting values of the parameters while specifying the corresponding components. However, in most cases the default settings work quite well. The optimization process is not guaranteed to converge to a maximum likelihood estimate. In most cases the difficulties in parameter estimation are associated with the specification of a model that is not appropriate for the series being modeled.

Parameter Estimation by Profile Likelihood Optimization

If a disturbance variance, such as the disturbance variance of the irregular component, is a part of the UCM and is a free parameter, then it can be profiled out of the likelihood. This means solving analytically for its optimum and plugging this expression back into the likelihood formula, giving rise to the so-called *profile* likelihood. The expression of the profile likelihood and the MLE of the profiled variance are given earlier in the section “[UCMs as State Space Models](#)” on page 2162, where the computation of the likelihood of the state space model is also discussed.

In some situations the optimization of the profile likelihood can be more efficient because the number of parameters to optimize is reduced by one; however, for a variety of reasons such gains might not always be observed. Moreover, in theory the estimates obtained by optimizing the profile likelihood and the usual likelihood should be the same, but in practice this might not hold because of numerical rounding and other conditions.

In the UCM procedure, by default the usual likelihood is optimized if any of the disturbance variance parameters is held fixed to a nonzero value by using the NOEST option in the corresponding component statement. In other cases the decision whether to optimize the profile likelihood or the usual likelihood is based on several factors that are difficult to document. You can choose which likelihood to optimize during parameter estimation by specifying the **PROFILE** option for the profile likelihood optimization or the **NOPROFILE** option for the usual likelihood optimization. In the presence of the **PROFILE** option, the disturbance variance to profile is checked in a specific order, so that if the irregular component disturbance variance is free then it is always chosen. The situation in other cases is more complicated.

Profiling in the Presence of Fixed Variance Parameters

Note that when the parameter estimation is done by optimizing the profile likelihood, the interpretation of the variance parameters that are held fixed to nonzero values changes. In the presence of the **PROFILE** option, the disturbance variances that are held at a fixed value by using the NOEST option in their respective component statements are interpreted as being restricted to be that fixed multiple of the profiled variance rather than being fixed at that nominal value. That is, implicitly, the parameter estimation is done under the restriction of holding the disturbance variance *ratio* fixed at a given value rather than the disturbance variance itself. For an example of this type of restriction to obtain a UC model that is equivalent to the famous Hodrick-Prescott filter, see [Example 30.5](#).

t Values

The *t* values reported in the table of parameter estimates are approximations whose accuracy depends on the validity of the model, the nature of the model, and the length of the observed series. The distributional properties of the maximum likelihood estimates of general unobserved components models have not been explored fully; therefore the probability values that correspond to a *t* distribution should be interpreted carefully, as they can be misleading. This is particularly true if the parameters in question are close to the boundary of the parameter space. The two sources by Harvey (1989, 2001) are good references for information about this topic. For some parameters, such as the cycle period, the reported *t* values are uninformative because comparison of the estimated parameter with 0 is never needed. In such cases the *t* values and the corresponding probability values should be ignored.

Computational Issues

Convergence Problems

As explained in the section “[Parameter Estimation](#)” on page 2173, the model parameters are estimated by nonlinear optimization of the likelihood. This process is not guaranteed to succeed. For some data sets, the optimization algorithm can fail to converge. Nonconvergence can result from a number of causes, including flat or ridged likelihood surfaces and ill-conditioned data. It is also possible for the algorithm to converge to a point that is not the global optimum of the likelihood.

If you experience convergence problems, the following points might be helpful:

- Data that are extremely large or extremely small can adversely affect results because of the internal tolerances used during the filtering steps of the likelihood calculation. Rescaling the data can improve stability.
- Examine your model for redundancies in the included components and regressors. If some of the included components or regressors are nearly collinear to each other, then the optimization process can become unstable.
- Experimenting with different options offered by the [NLOPTIONS](#) statement can help.
- Lack of convergence can indicate model misspecification or a violation of the normality assumption.

Computer Resource Requirements

The computing resources required for the UCM procedure depend on several factors. The memory requirement for the procedure is largely dependent on the number of observations to be processed and the size of the state vector underlying the specified model. If n denotes the sample size and m denotes the size of the state vector, the memory requirement for the smoothing stage of the Kalman filter is of the order of $6 \times 8 \times n \times m^2$ bytes, ignoring the lower-order terms. If the smoothed component estimates are not needed then the memory requirement is of the order of $6 \times 8 \times (m^2 + n)$ bytes. Besides m and n , the computing time for the parameter estimation depends on the type of components included in the model. For example, the parameter estimation is usually faster if the model parameter vector consists only of disturbance variances, because in this case there is an efficient way to compute the likelihood gradient.

Displayed Output

The default printed output produced by the UCM procedure is described in the following list:

- brief information about the input data set, including the data set name and label, and the name of the ID variable specified in the ID statement
- summary statistics for the data in the estimation and forecast spans, including the names of the variables in the model, their categorization as dependent or predictor, the index of the beginning and ending observations in the spans, the total number of observations and the number of missing observations, the smallest and largest measurements, and the mean and standard deviation

- information about the model parameters at the start of the model-fitting stage, including the fixed parameters in the model and the initial estimates of the free parameters in the model
- convergence status of the likelihood optimization process if any parameter estimation is done
- estimates of the free parameters at the end of the model fitting-stage, including the parameter estimates, their approximate standard errors, t statistics, and the approximate p -value
- the likelihood-based goodness-of-fit statistics, including the full likelihood, the portion of the likelihood corresponding to the diffuse initialization, the sum of squares of residuals normalized by their standard errors, and the information criteria: AIC, AICC, HQIC, BIC, and CAIC
- the fit statistics that are based on the raw residuals (observed minus predicted), including the mean squared error (MSE), the root mean squared error (RMSE), the mean absolute percentage error (MAPE), the maximum percentage error (MAXPE), the R-square, the adjusted R-square, the random walk R-square, and Amemiya's R-square
- the significance analysis of the components included in the model that is based on the estimation span
- brief information about the components included in the model
- additive outliers in the series, if any are detected
- the multistep series forecasts
- post-sample-prediction analysis table that compares the multistep forecasts with the observed series values, if the BACK= option is used in the FORECAST statement

Statistical Graphics

Statistical procedures use ODS Graphics to create graphs as part of their output. ODS Graphics is described in detail in Chapter 24, “Statistical Graphics Using ODS” (*SAS/STAT User's Guide*).

This section provides information about the basic ODS statistical graphics produced by the UCM procedure.

You can obtain most plots relevant to the specified model by using the global **PLOTS=** option in the PROC UCM statement. The plot of series forecasts in the forecast horizon is produced by default. You can further control the production of individual plots by using the **PLOT=** options in the different statements.

The main types of plots available are as follows:

- Time series plots of the component estimates, either filtered or smoothed, can be requested by using the **PLOT=** option in the respective component statements. For example, the use of **PLOT=SMOOTH** option in a CYCLE statement produces a plot of smoothed estimate of that cycle.
- Residual plots for model diagnostics can be obtained by using the **PLOT=** option in the ESTIMATE statement.
- Plots of series forecasts and model decompositions can be obtained by using the **PLOT=** option in the FORECAST statement.

The following example is a simple illustration of the available plot options.

Analysis of Sunspot Data: Illustration of ODS Graphics

In this example a well-known series, Wolfer's sunspot data (Anderson 1971), is considered. The data consist of yearly sunspot numbers recorded from 1749 to 1924. These sunspot numbers are known to have a cyclical pattern with a period of about eleven years. The following DATA step creates the input data set:

```
data sunspot;
    input year wolfer @@;
    year = mdy(1,1, year);
    format year year4.;
datalines;
1749  809 1750  834 1751  477 1752  478 1753  307 1754  122 1755   96
1756  102 1757  324 1758  476 1759  540 1760  629 1761  859 1762  612

... more lines ...

/* Load data into mylib */
data mylib.sunspot;
    set sunspot;
run;
```

The following statements specify a UCM that includes a cycle component and a random walk trend component:

```
proc ucm data=mylib.sunspot;
    id year interval=year;
    model wolfer;
    irregular;
    level ;
    cycle plot=(filter smooth);
    estimate back=24 plot=(loess panel cusum wn);
    forecast back=24 lead=24 plot=(forecasts decomp);
run;
```

The following subsections explain the graphics produced by the preceding statements.

Component Plots

The plots in Figure 30.8 and Figure 30.9, produced by specifying PLOT=(FILTER SMOOTH) in the CYCLE statement, show the filtered and smoothed estimates, respectively, of the cycle component in the model. Note that the smoothed estimate appears smoother than the filtered estimate. This is always true because the filtered estimate of a component at time t is based on the observations prior to time t —that is, it uses measurements from the first observation up to the $(t - 1)$ th observation. On the other hand, the corresponding smoothed estimate uses all the available observations—that is, all the measurements from the first observation to the last. This makes the smoothed estimate of the component more precise than the filtered estimate for the time points within historical period. In the forecast horizon, both filtered and smoothed estimates are identical, being based on the same set of observations.

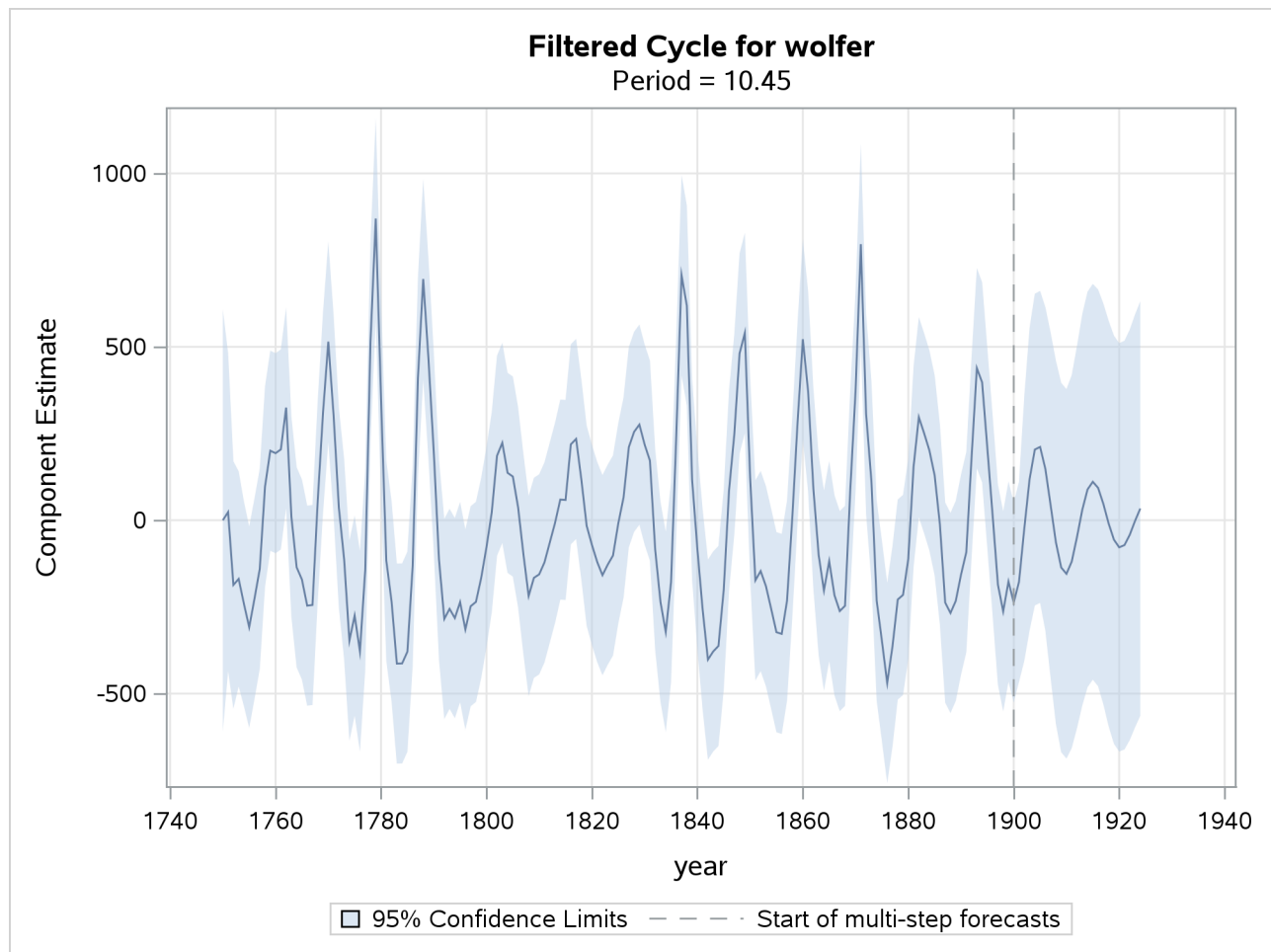
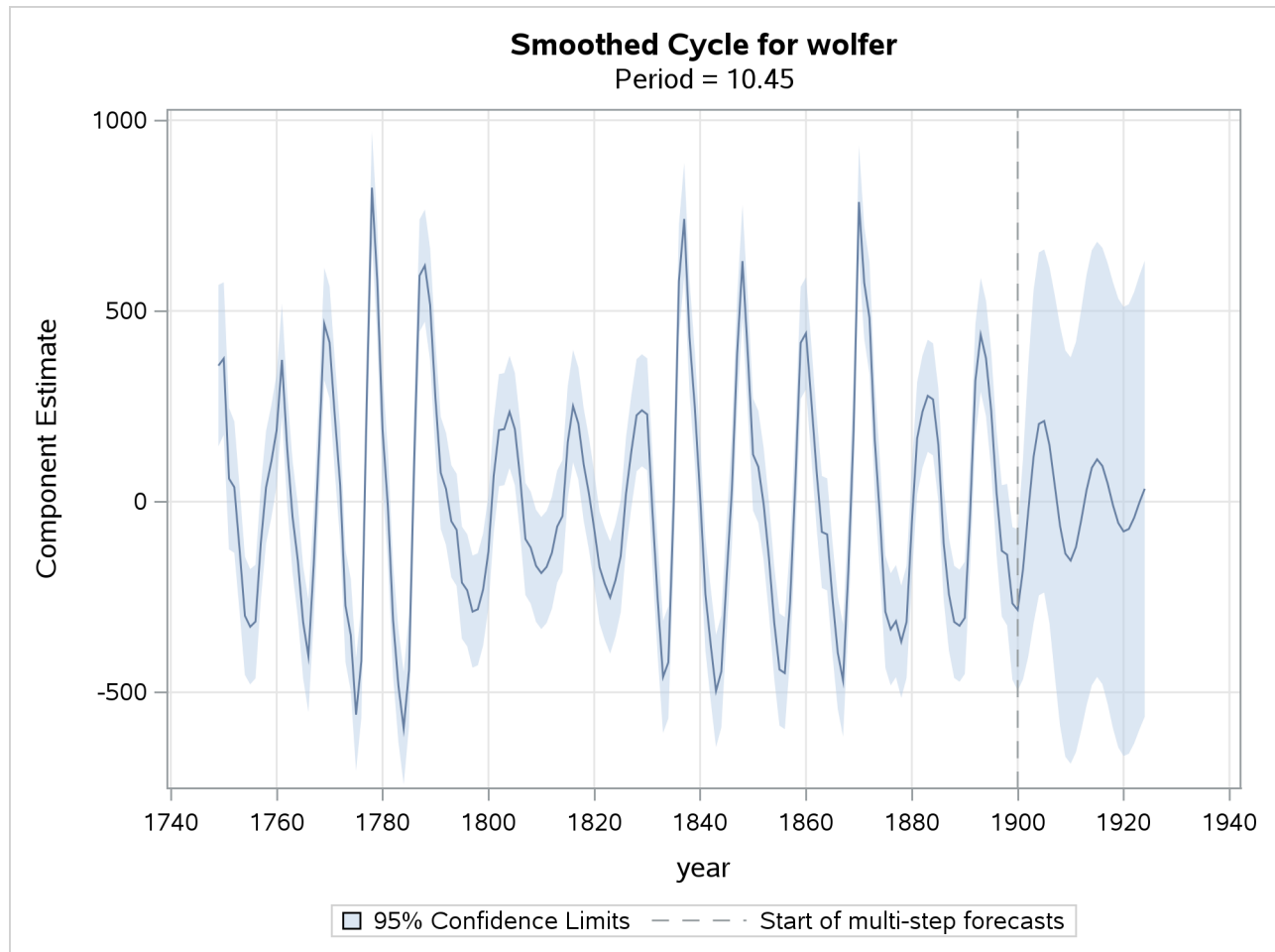
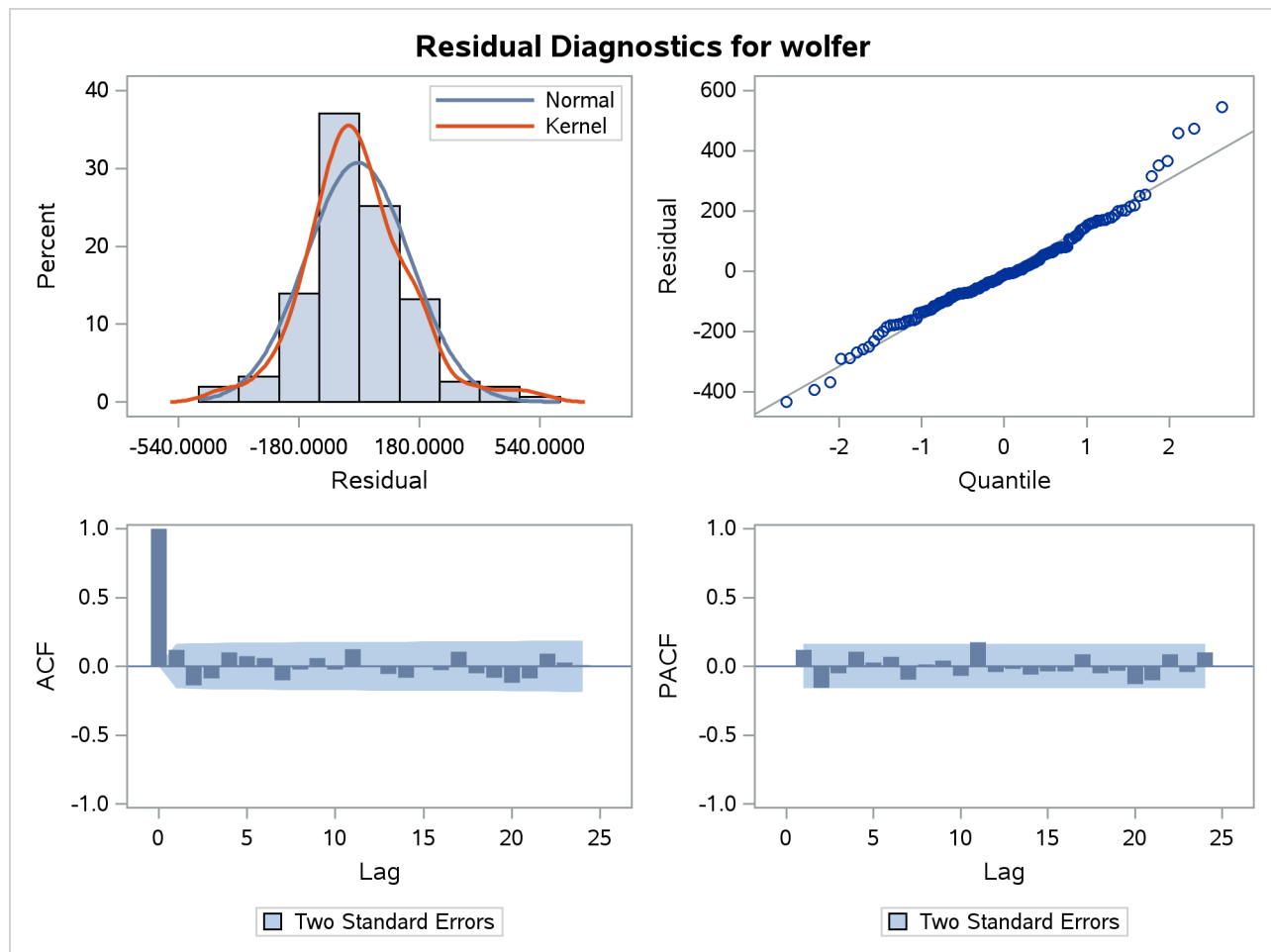
Figure 30.8 Sunspots Series: Filtered Cycle

Figure 30.9 Sunspots Series: Smoothed Cycle**Residual Diagnostics**

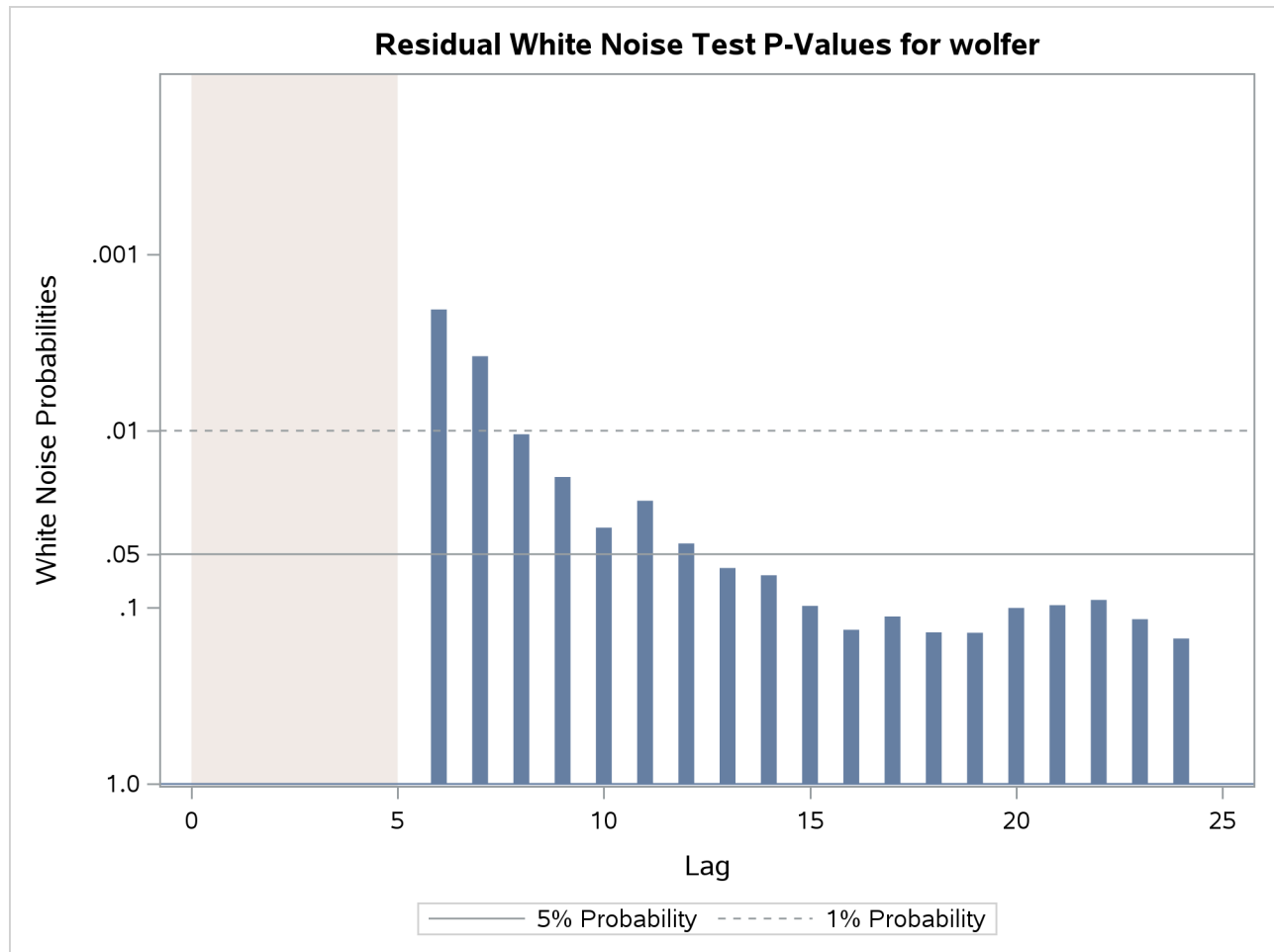
If the fitted model is appropriate for the given data, then the corresponding one-step-ahead residuals should be approximately *white*—that is, uncorrelated—and approximately normal. Moreover, the residuals should not display any discernible pattern. You can detect departures from these conditions graphically. Different residual diagnostic plots can be requested by using the `PLOT=` option in the `ESTIMATE` statement.

The normality can be checked by examining the histogram and the normal quantile plot of residuals. The whiteness can be checked by examining the ACF and PACF plots that show the sample autocorrelation and sample partial-autocorrelation at different lags. The diagnostic panel shown in [Figure 30.10](#), produced by specifying `PLOT=PANEL`, contains these four plots.

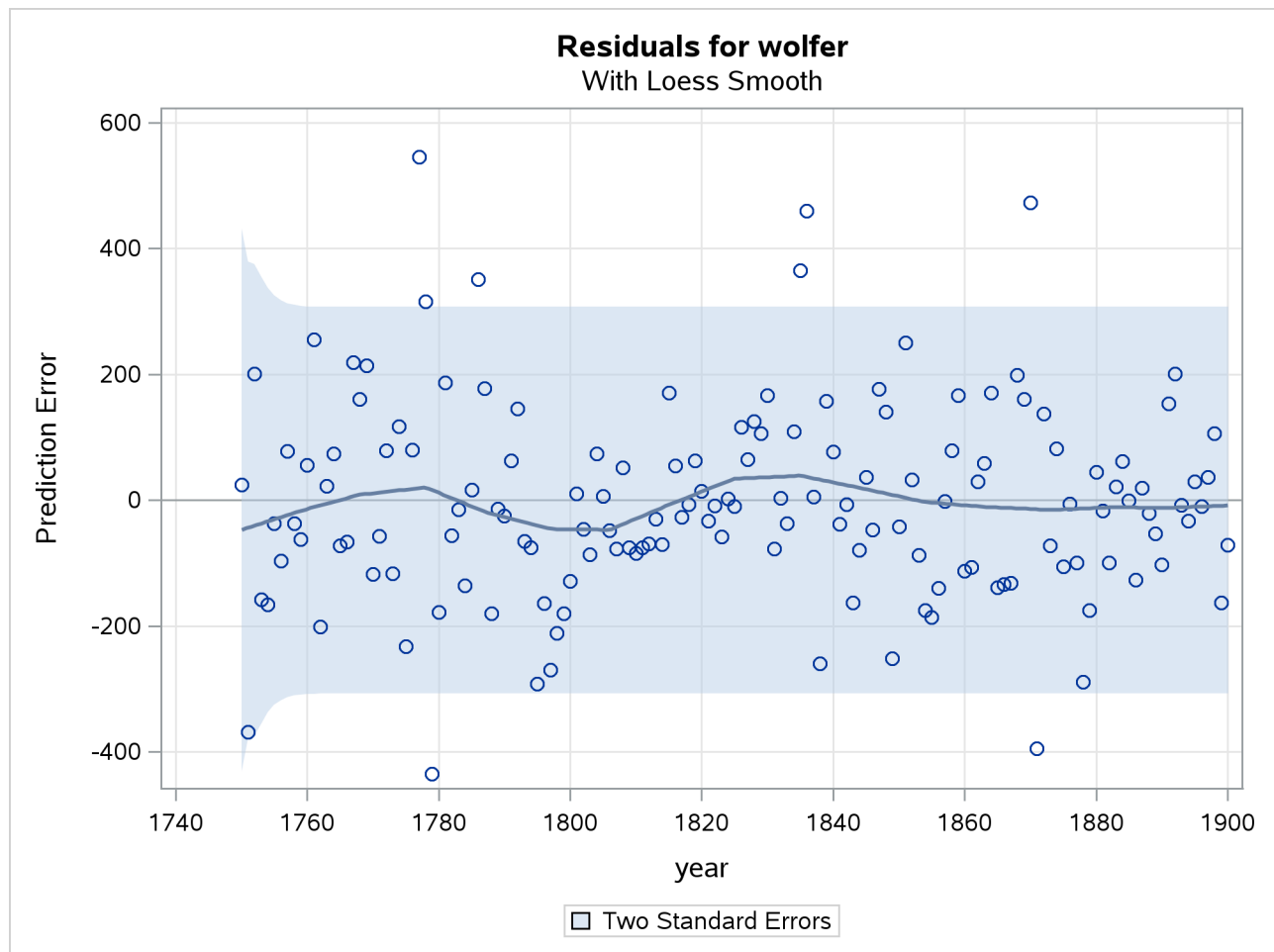
Figure 30.10 Sunspots Series: Residual Diagnostics

The residual histogram and Q-Q plot show no serious violation of normality. The histogram appears reasonably symmetric and follows the overlaid normal density curve reasonably closely. Similarly, in the Q-Q plot the residuals follow the reference line fairly closely. The ACF and PACF plots also do not exhibit any violation of the whiteness assumption; the correlations at all nonzero lags seem to be insignificant.

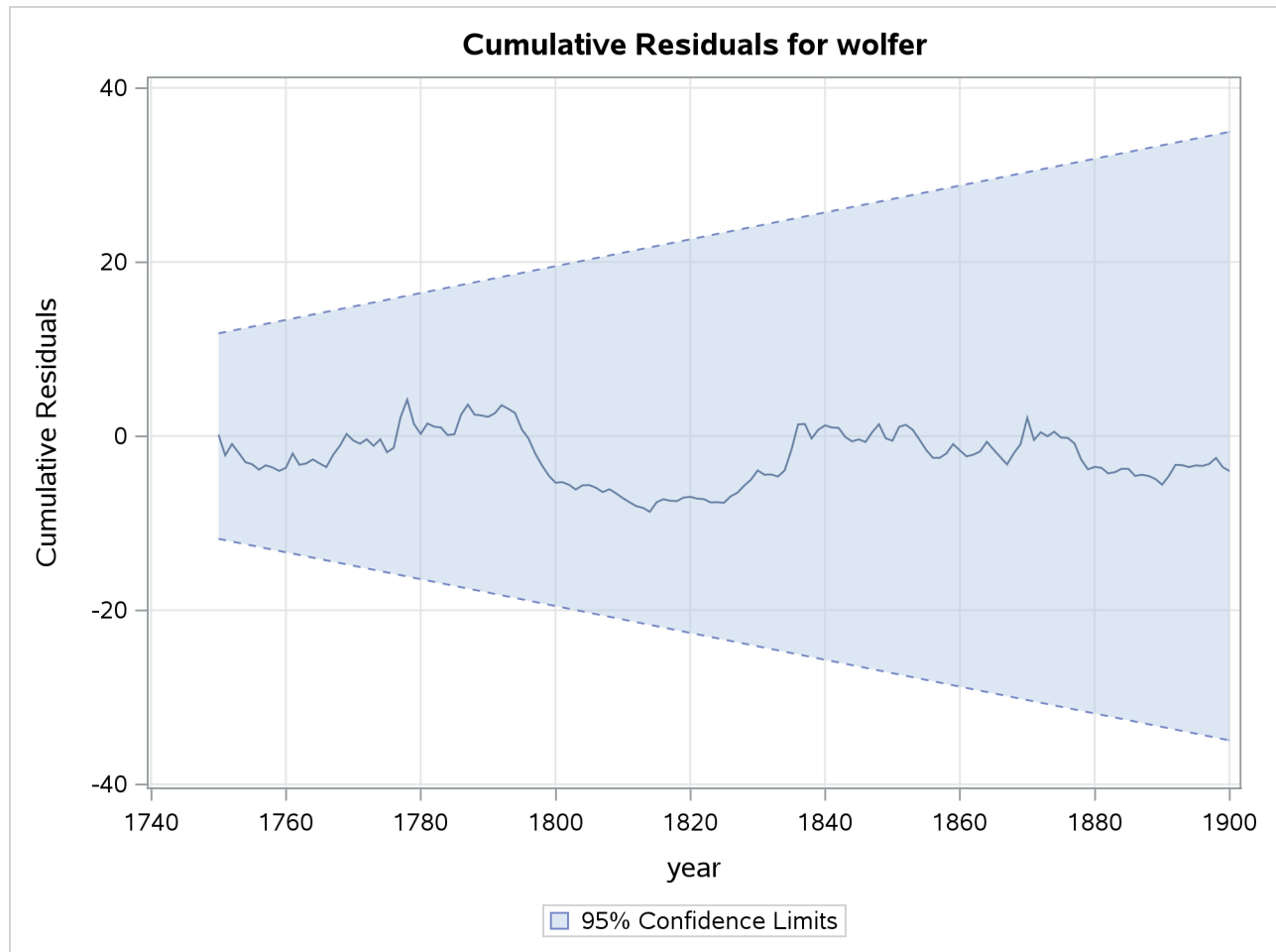
The residual whiteness can also be formally tested by using the Ljung-Box portmanteau test. The plot in [Figure 30.11](#), produced by specifying `PLOT=WN`, shows the p -values of the Ljung-Box test statistics at different lags. In these plots the p -values for the first few lags, equal to the number of estimated parameters in the model, are not shown because they are always missing. This portion of the plot is shaded blue to indicate this fact. In the case of this model, five parameters are estimated so the p -values for the first five lags are not shown. The p -values are displayed on a log scale in such a way that higher bars imply more extreme test statistics. In this plot some early p -values appear extreme. However, these p -values are based on large sample theory, which suggests that these statistics should be examined for lags larger than the square root of sample size. In this example it means that the p -values for the first $\sqrt{154} \approx 12$ lags can be ignored. With this consideration, the plot shows no violation of whiteness since the p -values after the 12th lag do not appear extreme.

Figure 30.11 Sunspots Series: Ljung-Box Portmanteau Test

The plot in [Figure 30.12](#), produced by specifying PLOT=LOESS, shows the residuals plotted against time with an overlaid loess curve. This plot is useful for checking whether any discernible pattern remains in the residuals. Here again, no significant pattern appears to be present.

Figure 30.12 Sunspots Series: Residual Loess Plot

The plot in [Figure 30.13](#), produced by specifying `PLOT=CUSUM`, shows the cumulative residuals plotted against time. This plot is useful for checking structural breaks. Here, there appears to be no evidence of structural break since the cumulative residuals remain within the confidence band throughout the sample period. Similarly, you can request a plot of the squared cumulative residuals by specifying `PLOT=CUSUMSQ`.

Figure 30.13 Sunspots Series: CUSUM Plot

Brockwell and Davis (1991) can be consulted for additional information on diagnosing residuals. For more information about CUSUM and CUSUMSQ plots, you can consult Harvey (1989).

Forecast and Series Decomposition Plots

You can use the PLOT= option in the FORECAST statement to obtain the series forecast plot and the series decomposition plots. The series decomposition plots show the result of successively adding different components in the model starting with the trend component. The IRREGULAR component is left out of this process. The following two plots, produced by specifying PLOT=DECOMP, show the results of successive component addition for this example. The first plot, shown in Figure 30.14, shows the smoothed trend component and the second plot, shown in Figure 30.15, shows the sum of smoothed trend and cycle.

Figure 30.14 Sunspots Series: Smoothed Trend

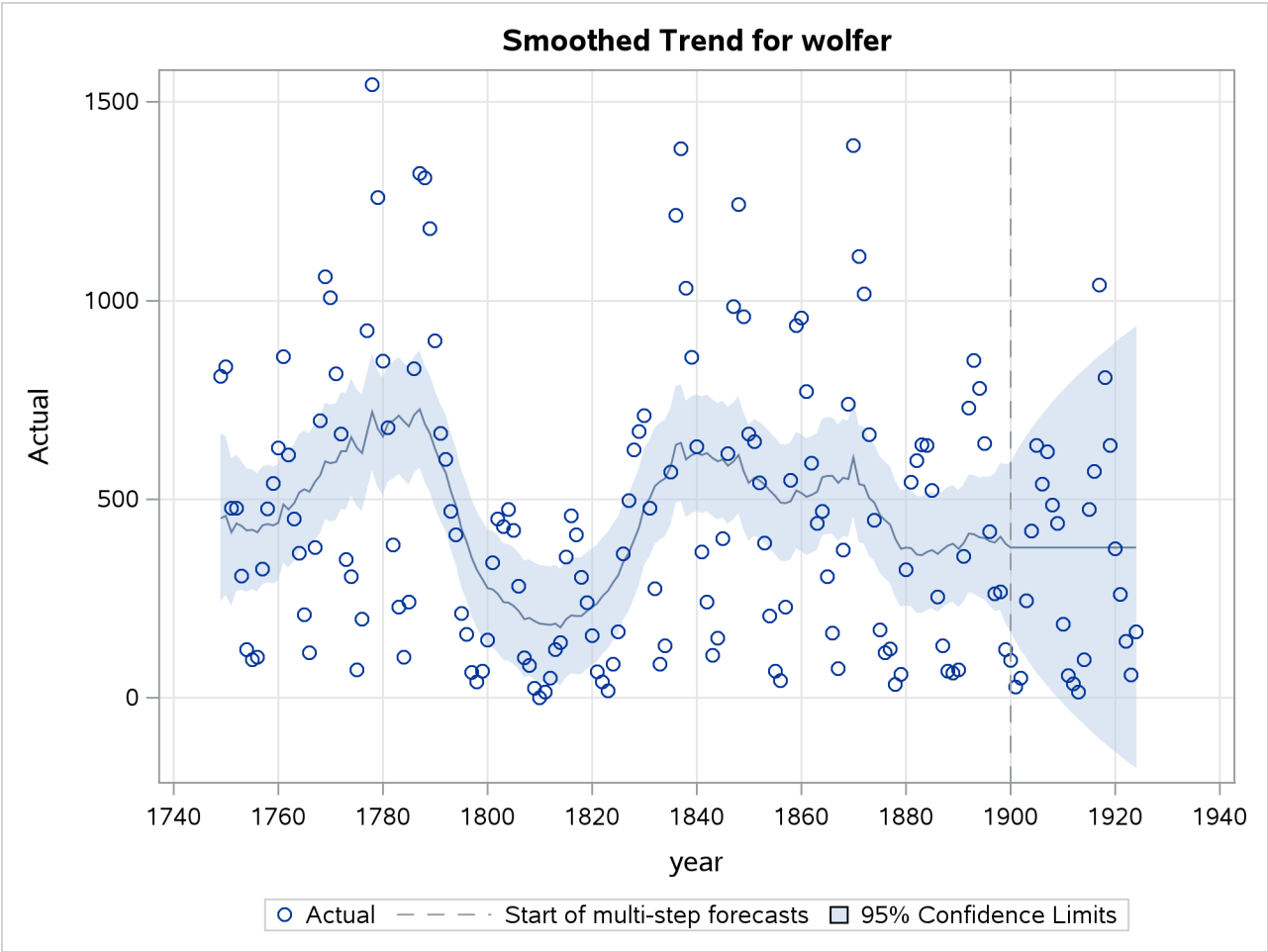
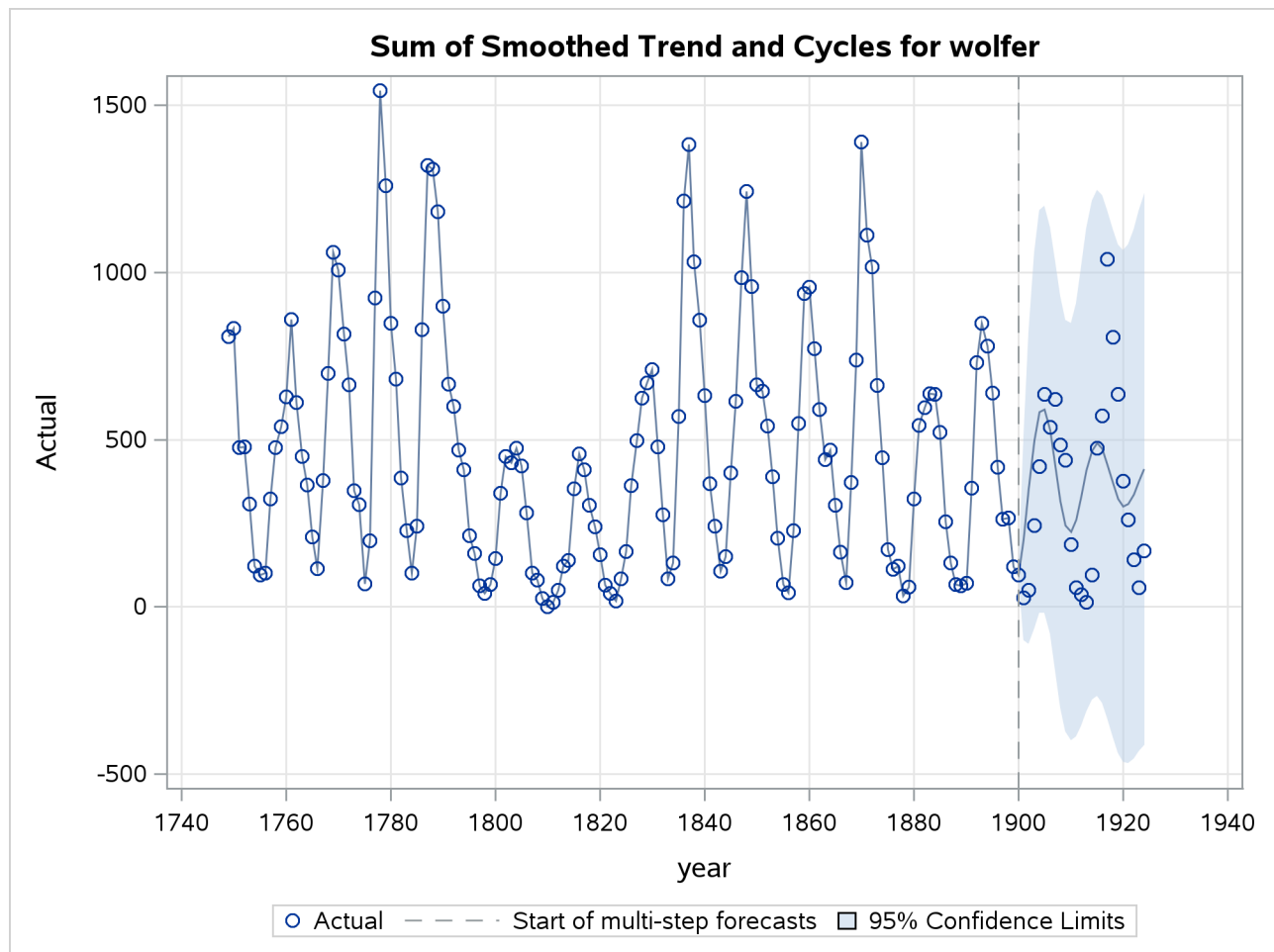
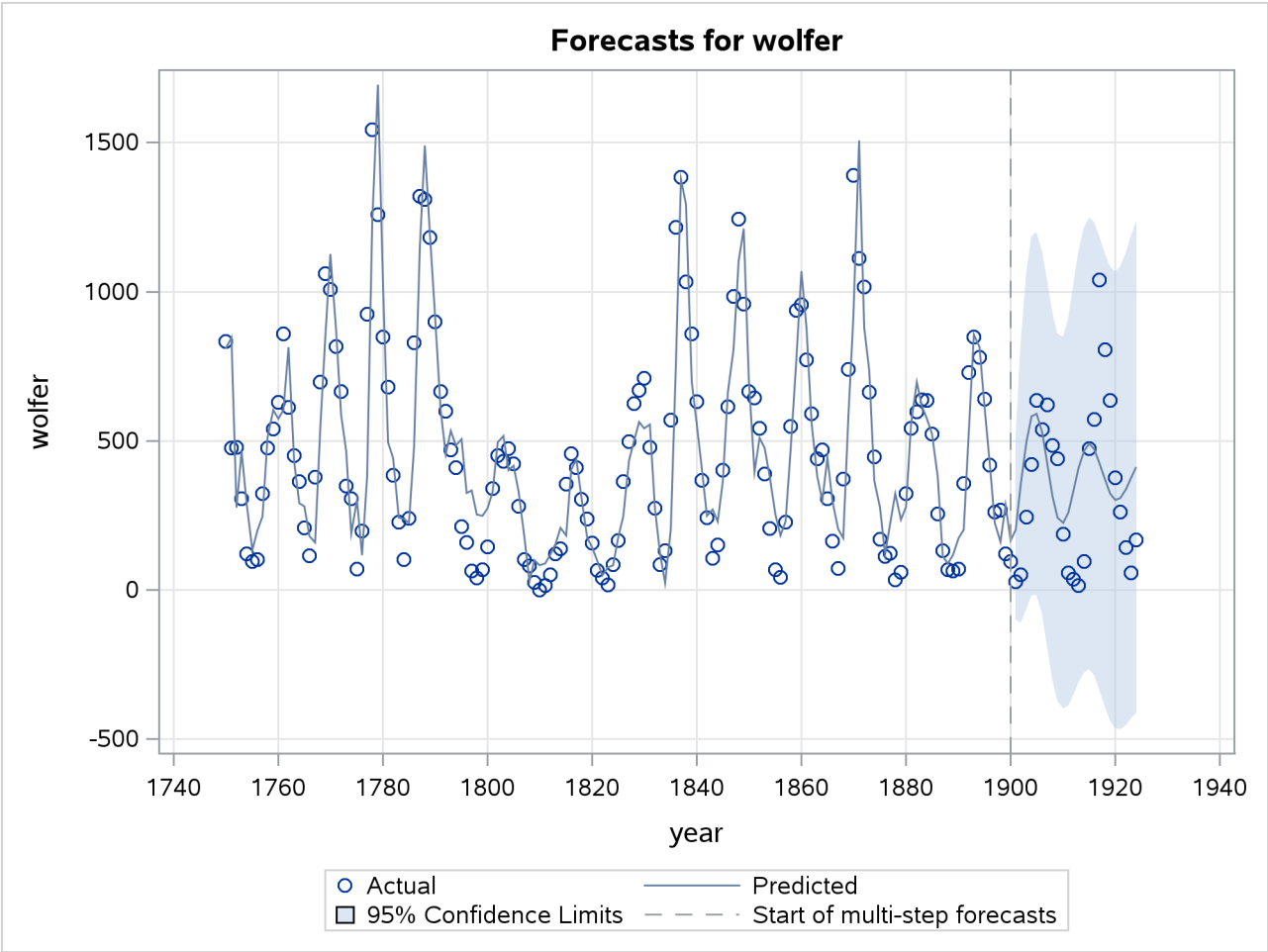


Figure 30.15 Sunspots Series: Smoothed Trend plus Cycle

Finally, Figure 30.16 shows the forecast plot.

Figure 30.16 Sunspots Series: Series Forecasts



ODS Table Names

The UCM procedure assigns a name to each table it creates. You can use these names to reference the table when using the Output Delivery System (ODS) to select tables and create output data sets. These names are listed in [Table 30.2](#).

Table 30.2 ODS Tables Produced by PROC UCM

ODS Table Name	Description	Statement	Option
Tables Summarizing the Estimation and Forecast Spans			
EstimationSpan	Estimation span summary information		Default
ForecastSpan	Forecast span summary information		Default

Table 30.2 *continued*

ODS Table Name	Description	Statement	Option
Tables Related to Model Parameters			
ConvergenceStatus	Convergence status of the estimation process		Default
FixedParameters	Fixed parameters in the model		Default
ParameterEstimates	Final estimates of the free parameters		Default
Tables Related to Model Information and Diagnostics			
BlockSeasonDescription	Information about the block seasonals in the model		Default
ComponentSignificance	Significance analysis of the components in the model		Default
CycleDescription	Information about the cycles in the model		Default
FitStatistics	Fit statistics based on the one-step-ahead predictions		Default
FitSummary	Likelihood-based fit statistics		Default
OutlierSummary	Summary table of the detected outliers		Default
AdditiveOutliers	AO statistics computed at each time point in the estimation span	OUTLIER	PRINT=DETAIL
LevelShifts	LS statistics computed at each time point in the estimation span	OUTLIER	PRINT=DETAIL
SeasonDescription	Information about the seasonals in the model		Default
SeasonHarmonics	Summary of harmonics in a trigonometric seasonal component	SEASON	PRINT=HARMONICS
SplineSeasonDescription	Information about the spline-seasonals in the model		Default
TrendInformation	Summary information of the level and slope components		Default
Tables Related to Filtered Component Estimates			
FilteredAutoReg	Filtered estimate of an autoreg component	AUTOREG	PRINT=FILTER
FilteredBlockSeason	Filtered estimate of a block seasonal component	BLOCKSEASON	PRINT=FILTER
FilteredCycle	Filtered estimate of a cycle component	CYCLE	PRINT=FILTER

Table 30.2 *continued*

ODS Table Name	Description	Statement	Option
FilteredIrregular	Filtered estimate of the irregular component	IRREGULAR	PRINT=FILTER
FilteredLevel	Filtered estimate of the level component	LEVEL	PRINT=FILTER
FilteredRandomReg	Filtered estimate of the time-varying random-regression coefficient	RANDOMREG	PRINT=FILTER
FilteredSeason	Filtered estimate of a seasonal component	SEASON	PRINT=FILTER
FilteredSlope	Filtered estimate of the slope component	SLOPE	PRINT=FILTER
FilteredSplineReg	Filtered estimate of the time-varying spline-regression coefficient	SPLINEREG	PRINT=FILTER
FilteredSplineSeason	Filtered estimate of a spline-seasonal component	SPLINESEASON	PRINT=FILTER
Tables Related to Smoothed Component Estimates			
SmoothedAutoReg	Smoothed estimate of an autoreg component	AUTOREG	PRINT=SMOOTH
SmoothedBlockSeason	Smoothed estimate of a block seasonal component	BLOCKSEASON	PRINT=SMOOTH
SmoothedCycle	Smoothed estimate of the cycle component	CYCLE	PRINT=SMOOTH
SmoothedIrregular	Smoothed estimate of the irregular component	IRREGULAR	PRINT=SMOOTH
SmoothedLevel	Smoothed estimate of the level component	LEVEL	PRINT=SMOOTH
SmoothedRandomReg	Smoothed estimate of the time-varying random-regression coefficient	RANDOMREG	PRINT=SMOOTH
SmoothedSeason	Smoothed estimate of a seasonal component	SEASON	PRINT=SMOOTH
SmoothedSlope	Smoothed estimate of the slope component	SLOPE	PRINT=SMOOTH
SmoothedSplineReg	Smoothed estimate of the time-varying spline-regression coefficient	SPLINEREG	PRINT=SMOOTH
SmoothedSplineSeason	Smoothed estimate of a spline-seasonal component	SPLINESEASON	PRINT=SMOOTH

Table 30.2 *continued*

ODS Table Name	Description	Statement	Option
Tables Related to Series Decomposition and Forecasting			
FilteredAllExceptIrreg	Filtered estimate of sum of all components except the irregular component	FORECAST	PRINT=FDECOMP
FilteredTrend	Filtered estimate of trend	FORECAST	PRINT= FDECOMP
FilteredTrendReg	Filtered estimate of trend plus regression	FORECAST	PRINT=FDECOMP
FilteredTrendRegCyc	Filtered estimate of trend plus regression plus cycles and autoreg	FORECAST	PRINT=FDECOMP
Forecasts	Dependent series forecasts		Default
PostSamplePrediction	Forecasting performance in the holdout period	FORECAST	BACK=
SmoothedAllExceptIrreg	Smoothed estimate of sum of all components except the irregular component	FORECAST	PRINT=DECOMP
SmoothedTrend	Smoothed estimate of trend	FORECAST	PRINT= DECOMP
SmoothedTrendReg	Smoothed estimate of trend plus regression	FORECAST	PRINT=DECOMP
SmoothedTrendRegCyc	Smoothed estimate of trend plus regression plus cycles and autoreg	FORECAST	PRINT=DECOMP

NOTE: The tables are related to a single series within a BY group. In the case of models that contain multiple cycles, seasonal components, or block seasonal components, the corresponding component estimate tables are sequentially numbered. For example, if a model contains two cycles and a seasonal component and the PRINT=SMOOTH option is used for each of them, the ODS tables containing the smoothed estimates will be named SmoothedCycle1, SmoothedCycle2, and SmoothedSeason. Note that the seasonal table is not numbered because there is only one seasonal component. There are some exceptions to this numbering rule: the tables, FilteredRandomReg, SmoothedRandomReg, FilteredSplineReg, and SmoothedSplineReg, are always numbered starting with 0.

ODS Graph Names

Statistical procedures use ODS Graphics to create graphs as part of their output. ODS Graphics is described in detail in Chapter 24, “Statistical Graphics Using ODS” (*SAS/STAT User’s Guide*).

You can reference every graph produced through ODS Graphics with a name. The names of the graphs that PROC UCM generates are listed in Table 30.3, along with the required statements and options.

Table 30.3 ODS Graphics Produced by PROC UCM

ODS Graph Name	Description	Statement	Option
Plots Related to Residual Analysis			
ErrorACFPlot	Prediction error autocorrelation plot	ESTIMATE	PLOT=ACF
ErrorPACFPlot	Prediction error partial-autocorrelation plot	ESTIMATE	PLOT=PACF
ErrorHistogram	Prediction error histogram	ESTIMATE	PLOT=NORMAL
ErrorQQPlot	Prediction error normal quantile plot	ESTIMATE	PLOT=QQ
ErrorPlot	Plot of prediction errors	ESTIMATE	PLOT=RESIDUAL
ErrorWhiteNoiseLogProbPlot	Plot of p -values at different lags for the Ljung-Box portmanteau white-noise test statistics	ESTIMATE	PLOT=WN
CUSUMPlot	Plot of cumulative residuals	ESTIMATE	PLOT=CUSUM
CUSUMSQPlot	Plot of cumulative squared residuals	ESTIMATE	PLOT=CUSUMSQ
ModelPlot	Plot of one-step-ahead forecasts in the estimation span	ESTIMATE	PLOT=MODEL
PanelResidualPlot	Panel of residual diagnostic plots	ESTIMATE	PLOT=PANEL
ResidualLoessPlot	Time series plot of residuals with superimposed loess smoother	ESTIMATE	PLOT=LOESS
Plots Related to Filtered Component Estimates			
FilteredAutoregPlot	Plot of filtered autoreg component	AUTOREG	PLOT=FILTER
FilteredBlockSeasonPlot	Plot of filtered block season component	BLOCKSEASON	PLOT=FILTER
FilteredCyclePlot	Plot of filtered cycle component	CYCLE	PLOT=FILTER
FilteredIrregularPlot	Plot of filtered irregular component	IRREGULAR	PLOT=FILTER
FilteredLevelPlot	Plot of filtered level component	LEVEL	PLOT=FILTER

Table 30.3 *continued*

ODS Graph Name	Description	Statement	Option
FilteredRandomRegPlot	Plot of filtered time-varying regression coefficient	RANDOMREG	PLOT=FILTER
FilteredSeasonPlot	Plot of filtered season component	SEASON	PLOT=FILTER
FilteredSlopePlot	Plot of filtered slope component	SLOPE	PLOT=FILTER
FilteredSplineRegPlot	Plot of filtered time-varying regression coefficient	SPLINEREG	PLOT=FILTER
FilteredSplineSeasonPlot	Plot of filtered spline-season component	SPLINESEASON	PLOT=FILTER
AnnualSeasonPlot	Plot of annual variation in the filtered season component	SEASON	PLOT=F_ANNUAL
Plots Related to Smoothed Component Estimates			
SmoothedAutoregPlot	Plot of smoothed autoreg component	AUTOREG	PLOT=SMOOTH
SmoothedBlockSeasonPlot	Plot of smoothed block season component	BLOCKSEASON	PLOT=SMOOTH
SmoothedCyclePlot	Plot of smoothed cycle component	CYCLE	PLOT=SMOOTH
SmoothedIrregularPlot	Plot of smoothed irregular component	IRREGULAR	PLOT=SMOOTH
SmoothedLevelPlot	Plot of smoothed level component	LEVEL	PLOT=SMOOTH
SmoothedRandomRegPlot	Plot of smoothed time-varying regression coefficient	RANDOMREG	PLOT=SMOOTH
SmoothedSeasonPlot	Plot of smoothed season component	SEASON	PLOT=SMOOTH
SmoothedSlopePlot	Plot of smoothed slope component	SLOPE	PLOT=SMOOTH
SmoothedSplineRegPlot	Plot of smoothed time-varying regression coefficient	SPLINEREG	PLOT=SMOOTH
SmoothedSplineSeasonPlot	Plot of smoothed spline-season component	SPLINESEASON	PLOT=SMOOTH
AnnualSeasonPlot	Plot of annual variation in the smoothed season component	SEASON	PLOT=S_ANNUAL
Plots Related to Series Decomposition and Forecasting			
ForecastsOnlyPlot	Series forecasts beyond the historical period	FORECAST	DEFAULT

Table 30.3 *continued*

ODS Graph Name	Description	Statement	Option
ForecastsPlot	One-step-ahead as well as multistep-ahead forecasts	FORECAST	PLOT=FORECASTS
FilteredAllExceptIrregPlot	Plot of sum of all filtered components except the irregular component	FORECAST	PLOT= FDECOMP
FilteredTrendPlot	Plot of filtered trend	FORECAST	PLOT= FDECOMP
FilteredTrendRegCycPlot	Plot of sum of filtered trend, cycles, and regression effects	FORECAST	PLOT= FDECOMP
FilteredTrendRegPlot	Plot of filtered trend plus regression effects	FORECAST	PLOT= FDECOMP
SmoothedAllExceptIrregPlot	Plot of sum of all smoothed components except the irregular component	FORECAST	PLOT= DECOMP
SmoothedTrendPlot	Plot of smoothed trend	FORECAST	PLOT= TREND
SmoothedTrendRegPlot	Plot of smoothed trend plus regression effects	FORECAST	PLOT= DECOMP
SmoothedTrendRegCycPlot	Plot of sum of smoothed trend, cycles, and regression effects	FORECAST	PLOT= DECOMP
FilteredAllExceptIrregVarPlot	Plot of standard error of sum of all filtered components except the irregular	FORECAST	PLOT= FDECOMPVAR
FilteredTrendVarPlot	Plot of standard error of filtered trend	FORECAST	PLOT= FDECOMPVAR
FilteredTrendRegVarPlot	Plot of standard error of filtered trend plus regression effects	FORECAST	PLOT= FDECOMPVAR
FilteredTrendRegCycVarPlot	Plot of standard error of filtered trend, cycles, and regression effects	FORECAST	PLOT= FDECOMPVAR
SmoothedAllExceptIrregVarPlot	Plot of standard error of sum of all smoothed components except the irregular	FORECAST	PLOT= DECOMPVAR
SmoothedTrendVarPlot	Plot of standard error of smoothed trend	FORECAST	PLOT= DECOMPVAR
SmoothedTrendRegVarPlot	Plot of standard error of smoothed trend plus regression effects	FORECAST	PLOT= DECOMPVAR
SmoothedTrendRegCycVarPlot	Plot of standard error of smoothed trend, cycles, and regression effects	FORECAST	PLOT= DECOMPVAR

OUTFOR= Data Set

You can use the OUTFOR= option in the FORECAST statement to store the series and component forecasts produced by the procedure. This data set contains the following columns:

- the BY variables
- the ID variable. If an ID variable is not specified, then a numerical variable, `_ID_`, is created that contains the observation numbers from the input data set.
- the dependent series and the predictor series
- FORECAST, a numerical variable containing the one-step-ahead predicted values and the multistep forecasts
- RESIDUAL, a numerical variable containing the difference between the actual and forecast values
- STD, a numerical variable containing the standard error of prediction
- LCL and UCL, numerical variables containing the lower and upper forecast confidence limits
- S_SERIES and VS_SERIES, numerical variables containing the smoothed values of the dependent series and their variances
- S_IRREG and VS_IRREG, numerical variables containing the smoothed values of the irregular component and their variances. These variables are present only if the model has an irregular component.
- F_LEVEL, VF_LEVEL, S_LEVEL, and VS_LEVEL, numerical variables containing the filtered and smoothed values of the level component and the respective variances. These variables are present only if the model has a level component.
- F_SLOPE, VF_SLOPE, S_SLOPE, and VS_SLOPE, numerical variables containing the filtered and smoothed values of the slope component and the respective variances. These variables are present only if the model has a slope component.
- F_AUTOREG, VF_AUTOREG, S_AUTOREG, and VS_AUTOREG, numerical variables containing the filtered and smoothed values of the autoreg component and the respective variances. These variables are present only if the model has an autoreg component.
- F_CYCLE, VF_CYCLE, S_CYCLE, and VS_CYCLE, numerical variables containing the filtered and smoothed values of the cycle component and the respective variances. If there are multiple cycles in the model, these variables are sequentially numbered as `F_CYCLE1`, `F_CYCLE2`, and so on. These variables are present only if the model has at least one cycle component.
- F_SEASON, VF_SEASON, S_SEASON, and VS_SEASON, numerical variables containing the filtered and smoothed values of the season component and the respective variances. If there are multiple seasons in the model, these variables are sequentially numbered as `F_SEASON1`, `F_SEASON2`, and so on. These variables are present only if the model has at least one season component.
- F_BLKSEAS, VF_BLKSEAS, S_BLKSEAS, and VS_BLKSEAS, numerical variables containing the filtered and smoothed values of the blockseason component and the respective variances. If there are multiple block seasons in the model, these variables are sequentially numbered as `F_BLKSEAS1`, `F_BLKSEAS2`, and so on.

- F_SPLSEAS, VF_SPLSEAS, S_SPLSEAS, and VS_SPLSEAS, numerical variables containing the filtered and smoothed values of the splinesseason component and the respective variances. If there are multiple spline seasons in the model, these variables are sequentially numbered as F_SPLSEAS1, F_SPLSEAS2, and so on. These variables are present only if the model has at least one splinesseason component.
- Filtered and smoothed estimates, and their variances, of the time-varying regression coefficients of the variables that are specified in the RANDOMREG and SPLINEREG statements. A variable is not included if its coefficient is time-invariant, that is, if the associated disturbance variance is 0.
- F_TF, VF_TF, S_TF, and VS_TF, numerical variables that contain the filtered and smoothed values of the transfer-function component and their variances. If there are multiple transfer-function components in the model, these variables are sequentially numbered as F_TF1, F_TF2, and so on. These variables are present only if the model has at least one transfer-function component.
- S_TREG and VS_TREG, numerical variables containing the smoothed values of level plus regression component and their variances. These variables are present only if the model has at least one predictor variable or has dependent lags.
- S_TREGCYC and VS_TREGCYC, numerical variables containing the smoothed values of level plus regression plus cycle component and their variances. These variables are present only if the model has at least one cycle or an autoreg component.
- S_NOIRREG and VS_NOIRREG, numerical variables containing the smoothed values of the sum of all components except the irregular component and their variances. These variables are present only if the model has at least one seasonal or block seasonal component.

OUTEST= Data Set

You can use the OUTEST= option in the ESTIMATE statement to store the model parameters and the related estimation details. This data set contains the following columns:

- the BY variables
- COMPONENT, a character variable containing the name of the component corresponding to the parameter being described
- PARAMETER, a character variable containing the parameter name
- TYPE, a character variable indicating whether the parameter value was fixed by the user or estimated
- _STATUS_, a character variable indicating whether the parameter estimation process converged or failed or there was an error of some other kind
- ESTIMATE, a numerical variable containing the parameter estimate
- STD, a numerical variable containing the standard error of the parameter estimate. This has a missing value if the parameter value is fixed.

- TVALUE, a numerical variable containing the t -statistic. This has a missing value if the parameter value is fixed.
- PVALUE, a numerical variable containing the p -value. This has a missing value if the parameter value is fixed.

Statistics of Fit

This section explains the goodness-of-fit statistics reported to measure how well the specified model fits the data.

First the various statistics of fit that are computed using the prediction errors, $y_t - \hat{y}_t$, are considered. In these formulas, n is the number of nonmissing prediction errors and k is the number of fitted parameters in the model. Moreover, the sum of squared errors, $SSE = \sum (y_t - \hat{y}_t)^2$, and the total sum of squares for the series corrected for the mean, $SST = \sum (y_t - \bar{y})^2$, where $\bar{y}$ is the series mean, and the sums are over all the nonmissing prediction errors.

Mean Square Error

The mean square prediction error, $MSE = \frac{1}{n}SSE$

Root Mean Square Error

The root mean square error, $RMSE = \sqrt{MSE}$

Mean Absolute Percentage Error

The mean absolute percentage prediction error, $MAPE = \frac{100}{n} \sum_{t=1}^n |(y_t - \hat{y}_t)/y_t|$.

The summation ignores observations where $y_t = 0$.

R-Square

The R-square statistic, $R^2 = 1 - SSE/SST$.

If the model fits the series badly, the model error sum of squares, SSE, might be larger than SST and the R-square statistic will be negative.

Adjusted R-Square

The adjusted R-square statistic, $1 - (\frac{n-1}{n-k})(1 - R^2)$

Amemiya's Adjusted R-Square

Amemiya's adjusted R-square, $1 - (\frac{n+k}{n-k})(1 - R^2)$

Random Walk R-Square

The random walk R-square statistic (Harvey's R-square statistic that uses the random walk model for comparison), $1 - (\frac{n-1}{n})SSE/RWSSE$, where $RWSSE = \sum_{t=2}^n (y_t - y_{t-1} - \mu)^2$, and $\mu = \frac{1}{n-1} \sum_{t=2}^n (y_t - y_{t-1})$

Maximum Percentage Error

The largest percentage prediction error, $100 \max((y_t - \hat{y}_t)/y_t)$. In this computation the observations where $y_t = 0$ are ignored.

The likelihood-based fit statistics are reported separately (see the section “UCMs as State Space Models” on page 2162). They include the full log likelihood (L_∞), the diffuse part of the log likelihood, the normalized residual sum of squares, and several information criteria: AIC, AICC, HQIC, BIC, and CAIC. Let q denote the number of estimated parameters, n be the number of nonmissing measurements in the estimation span, and d

be the number of diffuse elements in the initial state vector that are successfully initialized during the Kalman filtering process. Moreover, let $n^* = (n - d)$. The reported information criteria, all in smaller-is-better form, are described in Table 30.4.

Table 30.4 Information Criteria

Criterion	Formula	Reference
AIC	$-2L_\infty + 2q$	Akaike (1974)
AICC	$-2L_\infty + 2qn^*/(n^* - q - 1)$	Hurvich and Tsai (1989)
HQIC	$-2L_\infty + 2q \log \log(n^*)$	Burnham and Anderson (1998)
BIC	$-2L_\infty + q \log(n^*)$	Hannan and Quinn (1979)
CAIC	$-2L_\infty + q(\log(n^*) + 1)$	Schwarz (1978)
		Bozdogan (1987)

Examples: UCM Procedure

Example 30.1: The Airline Series Revisited

The series in this example, the monthly airline passenger series, has already been discussed earlier; see the section “A Seasonal Series with Linear Trend” on page 2115. Recall that the series consists of monthly numbers of international airline travelers (from January 1949 to December 1960). Here additional output features of the UCM procedure are illustrated, such as how to use the ESTIMATE and FORECAST statements to limit the span of the data used in parameter estimation and forecasting. The following statements fit a basic structural model (BSM) to the logarithm of the airline passenger numbers. The disturbance variance for the slope component is held fixed at value 0; that is, the trend is locally linear with constant slope. In order to evaluate the performance of the fitted model on observed data, some of the observed data are withheld during parameter estimation and forecast computations. The observations in the last two years, 1959 and 1960, are not used in parameter estimation, while the observations in the last year, 1960, are not used in the forecasting computations. This is done using the BACK= option in the ESTIMATE and FORECAST statements. In addition, a panel of residual diagnostic plots is obtained using the PLOT= PANEL option in the ESTIMATE statement.

```
data seriesG;
  set sashelp.air;
  logair = log(air);
run;

/* Load data into mylib */
data mylib.seriesG;
  set seriesG;
run;
```

```

proc ucm data = mylib.seriesG;
  id date interval = month;
  model logair;
  irregular;
  level;
  slope var = 0 noest;
  season length = 12 type=trig;
  estimate back=24 plot=panel;
  forecast back=12 lead=24 print=forecasts;
run;

```

Output 30.1.1 and Output 30.1.2 display the summary of data used in estimation and forecasting, respectively. These tables provide simple summary statistics for the estimation and forecast spans; they include useful information such as the beginning and ending dates of the span, the number of nonmissing values, and so on.

Output 30.1.1 Observation Span Used in Parameter Estimation (Partial Output)

Variable	Type	First	Last	Nobs	Mean
logair	Dependent	JAN1949	DEC1958	120	5.43035

Output 30.1.2 Observation Span Used in Forecasting (Partial Output)

Variable	Type	First	Last	Nobs	Mean
logair	Dependent	JAN1949	DEC1959	132	5.48654

Output 30.1.3, and Output 30.1.4 display the fixed parameters in the model and the final estimates of the free parameters, respectively.

Output 30.1.3 Fixed Parameters in the Model

The UCM Procedure

Fixed Parameters in the Model		
Component	Parameter	Value
Slope	Error Variance	0

Output 30.1.4 Maximum Likelihood Estimates of the Free Parameters

The UCM Procedure

Final Estimates of the Free Parameters					
Component	Parameter	Estimate	Approx Std Error	t Value	Approx Pr > t
Irregular	Error Variance	0.00018686	0.0001	1.54	0.1233
Level	Error Variance	0.00040314	0.0002	2.57	0.0100
Season	Error Variance	0.00000350	0.0000	2.10	0.0354

Two types of goodness-of-fit statistics are reported after a model is fit to the series (see Output 30.1.5 and Output 30.1.6). The first type is the likelihood-based goodness-of-fit statistics, which include the full

likelihood of the data, the diffuse portion of the likelihood (see the section “[Details: UCM Procedure](#)” on page 2156), and the information criteria. The second type of statistics is based on the raw residuals: residual = observed – predicted. If the model is nonstationary, then one-step-ahead predictions are not available for some initial observations, and the number of values used in computing these fit statistics will be different from those used in computing the likelihood-based test statistics.

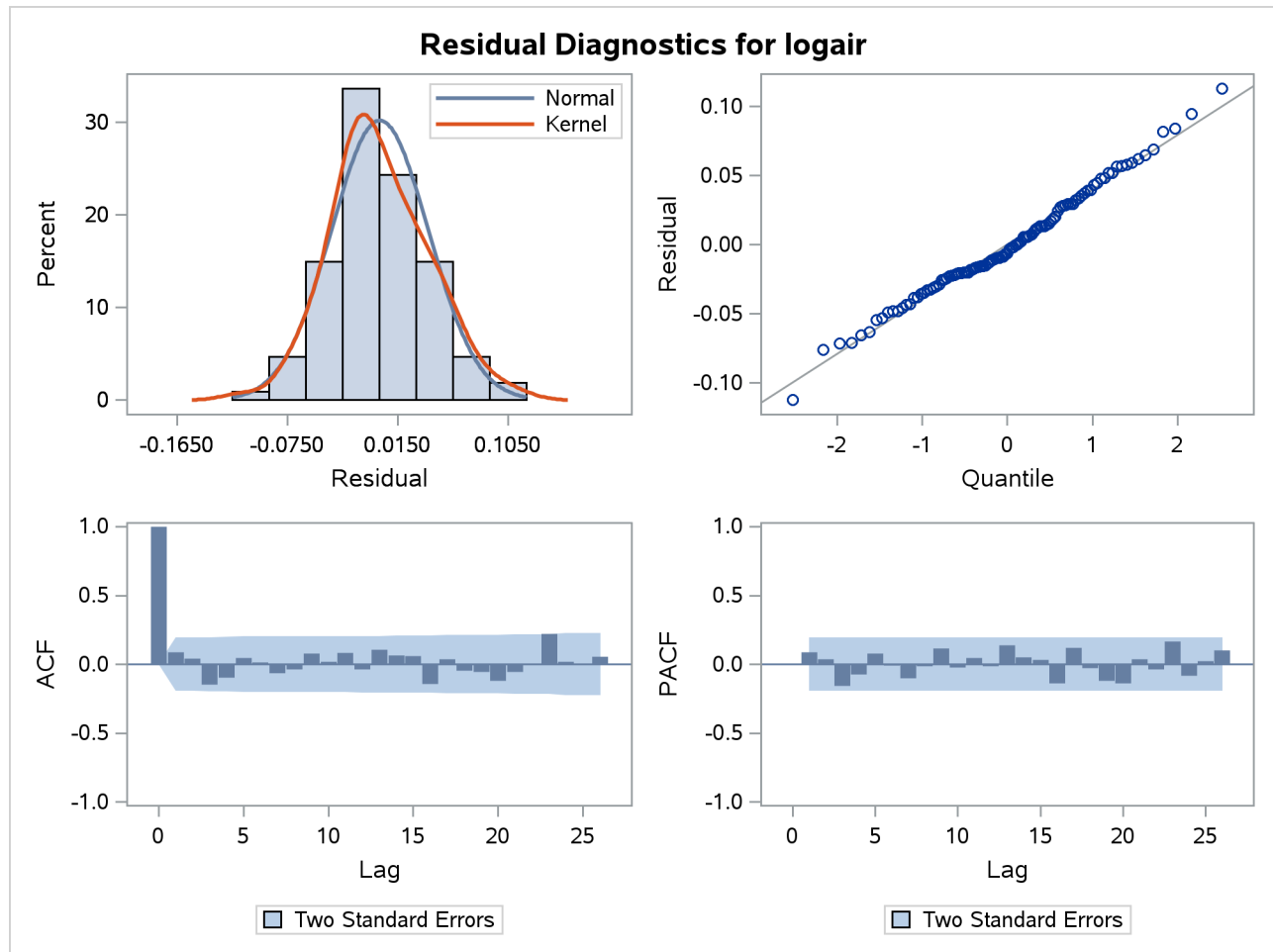
Output 30.1.5 Likelihood-Based Fit Statistics for the Airline Data

Likelihood Based Fit Statistics	
Statistic	Value
Diffuse Log Likelihood	180.63
Diffuse Part of Log Likelihood	-13.93
Non-Missing Observations Used	120
Estimated Parameters	3
Initialized Diffuse State Elements	13
Normalized Residual Sum of Squares	107
AIC (smaller is better)	-355.3
BIC (smaller is better)	-347.2
AICC (smaller is better)	-355
HQIC (smaller is better)	-352
CAIC (smaller is better)	-344.2

Output 30.1.6 Residuals-Based Fit Statistics for the Airline Data

Fit Statistics Based on Residuals	
Mean Squared Error	0.00156
Root Mean Squared Error	0.03944
Mean Absolute Percentage Error	0.57677
Maximum Percent Error	2.19396
R-Square	0.98705
Adjusted R-Square	0.98680
Random Walk R-Square	0.86370
Amemiya's Adjusted R-Square	0.98630
Number of non-missing residuals used for computing the fit statistics = 107	

The diagnostic plots based on the one-step-ahead residuals are shown in [Output 30.1.7](#). The residual histogram and the Q-Q plot show no reasons to question the approximate normality of the residual distribution. The remaining plots check for the *whiteness* of the residuals. The sample correlation plots, the autocorrelation function (ACF) and the partial autocorrelation function (PACF), also do not show any significant violations of the whiteness of the residuals. Therefore, on the whole, the model seems to fit the data well.

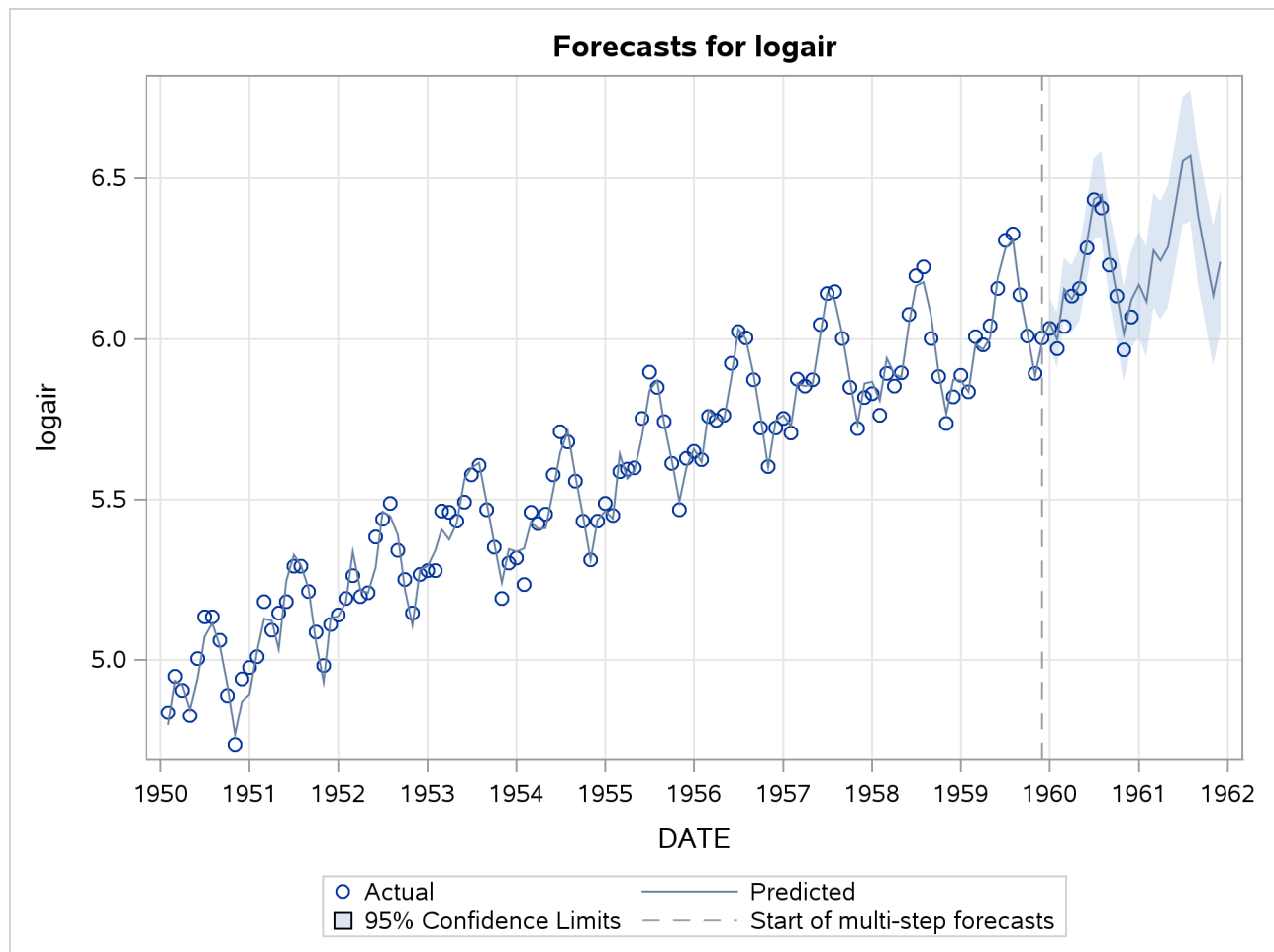
Output 30.1.7 Residual Diagnostics for the Airline Series Using a BSM

The forecasts are given in [Output 30.1.8](#). To save space, the upper and lower confidence limit columns are dropped from the output, and only the rows corresponding to the year 1960 are shown. Recall that the actual measurements in 1959 and 1960 were withheld during the parameter estimation, and the ones in 1960 were not used in the forecast computations.

Output 30.1.8 Forecasts for the Airline Data

date	Forecast	StdErr	logair	Residual
JAN60	6.050	0.038	6.033	-0.017
FEB60	5.996	0.044	5.969	-0.027
MAR60	6.156	0.049	6.038	-0.118
APR60	6.124	0.053	6.133	0.010
MAY60	6.168	0.058	6.157	-0.011
JUN60	6.303	0.061	6.282	-0.021
JUL60	6.435	0.065	6.433	-0.002
AUG60	6.450	0.068	6.407	-0.043
SEP60	6.265	0.071	6.230	-0.035
OCT60	6.138	0.073	6.133	-0.005
NOV60	6.015	0.075	5.966	-0.049
DEC60	6.121	0.077	6.068	-0.053

Output 30.1.9 shows the forecast plot. The forecasts in 1960 show that the model predictions were quite good.

Output 30.1.9 Forecast Plot of the Airline Series Using a BSM

Example 30.2: Variable Star Data

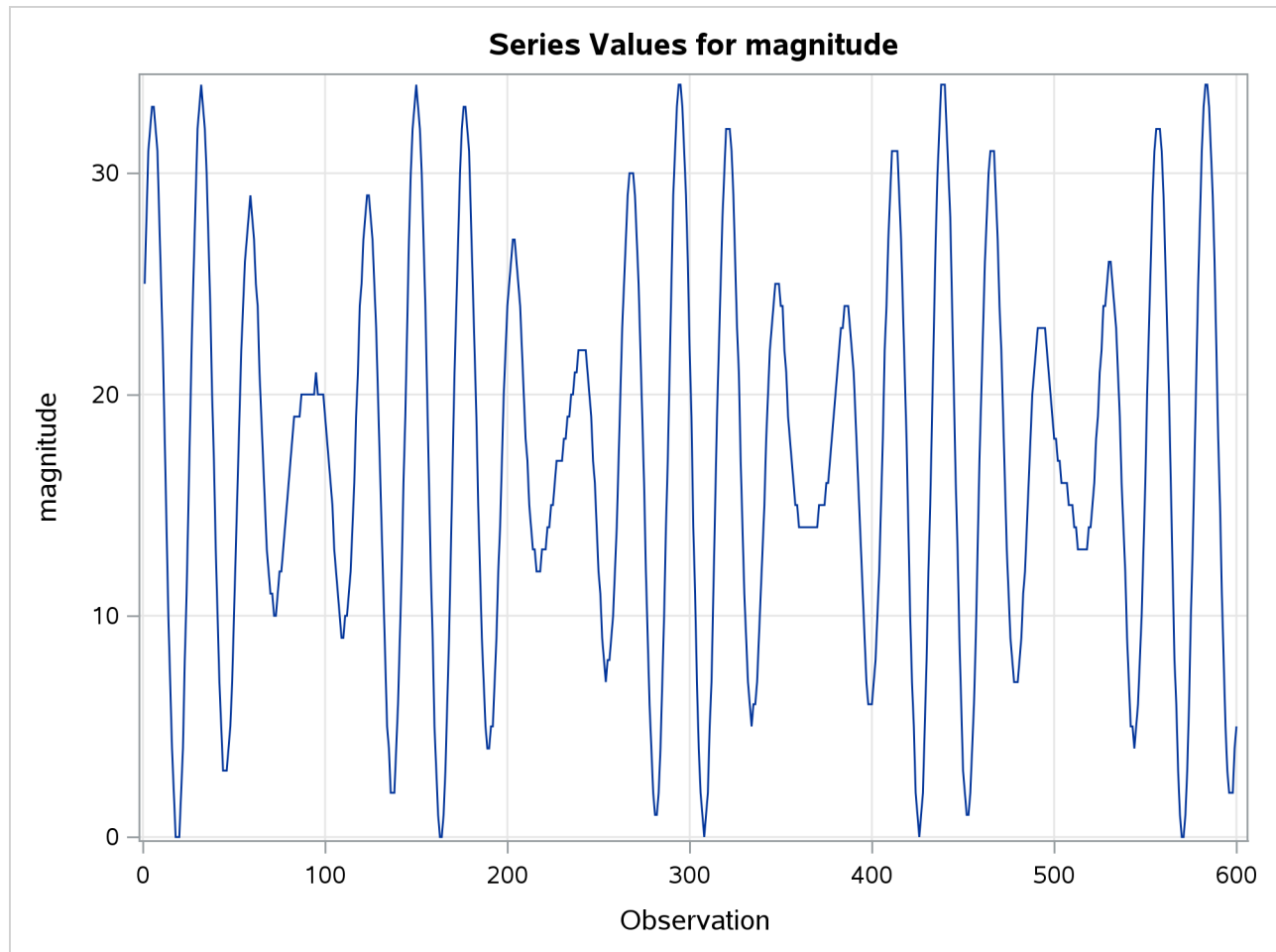
The series in this example is studied in detail in Bloomfield (2000). This series consists of brightness measurements (magnitude) of a variable star taken at midnight for 600 consecutive days. The data can be downloaded from a time series archive maintained by the University of York, England (<http://www.york.ac.uk/depts/maths/data/ts/welcome.htm> (series number 26)). The following DATA step statements read the data in a SAS data set:

```
data star;
    input magnitude @@;
    day = _n_;
    format day best4.;
datalines;
25 28 31 32 33 33 32 31 28 25 22 18
14 10 7 4 2 0 0 0 2 4 8 11
15 19 23 26 29 32 33 34 33 32 30 27
24 20 17 13 10 7 5 3 3 3 4 5
7 10 13 16 19 22 24 26 27 28 29 28

... more lines ...
```

The following statements use the TIMESERIES procedure to get a timeseries plot of the series (see [Output 30.2.1](#)):

```
proc timeseries data=star plot=series;
    var magnitude;
run;
```

Output 30.2.1 Plot of Star Brightness on Successive Days

The plot clearly shows the cyclic nature of the series. Bloomfield shows that the series is very well explained by a model that includes two deterministic cycles that have periods 29.0003 and 24.0001 days, a constant term, and a simple error term. He also mentions the difficulty involved in estimating the periods from the data (Bloomfield 2000, Chapter 3). In his case the cycle periods are estimated by least squares, and the sum of squares surface has multiple local optima and ridges. The following statements show how to use the UCM procedure to fit this two-cycle model to the series. The constant term in the model is specified by holding the variance parameter of the level component to 0.

```
/* Load data into mylib */
data mylib.star;
  set star;
run;

proc ucm data= mylib.star;
  id day interval=obs;
  model magnitude;
  irregular;
  level var=0 noest;
```

```

cycle;
cycle;
estimate;
run;

```

The final parameter estimates and the goodness-of-fit statistics are shown in [Output 30.2.2](#) and [Output 30.2.3](#), respectively. The model fit appears to be good.

Output 30.2.2 Two-Cycle Model: Parameter Estimates
The UCM Procedure

Final Estimates of the Free Parameters					
Component	Parameter	Estimate	Approx Std Error	t Value	Approx Pr > t
Irregular	Error Variance	0.09257	0.0054	17.19	<.0001
Cycle_1	Damping Factor	1.00000	0.0000	5519514	<.0001
Cycle_1	Period	29.00036	0.0023	12770.4	<.0001
Cycle_1	Error Variance	0.00000882	0.0000	1.67	0.0944
Cycle_2	Damping Factor	1.00000	0.0000	4718334	<.0001
Cycle_2	Period	24.00011	0.0019	12547.2	<.0001
Cycle_2	Error Variance	0.00000535	0.0000	1.50	0.1330

Output 30.2.3 Two-Cycle Model: Goodness of Fit

Fit Statistics Based on Residuals	
Mean Squared Error	0.12072
Root Mean Squared Error	0.34745
Mean Absolute Percentage Error	2.65141
Maximum Percent Error	36.38991
R-Square	0.99850
Adjusted R-Square	0.99849
Random Walk R-Square	0.97281
Amemiya's Adjusted R-Square	0.99847
Number of non-missing residuals used for computing the fit statistics = 599	

A summary of the cycles in the model is given in [Output 30.2.4](#).

Output 30.2.4 Two-Cycle Model: Summary

Name	Type	period	Rho	ErrorVar
Cycle_1	Stationary	29.00036	1.00000	0.00000882
Cycle_2	Stationary	24.00011	1.00000	0.00000535

Note that the estimated periods are the same as in Bloomfield's model, the damping factors are nearly equal to 1.0, and the disturbance variances are very close to 0, implying persistent deterministic cycles. In fact, this model is identical to Bloomfield's model.

Finally, [Output 30.2.5](#) shows the likely outlying observations that do not conform to the fitted model. The first outlier in this table, day 301, is also flagged by Bloomfield (according to his zero-based index system, it

is day 300).

Output 30.2.5 Two-Cycle Model: Outlier Summary

Outlier Summary						
		Standard				
day	Break Type	Estimate	Error	Chi-Square	DF	Pr > ChiSq
301	Additive Outlier	0.83088	0.3064	7.35	1	0.0067
407	Additive Outlier	0.82905	0.3065	7.32	1	0.0068
351	Additive Outlier	0.80169	0.3065	6.84	1	0.0089
119	Additive Outlier	0.80261	0.3069	6.84	1	0.0089
191	Additive Outlier	0.78733	0.3065	6.60	1	0.0102

Example 30.3: Modeling Long Seasonal Patterns

This example illustrates some of the techniques that you can use to model long seasonal patterns in a series. If the seasonal pattern is of moderate length and the underlying dynamics are simple, then it is easily modeled by using the basic settings of the SEASON statement, and these additional techniques are not needed. However, if the seasonal pattern has a long season length and/or has complex stochastic dynamics, then the techniques that are discussed here can be useful. You can obtain parsimonious models for a long seasonal pattern by using an appropriate subset of trigonometric harmonics, by using a suitable spline function, or by using a block-season pattern in combination with a much shorter seasonal component. You can also vary the disturbance variances of the subcomponents that combine to form the seasonal component.

The time series in this example consists of the number of calls that are received per shift at a call center. Each shift is six hours long, and the first shift of the day begins at midnight, resulting in four shifts per day. The observations are available from December 15, 1999, to April 30, 2000. This series is seasonal, with a season length of 28, which is moderate; in fact, there is no particular need to use pattern approximation techniques in this case. However, the series is adequate for demonstration purposes. This example proceeds as follows. First, an initial model with a full seasonal component is created. This model is used as a baseline for comparing alternative models created by the techniques that are being illustrated. In practice, any candidate model is first checked for adequacy by using various diagnostic procedures. In this example, the main focus is on the different ways that a long seasonal pattern can be modeled, and no model diagnostics are performed for the models that are being considered. The alternative models are compared by using the sum of absolute prediction errors in the holdout region.

The following DATA step statements create the input data set for this example:

```
data callCenter;
  input calls @@;
  label calls= "Number of Calls Received in a 6 Hour Shift";
  start = '15dec99:00:00'dt;
  datetime = INTNX( 'dthour6', start, _n_-1 );
  format datetime datetime10.;
datalines;
  18    122    244    128    19    113    230    119    17    112
  219    93    14    73    139    53    11    32    74    56
  15    137    289    153    20    125    227    106    16    101
```

```
... more lines ...
```

Initial exploration of the series clearly indicates that the series does not show any significant trend, and time of day and day of the week have a significant influence on the number of calls received. These considerations suggest a simple random walk trend model along with a seasonal component of season length 28, the total number of shifts in a week. The following statements specify this model. Note the PRINT=HARMONICS option in the SEASON statement, which produces a table that lists the full set of harmonics contributing to the seasonal along with the significance of their contribution. This table will be useful later in choosing a subset trigonometric model. The BACK=28 and LEAD=28 specifications in the FORECAST statement create a holdout region of 28 observations. The sum of absolute prediction errors (SAE) in this holdout region is used to compare the different models.

```
/* Load data into mylib */
data mylib.callCenter;
    set callCenter;
run;

proc ucm data= mylib.callCenter;
    id datetime interval=dthour6;
    model calls;
    irregular;
    level;
    season length=28 type=trig
        print=(harmonics);
    estimate back=28;
    forecast back=28 lead=28;
run;
```

The forecasting performance of this model in the holdout region is shown in [Output 30.3.1](#). The SAE is 516.22, which appears in the last row of the holdout analysis table.

Output 30.3.1 Predictions in the Holdout Region: Baseline Model

datetime	Actual	Forecast	Error	SAE
24APR00:00	12.000	-4.004	16.004	16.004
24APR00:06	136.000	110.825	25.175	41.179
24APR00:12	295.000	262.820	32.180	73.360
24APR00:18	172.000	145.127	26.873	100.232
25APR00:00	20.000	2.188	17.812	118.044
25APR00:06	127.000	105.442	21.558	139.602
25APR00:12	236.000	217.043	18.957	158.559
25APR00:18	125.000	114.313	10.687	169.246
26APR00:00	16.000	2.855	13.145	182.391
26APR00:06	108.000	95.202	12.798	195.189
26APR00:12	207.000	194.184	12.816	208.005
26APR00:18	112.000	97.687	14.313	222.317
27APR00:00	15.000	1.270	13.730	236.047
27APR00:06	98.000	85.875	12.125	248.172
27APR00:12	200.000	184.891	15.109	263.281
27APR00:18	113.000	93.113	19.887	283.168
28APR00:00	15.000	-1.120	16.120	299.288
28APR00:06	104.000	84.983	19.017	318.305
28APR00:12	205.000	177.940	27.060	345.365
28APR00:18	89.000	64.292	24.708	370.073
29APR00:00	12.000	-6.020	18.020	388.093
29APR00:06	68.000	46.286	21.714	409.807
29APR00:12	116.000	100.339	15.661	425.468
29APR00:18	54.000	34.700	19.300	444.768
30APR00:00	10.000	-6.209	16.209	460.978
30APR00:06	30.000	12.167	17.833	478.811
30APR00:12	66.000	49.524	16.476	495.287
30APR00:18	61.000	40.071	20.929	516.216

Now that a baseline model is created, the exploration for alternative models can begin. The review of the harmonic table in [Output 30.3.2](#) shows that all but the last three harmonics are significant, and deleting any of them to form a subset trigonometric seasonal component will lead to a poorer model. The last three harmonics, 12th, 13th, and 14th, with periods of 2.333, 2.15, and 2.0, respectively, do appear to be possible choices for deletion. Note that the disturbance variance of the seasonal component is not very insignificant (see [Output 30.3.3](#)); therefore the seasonal component is stochastic and the preceding logic, which is based on the final state estimate, provides only a rough guideline.

Output 30.3.2 Harmonic Analysis of the Season: Initial Model**The UCM Procedure**

Harmonic Analysis of Trigonometric Seasons (Based on the Final State)						
Season		Harmonic	Period	Chi-Square	DF	Pr > ChiSq
Name	Length					
Season	28	1	28.00000	234.19	2	<.0001
Season	28	2	14.00000	264.19	2	<.0001
Season	28	3	9.33333	95.65	2	<.0001
Season	28	4	7.00000	105.64	2	<.0001
Season	28	5	5.60000	146.74	2	<.0001
Season	28	6	4.66667	121.93	2	<.0001
Season	28	7	4.00000	4299.12	2	<.0001
Season	28	8	3.50000	150.79	2	<.0001
Season	28	9	3.11111	89.68	2	<.0001
Season	28	10	2.80000	8.95	2	0.0114
Season	28	11	2.54545	6.14	2	0.0464
Season	28	12	2.33333	2.20	2	0.3325
Season	28	13	2.15385	3.40	2	0.1828
Season	28	14	2.00000	2.33	1	0.1272

Output 30.3.3 Parameter Estimates: Initial Model**The UCM Procedure**

Final Estimates of the Free Parameters					
Component	Parameter	Estimate	Approx Std Error	t Value	Approx Pr > t
Irregular	Error Variance	92.14591	13.1099	7.03	<.0001
Level	Error Variance	44.83595	10.6547	4.21	<.0001
Season	Error Variance	0.01250	0.0065	1.92	0.0551

The following statements fit a subset trigonometric model formed by dropping the last three harmonics by specifying the DROPH= option in the SEASON statement:

```
proc ucm data=mylib.callCenter;
  id datetime interval=dthour6;
  model calls;
  irregular;
  level;
  season length=28 type=trig droph=12 13 14;
  estimate back=28;
  forecast back=28 lead=28;
run;
```

The last row of the holdout region prediction analysis table for the preceding model is shown in [Output 30.3.4](#). It shows that the subset trigonometric model has better prediction performance in the holdout region than the full trigonometric model; its SAE is 471.53, compared to an SAE of 516.22 for the full model.

Output 30.3.4 SAE for the Subset Trigonometric Model

datetime	Actual	Forecast	Error	SAE
30APR00:18	61.000	40.836	20.164	471.534

The following statements illustrate a spline approximation to this seasonal component. In the spline specification the knot placement is quite important, and usually some experimentation is needed. In the following model the knots are placed at the beginning and the middle of each day. Note that the knots at the beginning and end of the season, 1 and 28 in this case, should not be listed in the knot list because knots are always placed there anyway.

```
proc ucm data=mylib.callCenter;
  id datetime interval=dthour6;
  model calls;
  irregular;
  level;
  splinesseason length=28
    knots=3 5 7 9 11 13 15 17 19 21 23 25 27
    degree=3;
  estimate back=28;
  forecast back=28 lead=28;
run;
```

The spline season model takes about half the time to fit that the baseline model takes. The last row of the holdout region prediction analysis table for this model is shown in [Output 30.3.5](#), which shows that the spline season model performs even better than the previous two models in the holdout region; its SAE is 313.79, compared to an SAE of 471.53 for the previous model.

Output 30.3.5 SAE for the Spline Season Model

datetime	Actual	Forecast	Error	SAE
30APR00:18	61.000	23.350	37.650	313.792

The following statements illustrate yet another way to approximate a long seasonal component. Here a combination of BLOCKSEASON and SEASON statements results in a seasonal component that is a sum of two seasonal patterns: one seasonal pattern is simply a regular season with season length 4 that captures the *within-day* seasonal pattern, and the other seasonal pattern is a block seasonal pattern that remains constant during the day but varies from day to day within a week. Note the use of the NLOPTIONS statement to change the optimization technique during the parameter estimation to DBLDOG, which in this case performs better than the default technique, TRUREG.

```
proc ucm data=mylib.callCenter;
  id datetime interval=dthour6;
  model calls;
  irregular;
  level;
  season length=4 type=trig;
  blockseason nblocks=7 blocksize=4
    type=trig;
```

```

estimate back=28;
forecast back=28 lead=28;
nloptions tech=dbldog;
run;

```

This model also takes about half the time to fit that the baseline model takes. The last row of the holdout region prediction analysis table for this model is shown in [Output 30.3.6](#), which shows that the block season model does slightly better than the baseline model but not as well as the other two models; its SAE is 508.52, compared to an SAE of 516.22 for the baseline model.

Output 30.3.6 SAE for the Block Season Model

datetime	Actual	Forecast	Error	SAE
30APR00:18	61.000	39.339	21.661	508.522

This example showed a few different ways to model a long seasonal pattern. It showed that parsimonious models for long seasonal patterns can be useful, and in some cases even more effective than the full model. Moreover, for very long seasonal patterns the high memory requirements and long computing times might make full models impractical.

Example 30.4: Modeling Time-Varying Regression Effects

In April 1979, the Albuquerque Police Department began a special enforcement program aimed at reducing the number of DWI (driving while intoxicated) accidents. The program was administered by a squad of police officers, who used breath alcohol testing (BAT) devices and a van that houses a BAT device (Batmobile). These data were collected by the Division of Governmental Research of the University of New Mexico, under a contract with the National Highway Traffic Safety Administration of the U.S. Department of Transportation, to evaluate the Batmobile program. The first 29 observations are for a control period, and the next 23 observations are for the experimental (Batmobile) period. The data consist of two variables: ACC, which represents injuries and fatalities from Wednesday to Saturday nighttime accidents, and FUEL, which represents fuel consumption (millions of gallons) in Albuquerque. The variables are measured quarterly starting from the first quarter of 1972 up to the last quarter of 1984, covering the span of 13 years. The following DATA step statements create the input data set:

```

data bat;
  input ACC FUEL @@;
  batProgram = 0;
  if _n_ > 29 then batProgram = 1;
  date = INTNX( 'qtr', '1jan1972'd, _n_ - 1 );
  format date YYQC8.;
datalines;
192      32.592      238      37.250      232      40.032
246      35.852      185      38.226      274      38.711
266      43.139      196      40.434      170      35.898
234      37.111      272      38.944      234      37.717
210      37.861      280      42.524      246      43.965
248      41.976      269      42.918      326      49.789
342      48.454      257      45.056      280      49.385
290      42.524      356      51.224      295      48.562

```

279	48.167	330	51.362	354	54.646
331	53.398	291	50.584	377	51.320
327	50.810	301	46.272	269	48.664
314	48.122	318	47.483	288	44.732
242	46.143	268	44.129	327	46.258
253	48.230	215	46.459	263	50.686
319	49.681	263	51.029	206	47.236
286	51.717	323	51.824	306	49.380
230	47.961	304	46.039	311	55.683
292	52.263				

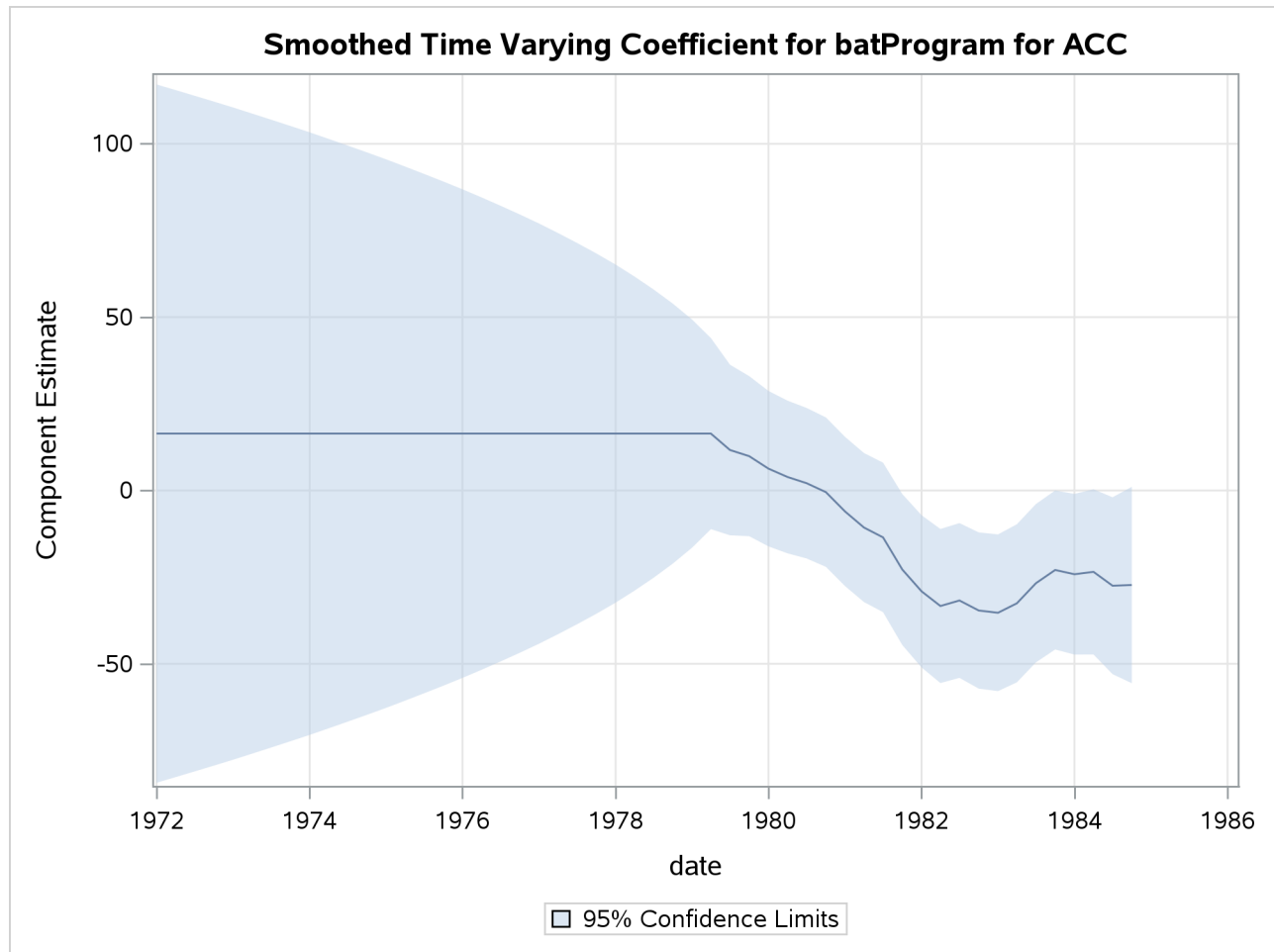
;

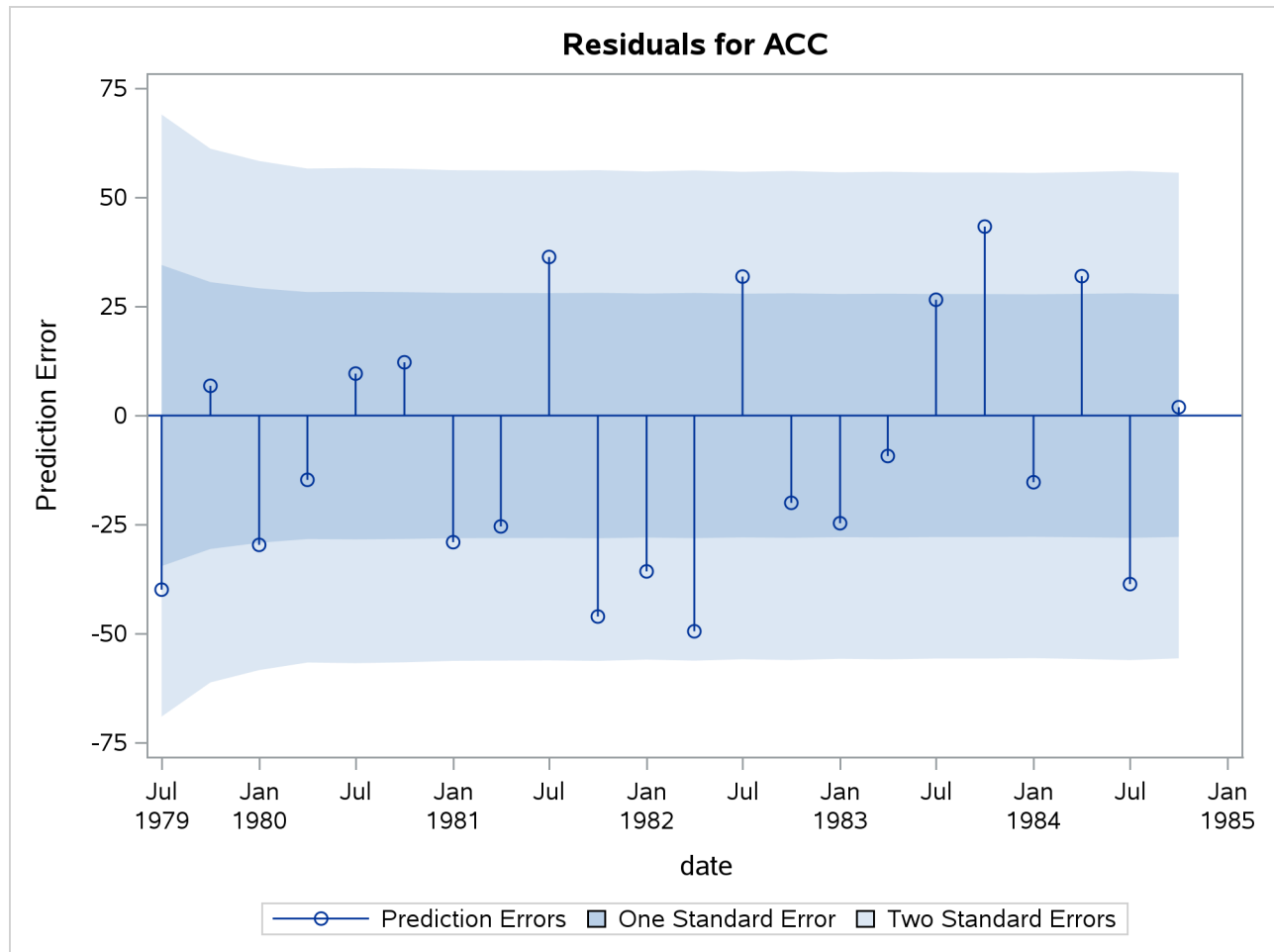
There are a number of ways to study these data and the question of the effectiveness of the BAT program. One possibility is to study the *before-after* difference in the injuries and fatalities per million gallons of fuel consumed, by regressing ACC on FUEL and the dummy variable BATPROGRAM, whose value is 0 before the program began and 1 while the program is in place. However, it is possible that the effect of the Batmobiles might well be cumulative, because as awareness of the program becomes dispersed, its effectiveness as a deterrent to driving while intoxicated increases. This suggests that the regression coefficient of the BATPROGRAM variable might be *time-varying*. The following program fits a model that incorporates these considerations. A seasonal component is included in the model since it is easy to see that the data show strong quarterly seasonality.

```
/* Load data into mylib */
data mylib.bat;
    set bat;
run;

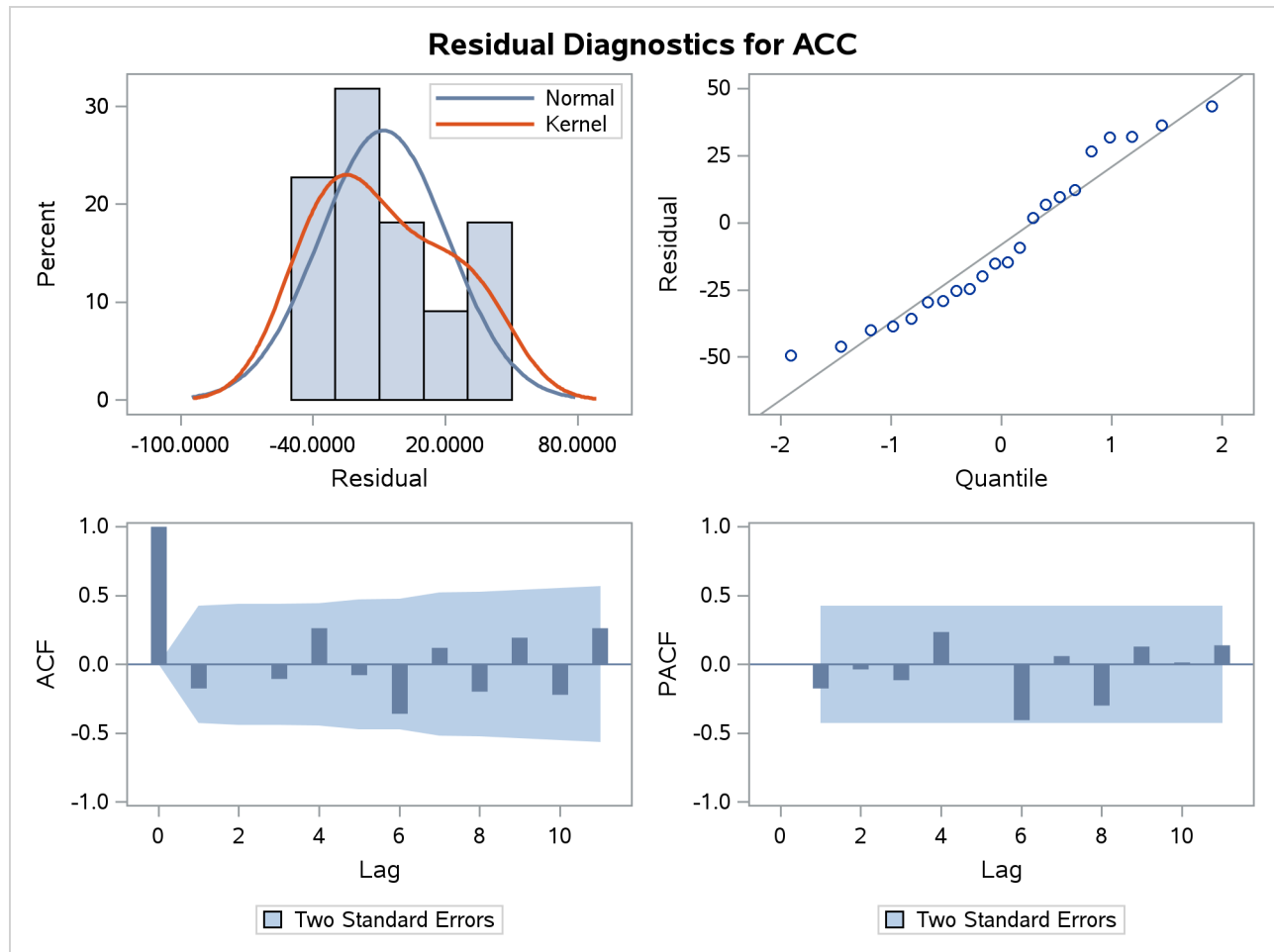
proc ucm data=mylib.bat;
    model acc = fuel;
    id date interval=qtr;
    irregular;
    level var=0 noest;
    randomreg batProgram / plot=smooth;
    season length=4 var=0 noest plot=smooth;
    estimate plot=(panel residual);
    forecast plot=forecasts lead=0;
run;
```

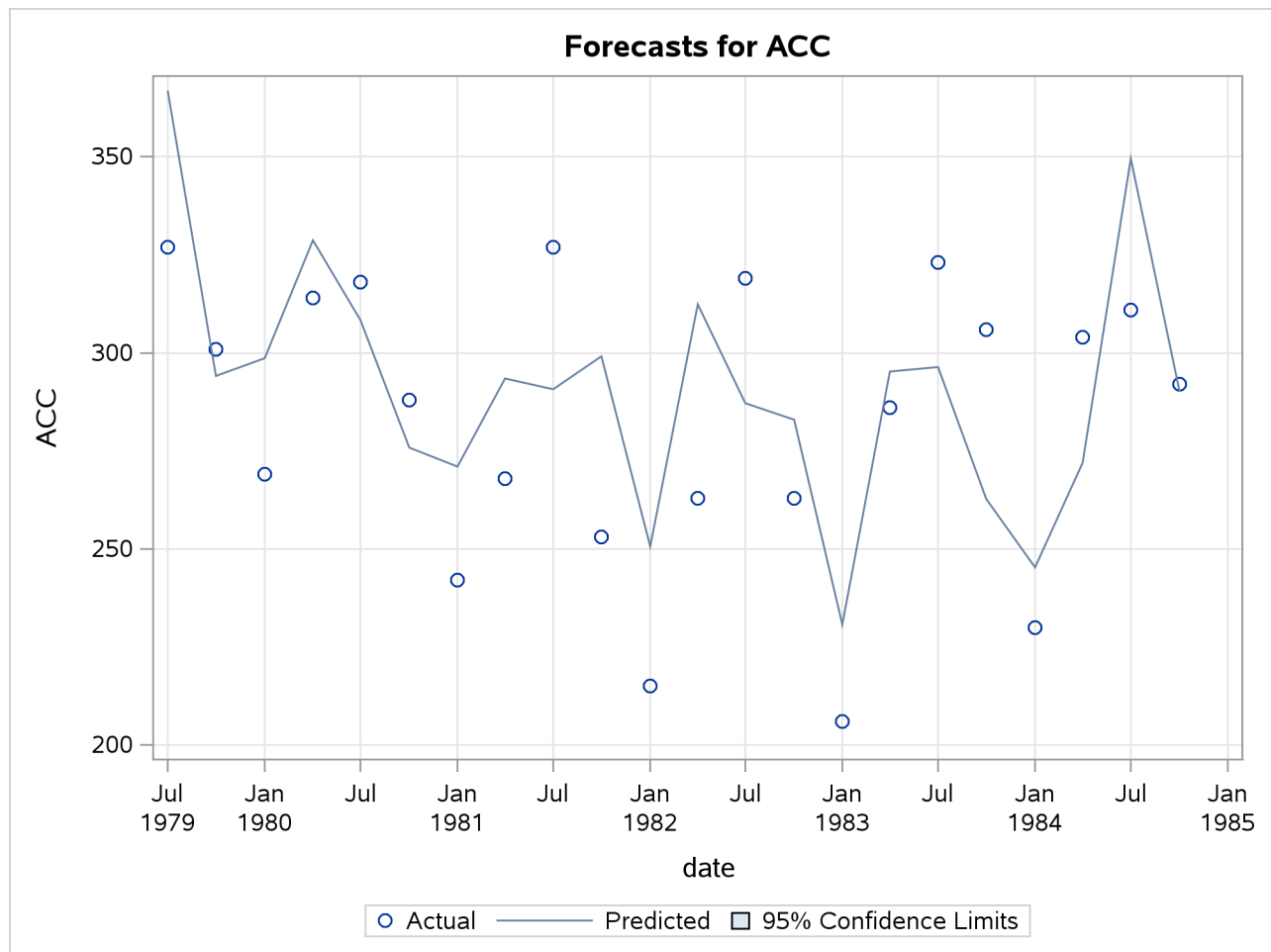
The model seems to fit the data adequately. No data are withheld for model validation because the series is relatively short. The plot of the time-varying coefficient of BATPROGRAM is shown in [Output 30.4.1](#). As expected, it shows that the effectiveness of the program increases as awareness of the program becomes dispersed. The effectiveness eventually seems to level off. The residual diagnostic plots are shown in [Output 30.4.2](#) and [Output 30.4.3](#), the forecast plot is in [Output 30.4.4](#), the goodness-of-fit statistics are in [Output 30.4.5](#), and the parameter estimates are in [Output 30.4.6](#).

Output 30.4.1 Time-Varying Regression Coefficient of BATPROGRAM

Output 30.4.2 Residuals for the Time-Varying Regression Model

Output 30.4.3 Residual Diagnostics for the Time-Varying Regression Model



Output 30.4.4 One-Step-Ahead Forecasts for the Time-Varying Regression Model**Output 30.4.5** Model Fit for the Time-Varying Regression Model

Fit Statistics Based on Residuals	
Mean Squared Error	866.75562
Root Mean Squared Error	29.44071
Mean Absolute Percentage Error	9.50326
Maximum Percent Error	14.15368
R-Square	0.32646
Adjusted R-Square	0.29278
Random Walk R-Square	0.63010
Amemiya's Adjusted R-Square	0.19175
Number of non-missing residuals used for computing the fit statistics = 22	

Output 30.4.6 Parameter Estimates for the Time-Varying Regression Model**The UCM Procedure**

Final Estimates of the Free Parameters					
Component	Parameter	Estimate	Approx Std Error	t Value	Approx Pr > t
Irregular	Error Variance	480.92258	109.2198	4.40	<.0001
FUEL	Coefficient	6.23279	0.6753	9.23	<.0001
batProgram	Error Variance	84.22334	79.8817	1.05	0.2917

Example 30.5: Trend Removal Using the Hodrick-Prescott Filter

The Hodrick-Prescott filter (Hodrick and Prescott 1997) is a popular tool in macroeconomics for fitting a smooth trend to time series. It is well known that the trend computation according to this filter is equivalent to fitting the local linear trend plus irregular model with the level disturbance variance restricted to 0 and the slope disturbance variance restricted to be a suitable multiple of the irregular component variance. The multiple used depends on the frequency of the series; for example, for quarterly series the commonly recommended multiple is $1/1600 = 0.000625$. For other intervals there is no consensus, but a frequently suggested value for monthly series is $1/14400$ and the value for an annual series can range from $1/400 = 0.0025$ to $1/7 = 0.15$. The data set considered in this example consists of quarterly GNP values for the United States from 1960 to 1991.

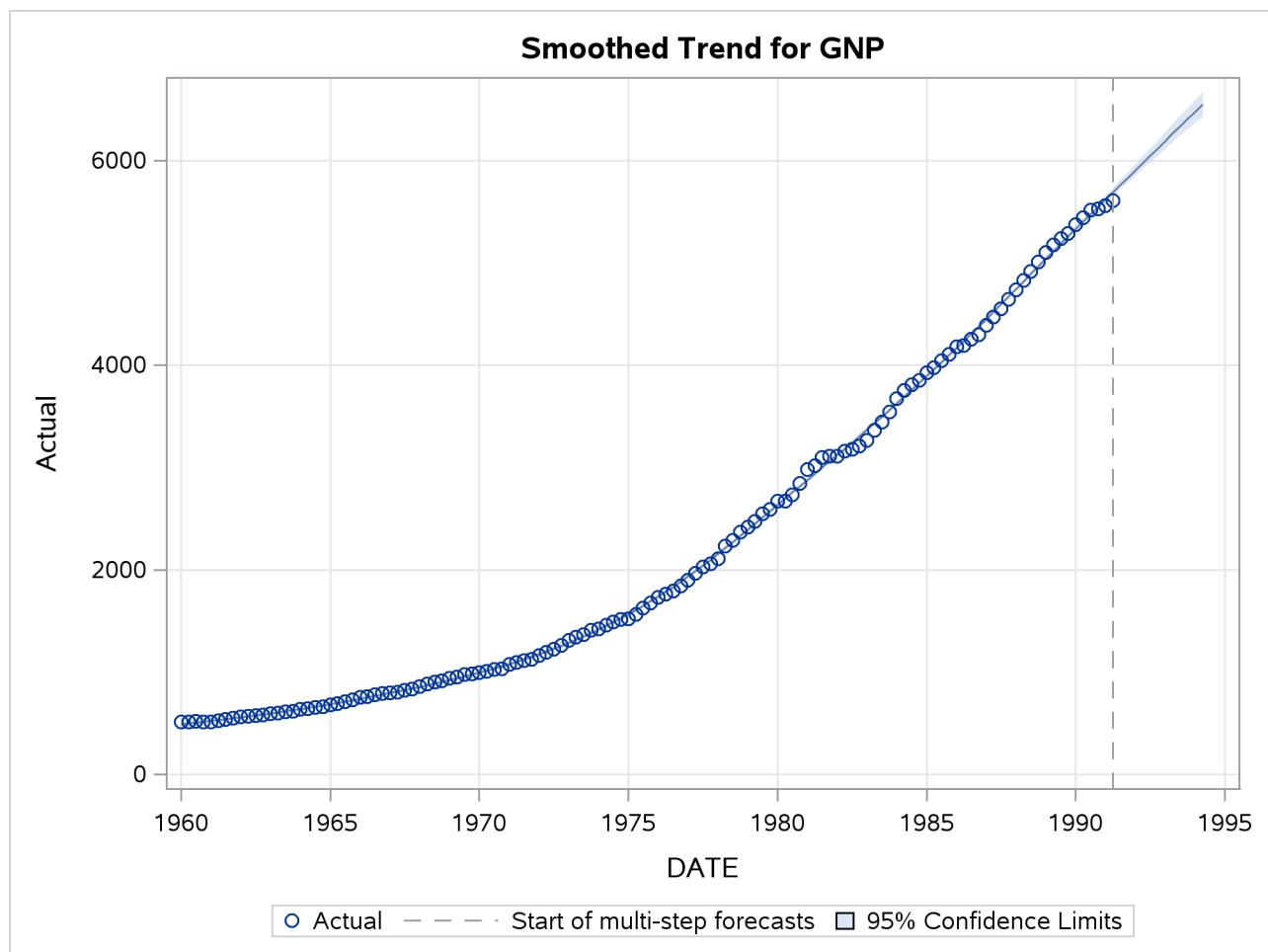
In the UCM procedure statements that follow, the presence of the PROFILE option in the ESTIMATE statement implies that the restriction that the disturbance variance of the slope component be fixed at 0.000625 is interpreted differently: it implies that the disturbance variance of the slope component be restricted to be 0.000625 *times* the estimated irregular component variance, as needed for the Hodrick-Prescott filter. The plot of the fitted trend is shown in [Output 30.5.1](#), and the plot of the smoothed irregular component, which corresponds to the detrended series, is given in [Output 30.5.2](#). The detrended series can be further analyzed for business cycles.

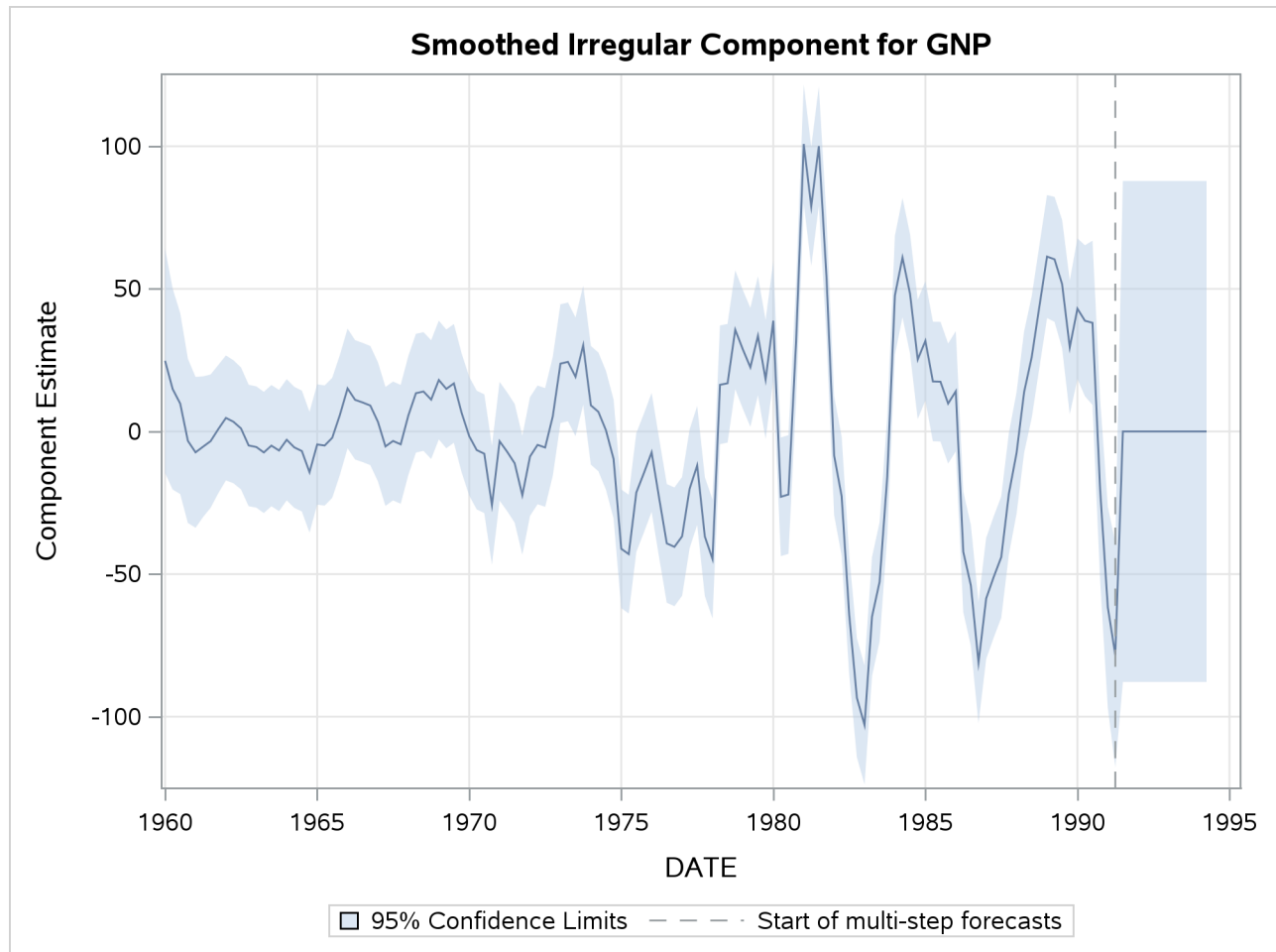
```

/* Load data into mylib */
data mylib.gnp;
  set sashelp.gnp;
run;

proc ucm data=mylib.gnp;
  id date interval=qtr;
  model gnp;
  irregular plot=smooth;
  level var=0 noest plot=smooth;
  slope var=0.000625 noest;
  estimate PROFILE;
  forecast plot=(decomp);
run;

```

Output 30.5.1 Smoothed Trend for the GNP Series as per the Hodrick-Prescott Filter

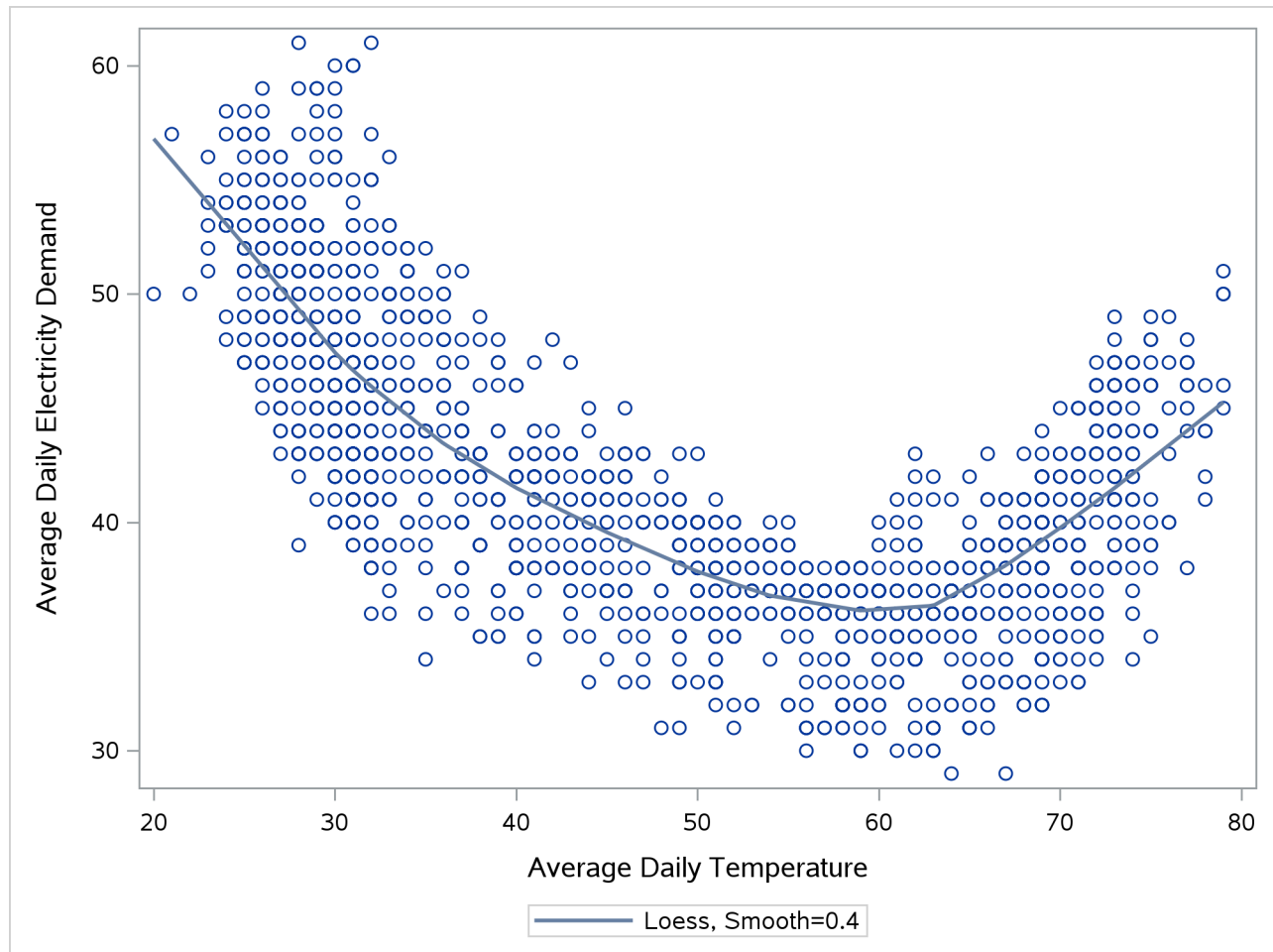
Output 30.5.2 Detrended GNP Series

Example 30.6: Using Splines to Incorporate Nonlinear Effects

The data in this example are created to mirror the electricity demand and temperature data recorded at a utility company in the midwest region of the United States. The data set (not shown), `utility`, has three variables: `load`, `temp`, and `date`. The `load` column contains the daily electricity demand, the `temp` column has the average daily temperature readings, and the `date` column records the observation date.

The following statements produce a plot, shown in [Output 30.6.1](#), of electricity load versus temperature. Clearly the relationship is smooth but nonlinear: the load generally increases when the temperatures are higher or lower than the comfortable sixties.

```
proc sgplot data=utility;
  loess x=temp y=load / smooth=0.4;
run;
```

Output 30.6.1 Load versus Temperature Plot

The time series plot of the load (not shown) also shows that, apart from a day-of-the-week seasonal effect, there are no additional easily identifiable patterns in the series. The series has no apparent upward or downward trend. The following statements fit a UCM to the series that takes into account these observations. The particular choice of the model is a result of a little modeling exercise that compared a small number of competing models. The chosen model is adequate but by no means the best possible. The temperature effect is modeled by a deterministic three-degree spline with knots at 30, 40, 50, 60, and 75. The knot locations and the degree were chosen by visual inspection of the plot (Output 30.6.1). An autoregression component is used in place of the simple irregular component, which improved the residual analysis. The last 60 days of data are withheld for out-of-sample forecast evaluation (note the BACK= option in both the ESTIMATE and FORECAST statements). The OUTLIER statement is used to increase the number of outliers reported to 10. Since no CHECKBREAK option is used in the LEVEL statement, only the additive outliers are searched. In this example the use of the EXTRADIFFUSE= option in the ESTIMATE and FORECAST statements is useful for discarding some early one-step-ahead forecasts and residuals with large variance.

```
/* Load data into mylib */
data mylib.utility;
  set utility;
run;
```

```

proc ucm data=mylib.utility;
  id date interval=day;
  model load;
  autoreg;
  level plot=smooth;
  splinereg temp knots=30 40 50 65 75 degree=3
    variance=0 noest;
  season length=7 var=0 noest;
  estimate plot=panel back=60
    extradiffuse=50;
  outlier maxnum=10;
  forecast back=60 lead=60
    extradiffuse=50;
run;

```

The parameter estimates are given in [Output 30.6.2](#), and the residual goodness-of-fit statistics are shown in [Output 30.6.3](#). The residual diagnostic plots are shown in [Output 30.6.4](#). The ACF and PACF plots appear satisfactory, but the normality plots, particularly the Q-Q plot, show possible violations. It appears that, at least in part, this nonnormal behavior of the residuals might be attributable to the outliers in the series. The outlier summary table, [Output 30.6.5](#), shows the most likely outlying observations. Notice that most of these outliers are holidays, like July 4th, when the electricity load is lower than usual for that day of the week.

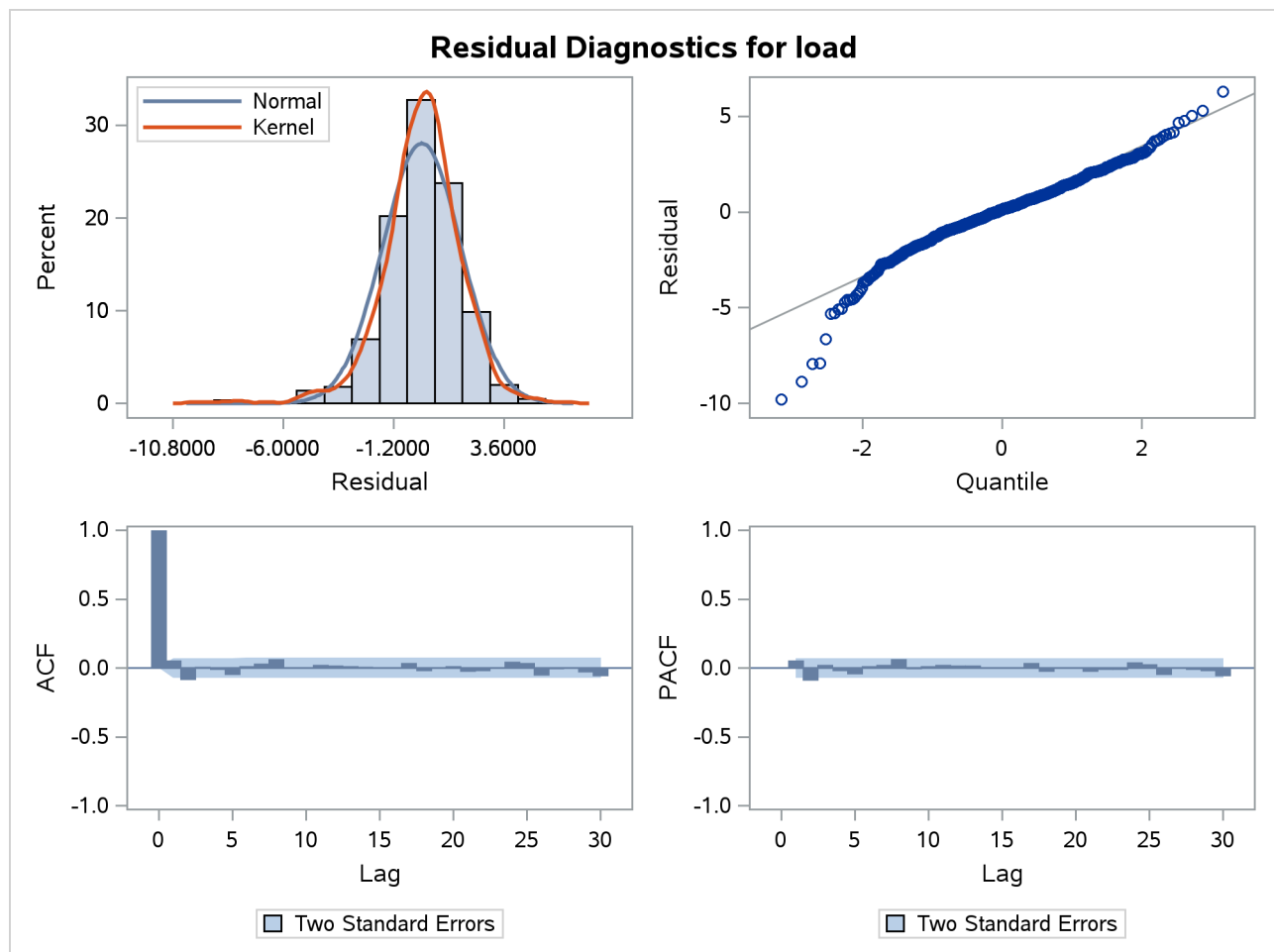
Output 30.6.2 Electricity Load: Parameter Estimates

The UCM Procedure

Final Estimates of the Free Parameters					
Component	Parameter	Estimate	Approx Std Error	t Value	Approx Pr > t
Level	Error Variance	0.21185	0.0503	4.22	<.0001
AutoReg	Damping Factor	0.57522	0.0347	16.60	<.0001
AutoReg	Error Variance	2.21057	0.2048	10.79	<.0001
temp	Spline Coefficient_1	4.72502	1.9400	2.44	0.0149
temp	Spline Coefficient_2	2.19116	1.7124	1.28	0.2007
temp	Spline Coefficient_3	-7.14492	1.5680	-4.56	<.0001
temp	Spline Coefficient_4	-11.39950	1.4510	-7.86	<.0001
temp	Spline Coefficient_5	-16.38055	1.3698	-11.96	<.0001
temp	Spline Coefficient_6	-18.76075	1.2890	-14.55	<.0001
temp	Spline Coefficient_7	-8.04628	1.0902	-7.38	<.0001
temp	Spline Coefficient_8	-2.30525	1.2510	-1.84	0.0654

Output 30.6.3 Electricity Load: Goodness-of-Fit

Fit Statistics Based on Residuals	
Mean Squared Error	2.90945
Root Mean Squared Error	1.70571
Mean Absolute Percentage Error	2.92586
Maximum Percent Error	14.96281
R-Square	0.92739
Adjusted R-Square	0.92721
Random Walk R-Square	0.69618
Amemiya's Adjusted R-Square	0.92684
Number of non-missing residuals used for computing the fit statistics = 791	

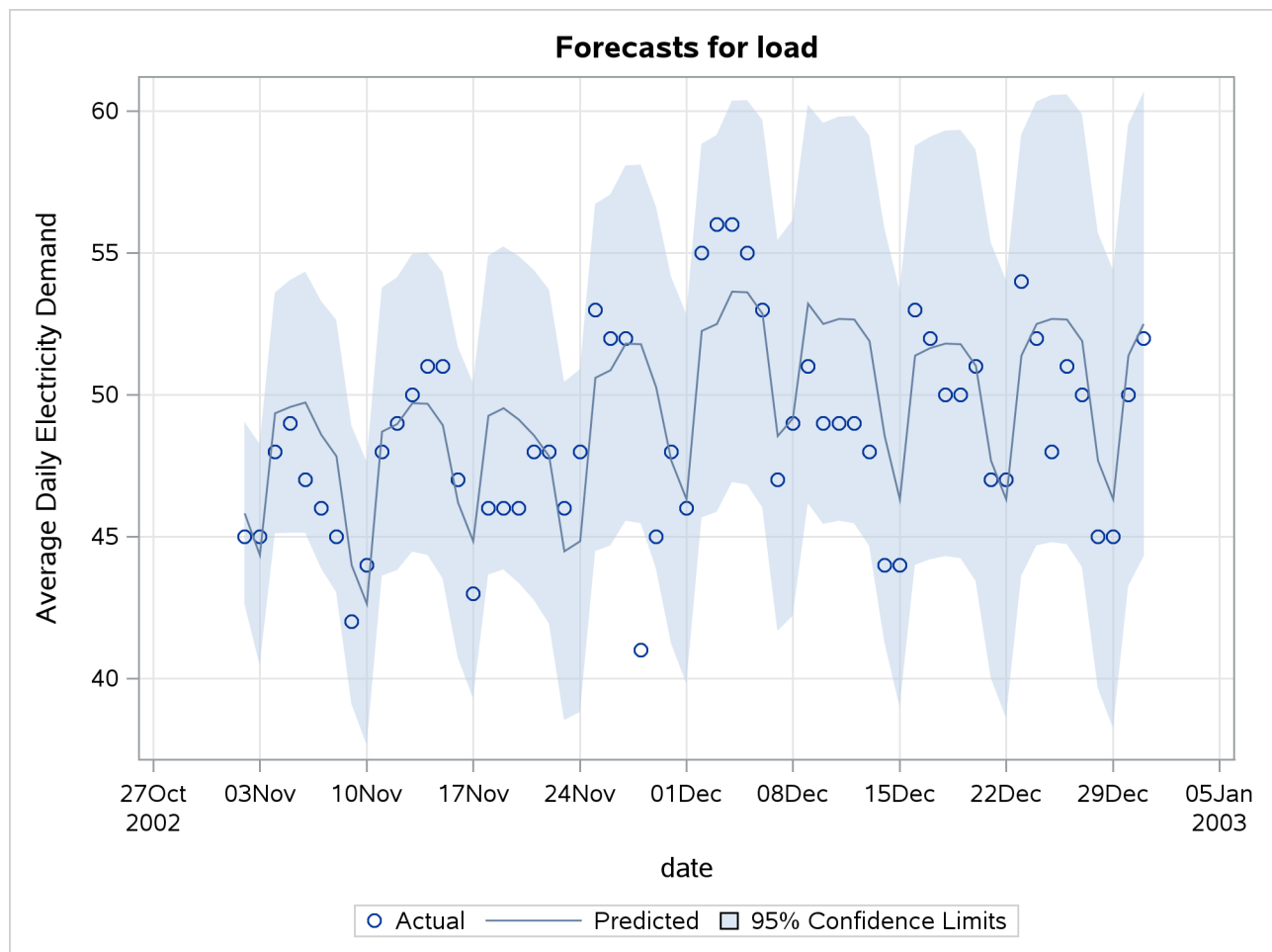
Output 30.6.4 Electricity Load: Residual Diagnostics

Output 30.6.5 Additive Outliers in the Electricity Load Series

	Time	Estimate	StdErr	ChiSq	DF	ProbChiSq
	04-JUL-2002	-7.99908	1.3417	35.54	1	<.0001
	04-JUL-2001	-6.55778	1.3384	24.01	1	<.0001
	25-NOV-1999	-5.85047	1.3380	19.12	1	<.0001
	03-SEP-2001	-5.67254	1.3389	17.95	1	<.0001
	02-SEP-2002	-5.49631	1.3378	16.88	1	<.0001
	23-NOV-2000	-5.27968	1.3374	15.58	1	<.0001
	03-JUL-2001	5.06557	1.3375	14.34	1	0.0002
	22-NOV-2001	-5.01550	1.3386	14.04	1	0.0002
	04-JUL-2000	-4.89965	1.3382	13.41	1	0.0003
	28-MAY-2001	-4.76135	1.3375	12.67	1	0.0004

The plot of the load forecasts for the withheld data is shown in [Output 30.6.6](#).

Output 30.6.6 Electricity Load: Forecast Evaluation of the Withheld Data

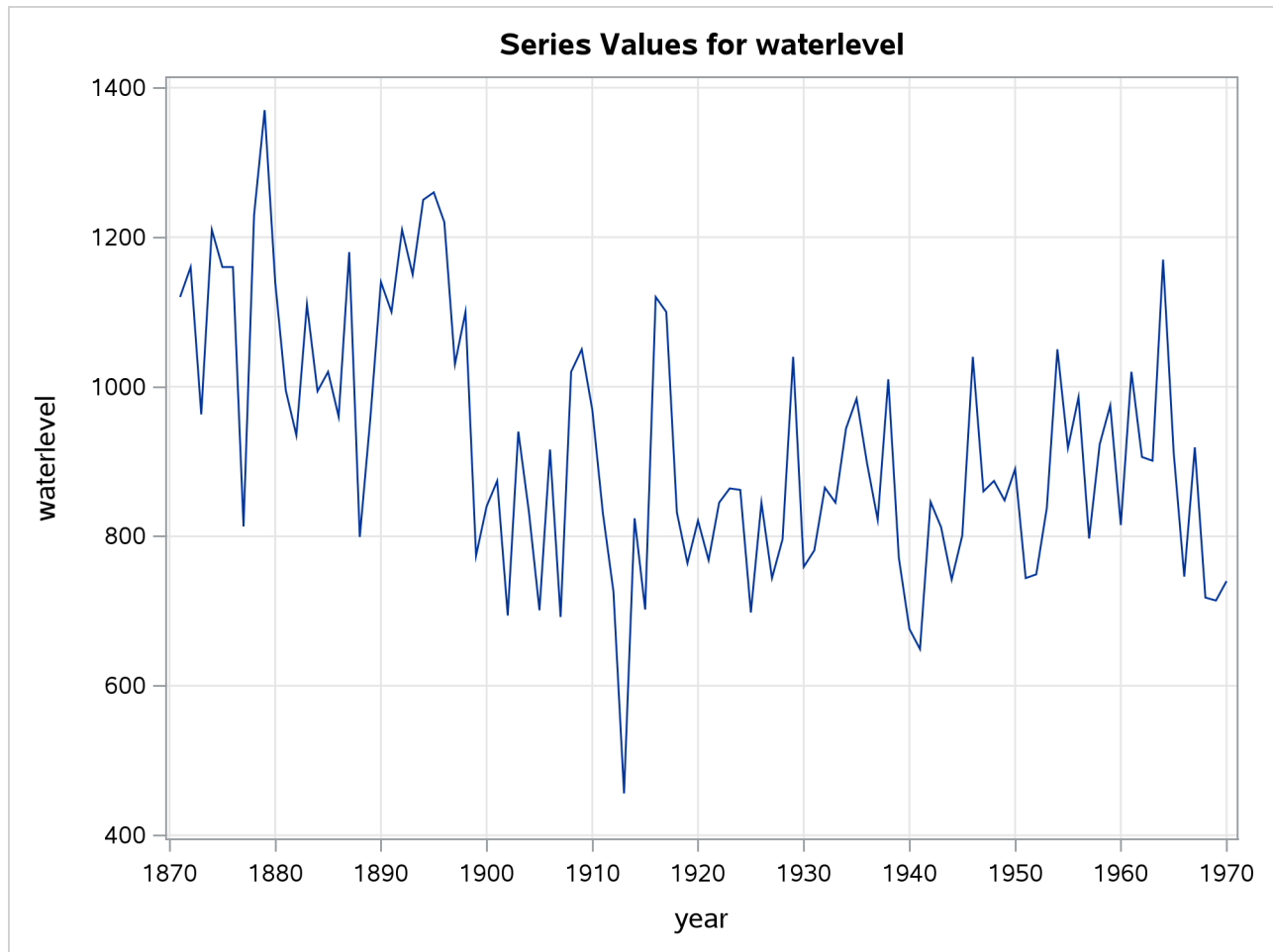


Example 30.7: Detection of Level Shift

The series in this example consists of the yearly water level readings of the Nile River recorded at Aswan, Egypt (Cobb 1978; De Jong and Penzer 1998). The readings are from the years 1871 to 1970. The series does not show any apparent trend or any other distinctive patterns; however, there is a shift in the water level starting at the year 1899. This shift could be attributed to the start of construction of a dam near Aswan in that year. A time series plot of this series is given in [Output 30.7.1](#). The following DATA step statements create the input data set:

```
data nile;
  input waterlevel @@;
  year = intnx( 'year', '1jan1871'd, _n_-1 );
  format year year4.;
datalines;
1120 1160 963 1210 1160 1160 813 1230 1370 1140
995 935 1110 994 1020 960 1180 799 958 1140
1100 1210 1150 1250 1260 1220 1030 1100 774 840
874 694 940 833 701 916 692 1020 1050 969
831 726 456 824 702 1120 1100 832 764 821
768 845 864 862 698 845 744 796 1040 759
781 865 845 944 984 897 822 1010 771 676
649 846 812 742 801 1040 860 874 848 890
744 749 838 1050 918 986 797 923 975 815
1020 906 901 1170 912 746 919 718 714 740
;

proc timeseries data=nile plot=series;
  id year interval=year;
  var waterlevel;
run;
```


Output 30.7.1 Nile Water Level

In this situation it is known that a shift in the water level occurred within the span of the series, and its effect can be easily taken into account by including an appropriate indicator variable as a regressor. However, in many situations such prior information is not available, and it is useful to detect such a shift by using data analysis. You can check for breaks in the level by using the [CHECKBREAK](#) option in the `LEVEL` statement. The following statements fit a simple locally constant level plus error model to the series:

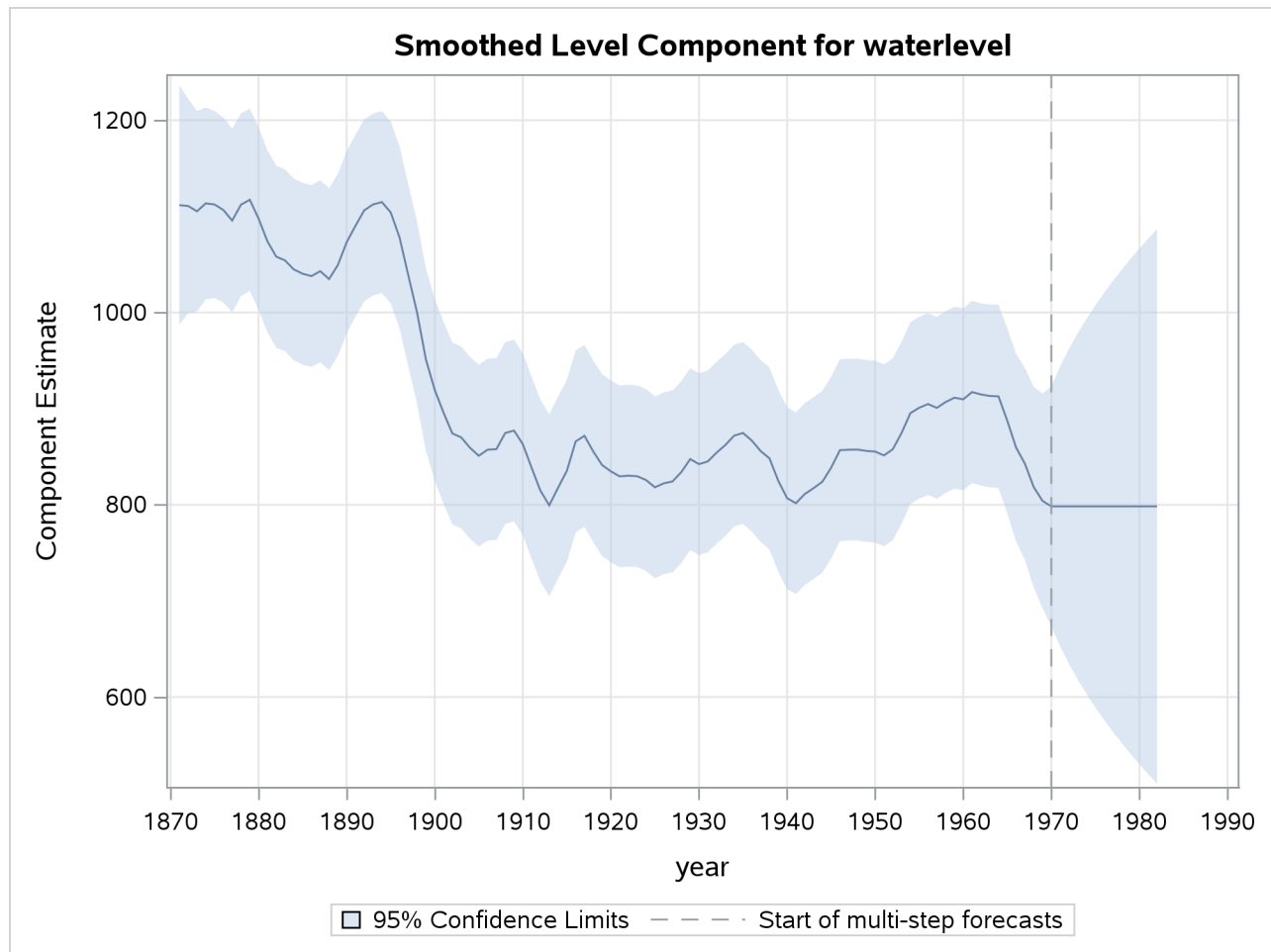
```
/* Load data into mylib */
data mylib.nile;
  set nile;
run;

proc ucm data=mylib.nile;
  id year interval=year;
  model waterlevel;
  irregular;
  level plot=smooth checkbreak;
  estimate;
  forecast plot=decomp;
```

```
run;
```

The plot in [Output 30.7.2](#) shows a noticeable drop in the smoothed water level around 1899.

Output 30.7.2 Smoothed Trend without the Shift of 1899



The “Outlier Summary” table in [Output 30.7.3](#) shows the most likely types of breaks and their locations within the series span. The shift of 1899 is easily detected.

Output 30.7.3 Detection of Structural Breaks in the Nile River Level

Outlier Summary						
Break		Standard				
year	Type	Estimate	Error	Chi-Square	DF	Pr > ChiSq
1899	Level	-315.73791	97.6398	10.46	1	0.0012

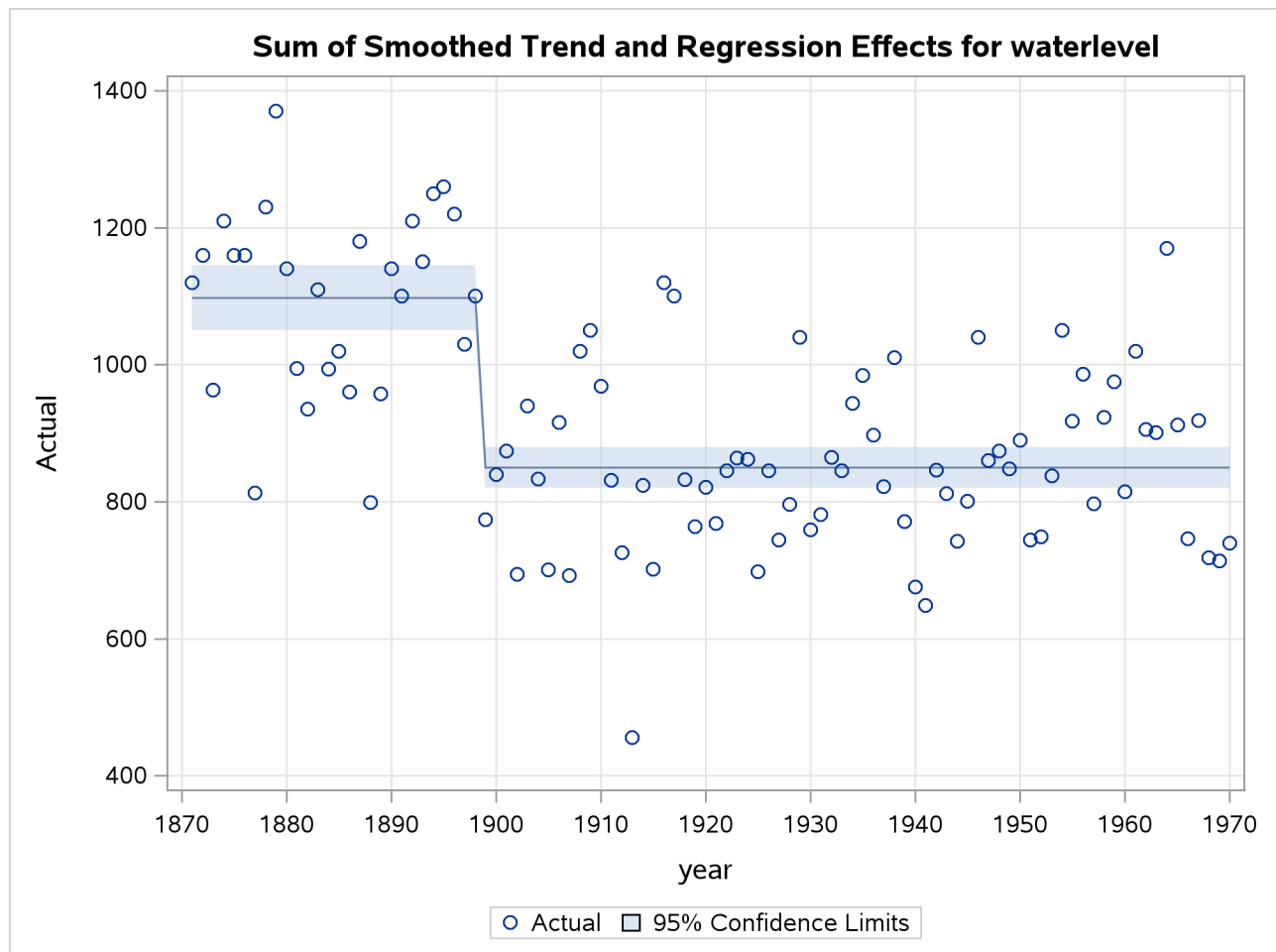
The following statements specify a UCM that models the level of the river as a locally constant series with a shift in the year 1899, represented by a dummy regressor (SHIFT1899):

```
data mylib.nile;
  set nile;
  shift1899 = ( year >= '1jan1899'd );
run;
```

```
proc ucm data=mylib.nile;
  id year interval=year;
  model waterlevel = shift1899;
  irregular;
  level;
  estimate;
  forecast plot=decomp;
run;
```

The plot in [Output 30.7.4](#) shows the smoothed trend, including the correction due to the shift in the year 1899. Notice the simplicity in the shape of the smoothed curve after the incorporation of the shift information.

Output 30.7.4 Smoothed Trend Plus Shift of 1899



Example 30.8: ARIMA Modeling

This example shows how you can use the UCM procedure for ARIMA modeling. The parameter estimates and predictions for ARIMA models obtained by using PROC UCM will be close to those obtained by using PROC ARIMA (in the presence of the ML option in its ESTIMATE statement) if the model is stationary or if the model is nonstationary and there are no missing values in the data. For more information about the ARIMA procedure, see Chapter 7, “ARIMA Procedure” (*SAS/ETS User’s Guide*). However, if there are missing values in the data and the model is nonstationary, then the UCM and ARIMA procedures can produce significantly different parameter estimates and predictions. An article by Kohn and Ansley (1986) suggests a statistically sound method of estimation, prediction, and interpolation for nonstationary ARIMA models with missing data. This method is based on an algorithm that is equivalent to the Kalman filtering and smoothing algorithm used in the UCM procedure. The results of an illustrative example in their article are reproduced here using the UCM procedure. In this example an $ARIMA(0,1,1) \times (0,1,1)_{12}$ model is applied to the logarithm of the air series in the `sashelp.air` data set. Four different missing value patterns are considered to highlight different aspects of the problem:

- `mylib.Data1`. The full data set of 144 observations.
- `mylib.Data2`. The set of 78 observations that omit January through November in each of the last 6 years.
- `mylib.Data3`. The data set with the 5 observations July 1949, June, July, and August 1957, and July 1960 missing.
- `mylib.Data4`. The data set with all July observations missing and June and August 1957 also missing.

The following DATA steps create these data sets:

```
data mylib.Data1;
    set sashelp.air;
    logair = log(air);
run;

data mylib.Data2;
    set mylib.data1;
    if year(date) >= 1955 and month(date) < 12 then logair = .;
run;

data mylib.Data3;
    set mylib.data1;
    if (year(date) = 1949 and month(date) = 7) then logair = .;
    if ( year(date) = 1957 and
        (month(date) = 6 or month(date) = 7 or month(date) = 8))
        then logair = .;
    if (year(date) = 1960 and month(date) = 7) then logair = .;
run;

data mylib.Data4;
    set mylib.data1;
```

```

    if month(date) = 7 then logair = .;
    if year(date) = 1957 and (month(date) = 6 or month(date) = 8)
        then logair = .;
run;

```

The following statements specify the $\text{ARIMA}(0, 1, 1) \times (0, 1, 1)_{12}$ model for the logair series in the first data set (mylib.Data1):

```

proc ucm data=mylib.Data1;
    id date interval=month;
    model logair;
    irregular q=1 sq=1 s=12;
    deplag lags=(1)(12) phi=1 1 noest;
    estimate outest=mylib.est1;
    forecast outfor=mylib.for1;
run;

```

Note that the moving average part of the model is specified by using the Q=, SQ=, and S= options in the IRREGULAR statement and the differencing operator, $(1 - B)(1 - B^{12})$, is specified by using the DEPLAG statement. The model does not contain an intercept term; therefore no LEVEL statement is needed. The parameter estimates are saved in a data set EST1 by using the OUTEST= option in the ESTIMATE statement and the forecasts and the component estimates are saved in a data set FOR1 by using the OUTFOR= option in the FORECAST statement. The same analysis is performed on the other three data sets, but is not shown here.

Output 30.8.1 resembles Table 1 in Kohn and Ansley (1986). This table is generated by merging the parameter estimates from the four analyses. Only the moving average parameter estimates and their standard errors are reported. The columns EST1 and STD1 correspond to the estimates for mylib.Data1. The parameter estimates and their standard errors for other three data sets are similarly named. Note that the parameter estimates closely match the parameter estimates in the article. However, their standard errors differ slightly. This difference could be the result of different ways of computing the Hessian at the optimum. The white-noise error variance estimates are not reported here, but they agree quite closely with those in the article.

Output 30.8.1 Data Sets 1–4: Parameter Estimates and Standard Errors

parameter	est1	std1	est2	std2	est3	std3	est4	std4
MA_1	0.402	0.090	0.457	0.121	0.408	0.092	0.431	0.091
SMA_1	0.557	0.073	0.758	0.236	0.566	0.075	0.573	0.074

Output 30.8.2 resembles Table 2 in Kohn and Ansley (1986). It contains forecasts and their standard errors for the four data sets. The numbers are very close to those in the article.

Output 30.8.2 Data Sets 1–4: Forecasts and Standard Errors

DATE	for1	std1	for2	std2	for3	std3	for4	std4
JAN61	6.110	0.037	6.084	0.052	6.110	0.037	6.111	0.037
FEB61	6.054	0.043	6.091	0.058	6.054	0.043	6.055	0.043
MAR61	6.172	0.048	6.247	0.063	6.173	0.048	6.174	0.048
APR61	6.199	0.053	6.205	0.068	6.199	0.053	6.200	0.052
MAY61	6.233	0.057	6.199	0.072	6.232	0.058	6.233	0.056
JUN61	6.369	0.061	6.308	0.076	6.367	0.062	6.368	0.060
JUL61	6.507	0.065	6.409	0.079	6.497	0.067	.	.
AUG61	6.503	0.069	6.414	0.082	6.503	0.069	6.503	0.067
SEP61	6.325	0.072	6.299	0.085	6.325	0.072	6.326	0.071
OCT61	6.209	0.075	6.174	0.087	6.209	0.076	6.209	0.074
NOV61	6.063	0.079	6.043	0.089	6.064	0.079	6.064	0.077
DEC61	6.168	0.082	6.174	0.086	6.168	0.082	6.169	0.080

Output 30.8.3 is based on Data2. It resembles Table 3 in Kohn and Ansley (1986). The columns S_SERIES and VS_SERIES in the **OUTFOR=** data set contain the interpolated values of logair and their variances. The estimate column in **Output 30.8.3** reports interpolated values (which are the same as S_SERIES), and the std column reports their standard errors (which are computed as square root of VS_SERIES) for January–November 1957. The actual logair values for these months, which are missing in Data2, are also provided for comparison. The numbers are very close to those in the article.

Output 30.8.3 Data Set 2: Interpolated Values and Standard Errors

DATE	logair	estimate	std
JAN57	5.753	5.733	0.045
FEB57	5.707	5.738	0.049
MAR57	5.875	5.893	0.052
APR57	5.852	5.850	0.054
MAY57	5.872	5.843	0.055
JUN57	6.045	5.951	0.055
JUL57	6.142	6.051	0.055
AUG57	6.146	6.055	0.054
SEP57	6.001	5.938	0.052
OCT57	5.849	5.812	0.049
NOV57	5.720	5.680	0.045

Output 30.8.4 resembles Table 4 in Kohn and Ansley (1986). These numbers are based on Data3, and they also are very close to those in the article.

Output 30.8.4 Data Set 3: Interpolated Values and Standard Errors

DATE	logair	estimate	std
JUL49	4.997	5.013	0.031
JUN57	6.045	6.024	0.030
JUL57	6.142	6.147	0.031
AUG57	6.146	6.148	0.030
JUL60	6.433	6.409	0.031

Output 30.8.5 resembles Table 5 in Kohn and Ansley (1986). As before, the numbers are very close to those in the article.

Output 30.8.5 Data Set 4: Interpolated Values and Standard Errors

DATE	logair	estimate	std
JUN57	6.045	6.023	0.030
AUG57	6.146	6.147	0.030

The similarity between the outputs in this example and the results shown in Kohn and Ansley (1986) demonstrate that PROC UCM can be effectively used for nonstationary ARIMA models with missing data.

Example 30.9: Extracting a Business Cycle

The data set (not shown) `gdp` in this example has two variables: `date` dates the observations, and `lgdp` contains the quarterly readings of the US real GDP (in log scale). Pelagatti (2015, Example 3.3, Example 8.2) uses this quarterly time series (`lgdp`) to illustrate how you can adjust the smoothness of the estimated cycle by changing the order of the cycle in a trend-cycle decomposition,

$$\text{lgdp}_t = \mu_t + \psi_t + \epsilon_t$$

where μ_t is an integrated random walk trend, ψ_t is a cycle component, and ϵ_t is an irregular component.

```
/* Load data into mylib */
data mylib.gdp;
  set gdp;
run;
```

The following statements fit the model $\text{lgdp}_t = \mu_t + \psi_t + \epsilon_t$, where the cycle component has an order of 1 (default):

```
proc ucm data= mylib.gdp;
  where year(date) >= 1970;
  id date interval=quarter;
  model lgdp;
  irregular;
  level variance=0 noest plot=smooth;
  slope;
  cycle plot=smooth;
  estimate plot=panel;
  forecast plot=decomp outfor= mylib.for1;
run;
```

The following statements fit the same model, except that the cycle order is 2. Similarly, a model with a cycle order of 4 is also fit (not shown).

```

proc ucm data=mylib.gdp;
  where year(date) >= 1970;
  id date interval=quarter;
  model lgdp;
  irregular;
  level variance=0 noest plot=smooth;
  slope;
  cycle order=2 plot=smooth;
  estimate plot=panel;
  forecast plot=decomp outfor= mylib.for2;
run;

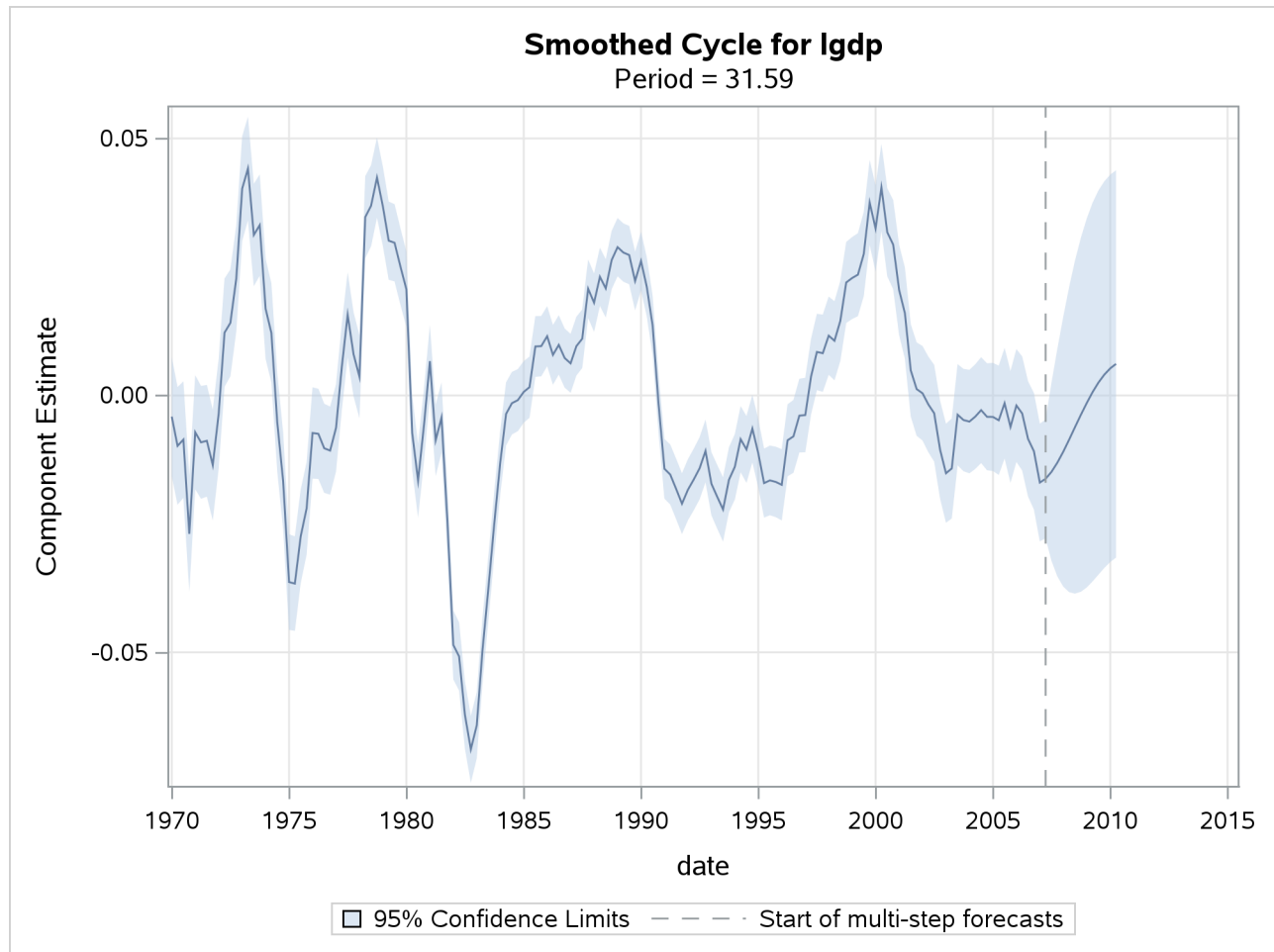
```

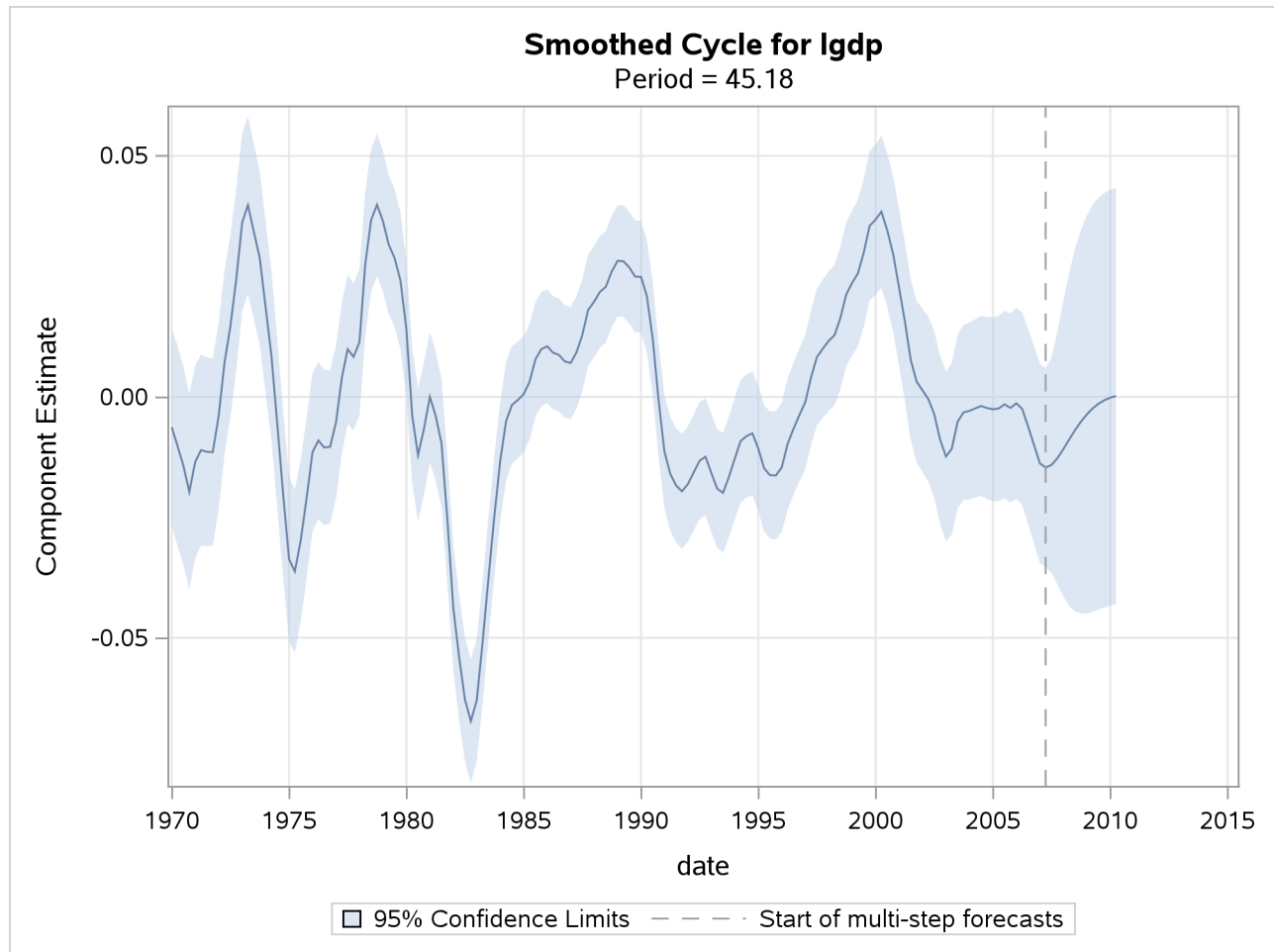
[Output 30.9.1](#) summarizes the features of the estimated cycles of different orders. The estimated periods of the first-order and second-order cycles, 31.59 and 45.18, are reasonable. However, the period of the fourth-order cycle seems quite unreasonable. Fortunately, Pelagatti (2015, Example 8.2) mentions that cycles of order 3 or higher are rarely needed when you are working with real economic series. Although they are not the same, the parameter estimates that the UCM procedure produces are reasonably close to those reported in Pelagatti (2015, Example 8.2).

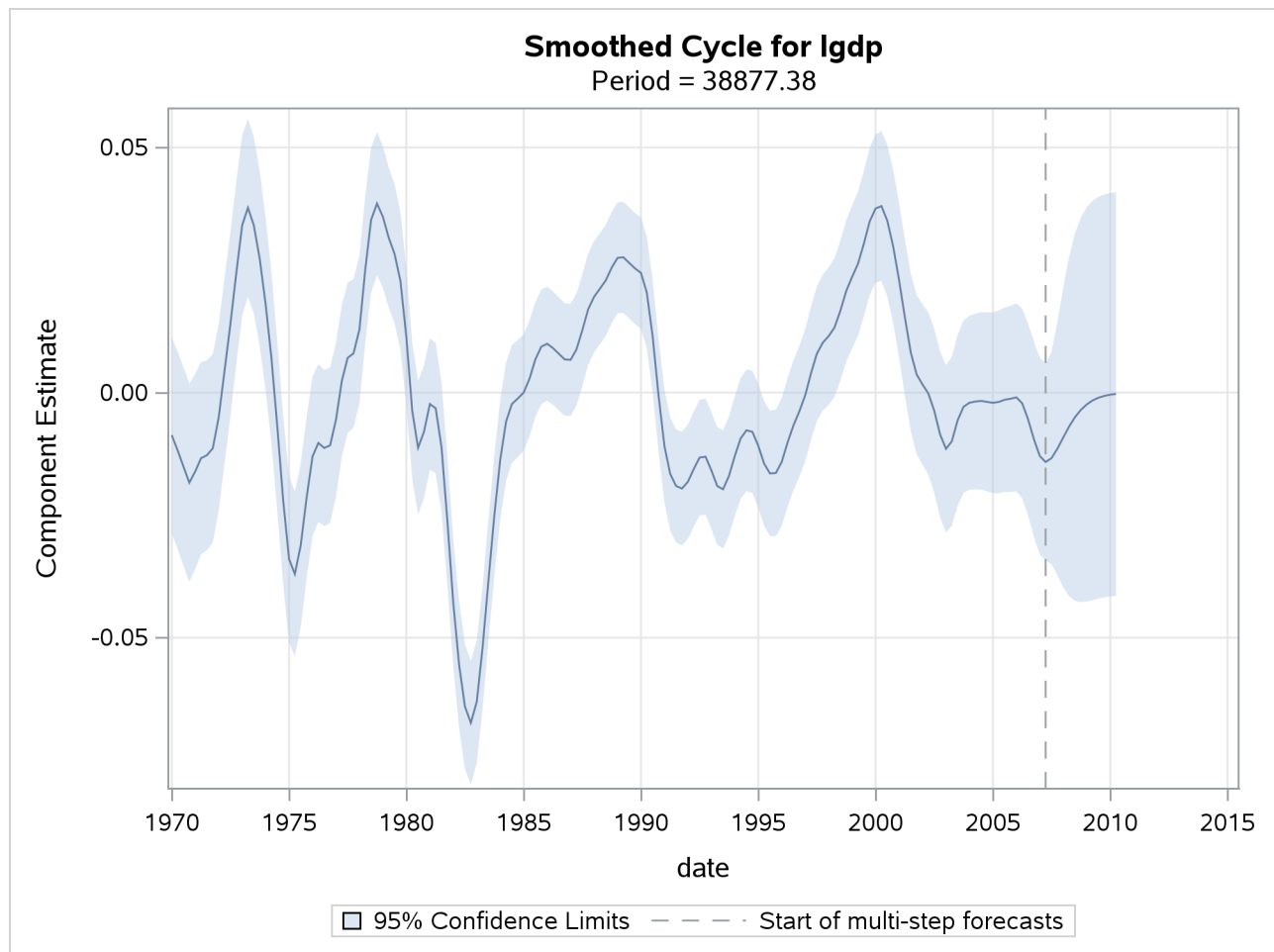
Output 30.9.1 Cycles of Orders 1, 2, and 4: Summary

order	period	Frequency	Rho	ErrorVar
1	31.59294	0.19888	0.94371	0.00004873
2	45.18257	0.13906	0.76177	0.00000956
4	38877	0.00016162	0.52055	0.00000856

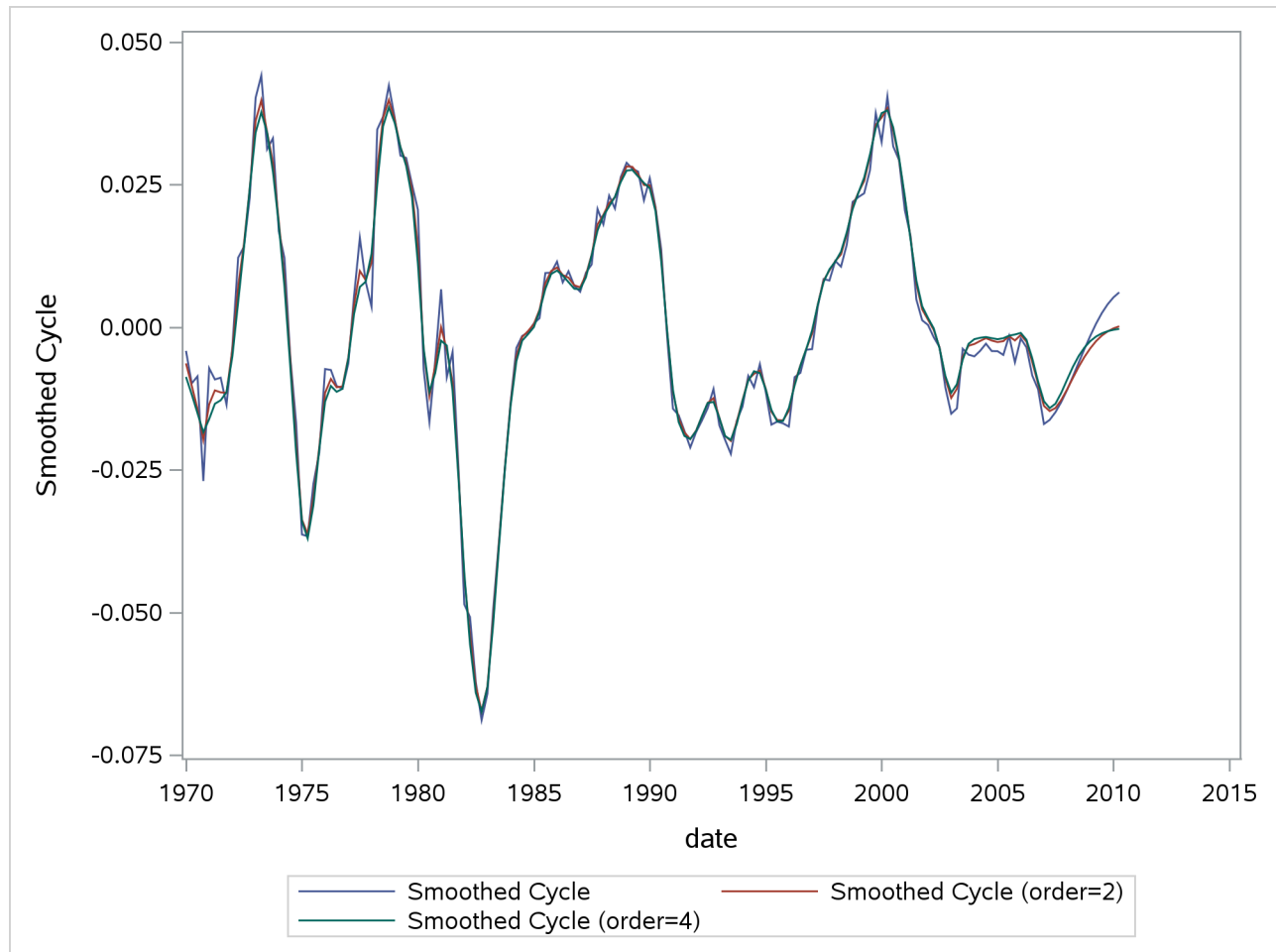
[Output 30.9.2](#) shows the plot of the first-order cycle, [Output 30.9.3](#) shows the plot of the second-order cycle, and [Output 30.9.4](#) shows the plot of the fourth-order cycle. You can see that although the overall form of the estimated cycle remains the same, the smoothness of the plot of the estimated cycle increases with the order.

Output 30.9.2 Estimated Cycle: Order = 1

Output 30.9.3 Estimated Cycle: Order = 2

Output 30.9.4 Estimated Cycle: Order = 4

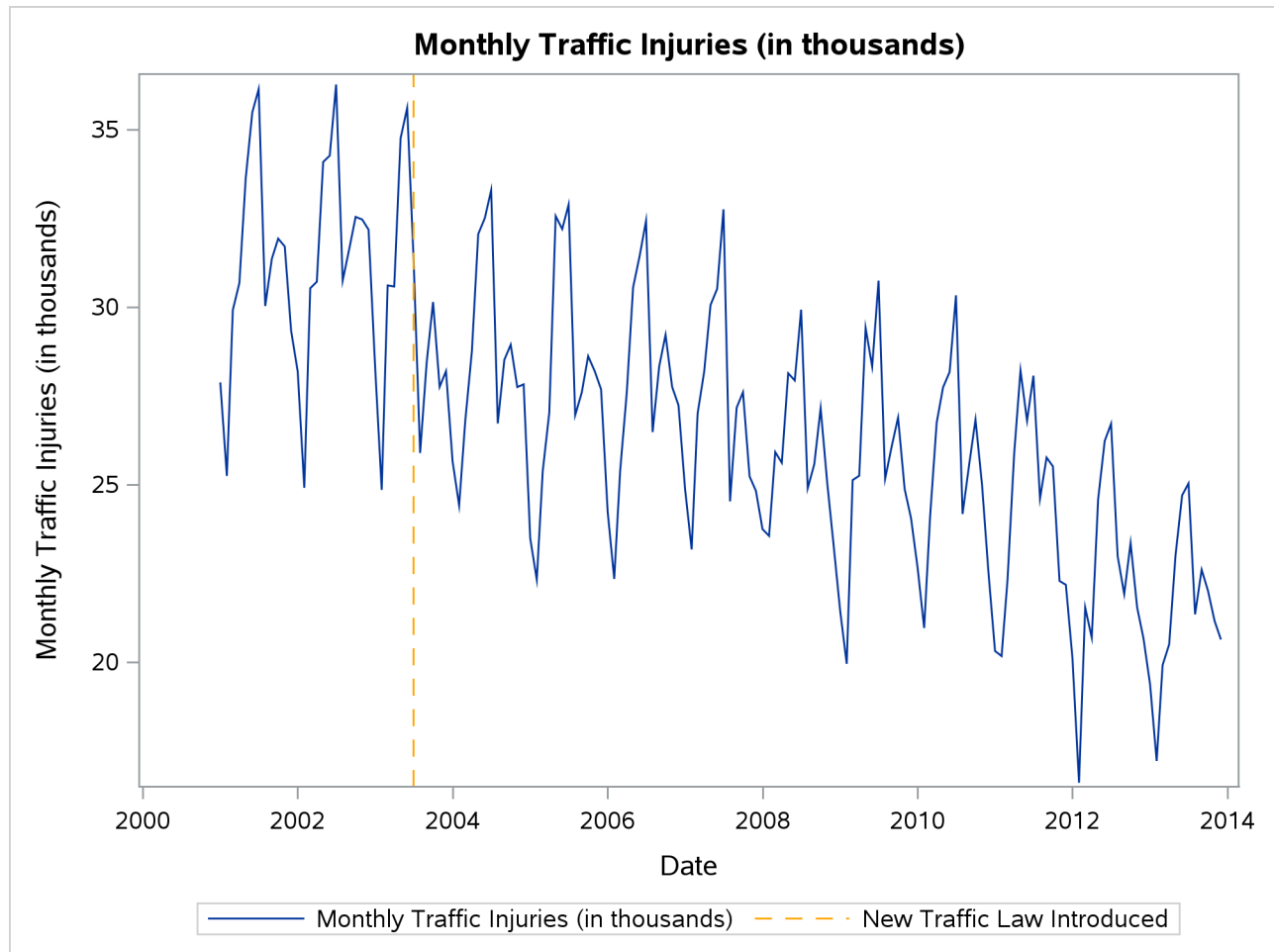
Output 30.9.5 shows the three cycle estimates in the same plot. It shows that the estimates do not differ very much.

Output 30.9.5 Estimated Cycles of Orders 1, 2, and 4

Example 30.10: A Transfer-Function Model for the Italian Traffic Accident Data

This example is based on a case study described in Pelagatti (2015, chap. 9, sec. 1). In July 2003, Italy introduced a new traffic monitoring system with the aim of improving traffic safety. The case study tried to answer the question, “Was the monitoring system effective in reducing the number of traffic injuries?” The time series plot in [Output 30.10.1](#) shows monthly traffic injuries for the span of January 2001 to December 2013. Visual inspection of the plot clearly shows that the series is seasonal and has an overall downward trend, which appears to be more pronounced after the intervention.

Output 30.10.1 Monthly Traffic Injuries in Italy



Pelagatti (2015, chap. 9, sec. 1) suggests the following model for this series:

$$y_t = \mu_t + \psi_t + \text{shift03} \beta + \xi_t + \epsilon_t$$

Various terms in the right-hand side of this model are explained as follows:

- μ_t is the trend component, which is modeled as an integrated random walk.
- ψ_t is the trigonometric seasonal component, which accounts for the monthly seasonality.
- The effect of the introduction of the monitoring system is modeled using two terms:
 - One term captures a permanent shift, which is a regression effect that is associated with the dummy regressor `shift03`. The value of this regressor is 0 before July 2003 and 1 thereafter.
 - The other term captures a transient effect that rapidly decays to 0. The transient effect ξ_t is a transfer-function effect

$$\xi_t = \frac{\gamma_0 \text{pulse03}_t}{(1 - \delta B)}$$

where `pulse03` is a dummy regressor whose value is 1 at July 2003 and 0 otherwise. In this example, the value of the transfer function ξ_t is clearly 0 before July 2003.

- ϵ_t is the simple irregular component.

The following statements show how to fit this model to the data. The LIKE=MARGINAL option in the ESTIMATE statement causes the parameter estimation to be based on marginal likelihood rather than on diffuse likelihood, which is the default. Since the parameter vector of this model contains δ (the denominator coefficient of the transfer function), the parameter estimations that are based on marginal likelihood and diffuse likelihood can lead to different results. In this example, the results turn out to be similar; however, this is not necessarily the case in general. Generally, parameter estimation that is based on marginal likelihood is the preferred choice in such cases.

```
/* Load data into mylib */
data mylib.italy;
    set italy;
run;

proc ucm data=mylib.italy;
    id date interval=month;
    model Injured = shift03;
    irregular;
    level variance=0 noest;
    slope;
    season length=12 type=trig;
    tf pulse03 den=1 tfstart=0 plot=smooth;
    estimate plot=(panel residual) like=marginal;
    forecast plot=decomp;
run;
```

Output 30.10.3 shows the parameter estimates. It shows that soon after the introduction of the monitoring system in July 2003, the accident level decreased by about 5.22 thousand ($\hat{\beta} + \hat{\gamma}_0 = -(2.48 + 2.74)$). However, the permanent decrease was only about 2.48 thousand ($\hat{\beta} = -2.48$). The estimate of the decay parameter of the transfer function, δ , is 0.587.

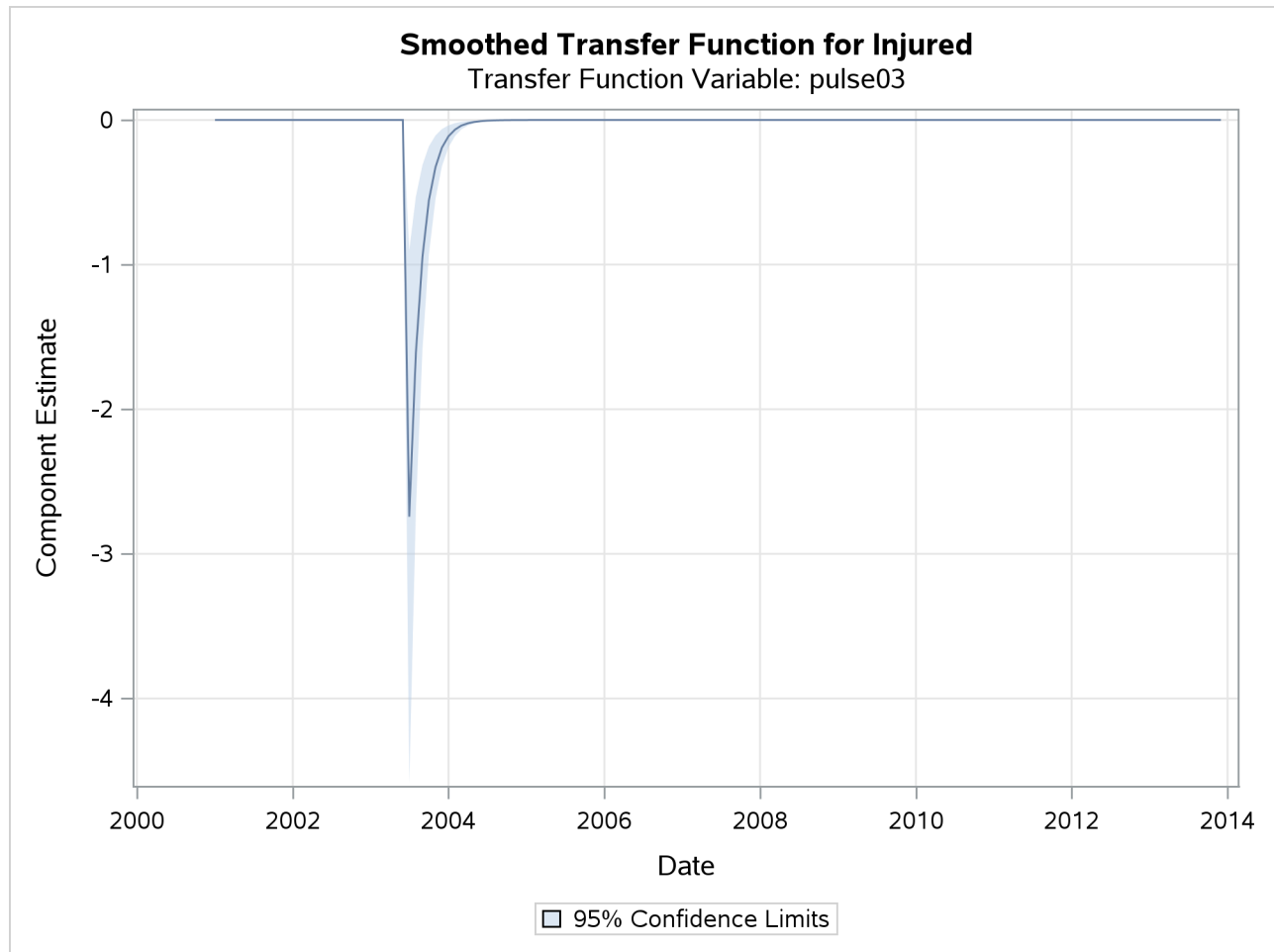
Output 30.10.2 Estimates of the Model Parameters

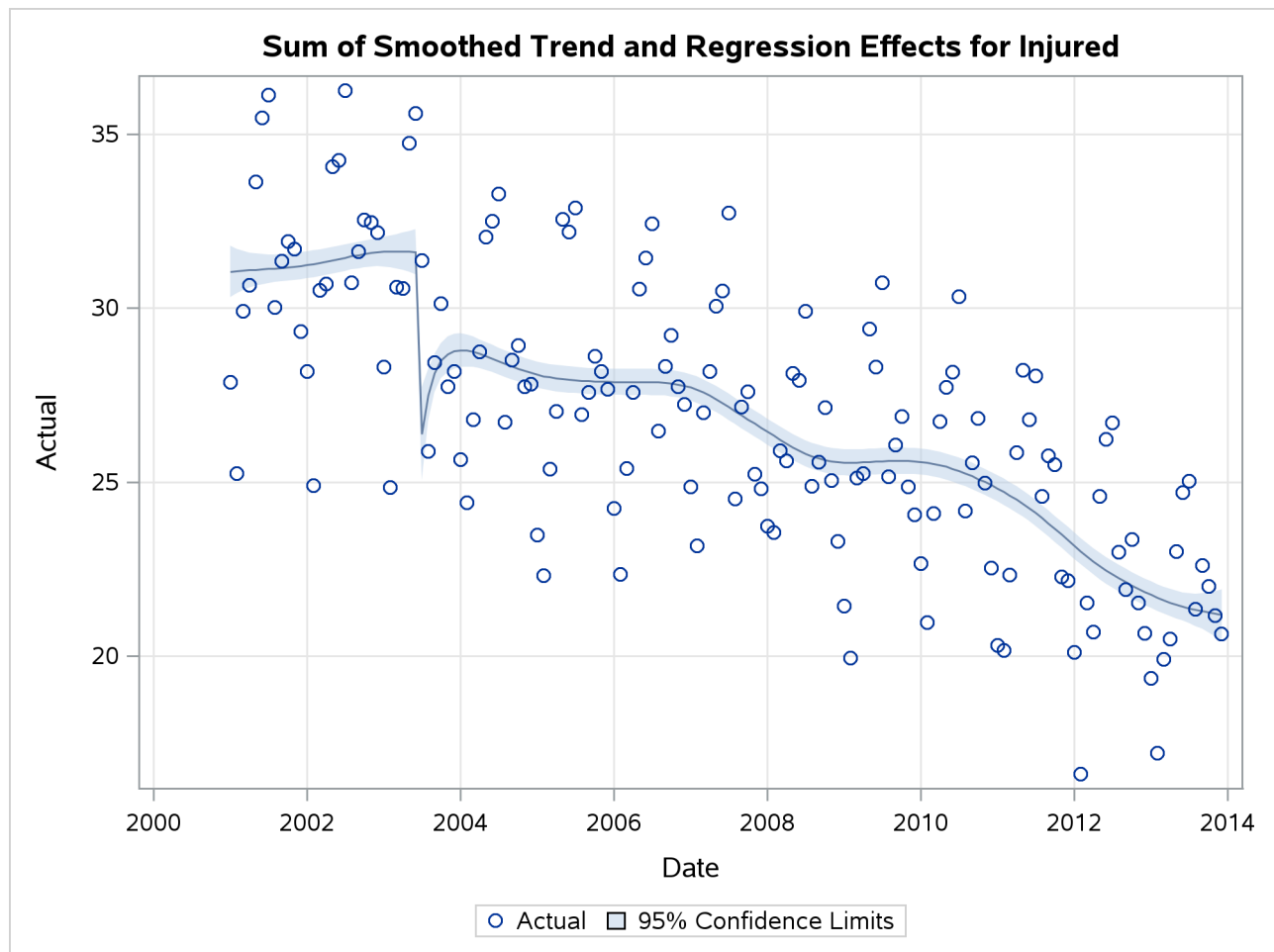
The UCM Procedure

Final Estimates of the Free Parameters					
Component	Parameter	Estimate	Approx Std Error	t Value	Approx Pr > t
Irregular	Error Variance	0.55447	0.0923	6.01	<.0001
Slope	Error Variance	0.00064586	0.0005	1.43	0.1526
Season	Error Variance	0.00068803	0.0005	1.33	0.1849
shift03	Coefficient	-2.47939	0.7093	-3.50	0.0005
pulse03	Coefficient	-2.74316	0.9385	-2.92	0.0035
pulse03	DEN_1	0.58714	0.1780	3.30	0.0010

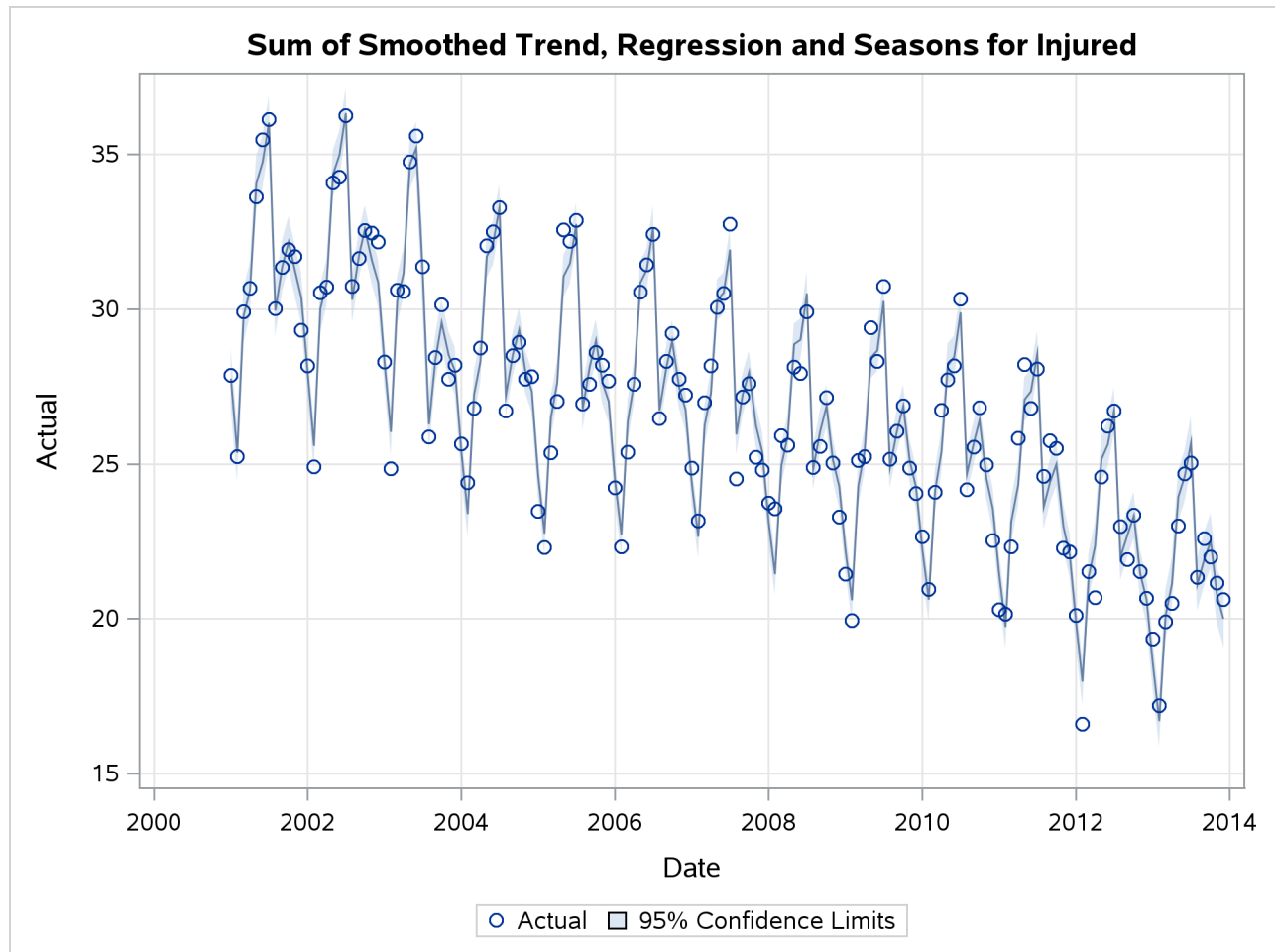
Output 30.10.3 shows the plot of smoothed estimate of the transfer function ξ_t , and Output 30.10.4 shows the plot of the estimate of the trend plus the total effect of the July 2003 intervention.

Output 30.10.3 Decaying Part of the July 2003 Intervention Effect (Smoothed Estimate of ξ_t)



Output 30.10.4 Smoothed Estimate of $\mu_t + \text{shift03 } \beta + \xi_t$ 

Finally, the [Output 30.10.5](#) shows the plot of the overall model fit.

Output 30.10.5 Sum of All Model Terms Except the Irregular

References

- Akaike, H. (1974). "A New Look at the Statistical Model Identification." *IEEE Transactions on Automatic Control* AC-19:716–723.
- Anderson, T. W. (1971). *The Statistical Analysis of Time Series*. New York: John Wiley & Sons.
- Bloomfield, P. (2000). *Fourier Analysis of Time Series*. 2nd ed. New York: John Wiley & Sons.
- Box, G. E. P., and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*. Rev. ed. San Francisco: Holden-Day.
- Bozdogan, H. (1987). "Model Selection and Akaike's Information Criterion (AIC): The General Theory and Its Analytical Extensions." *Psychometrika* 52:345–370.
- Brockwell, P. J., and Davis, R. A. (1991). *Time Series: Theory and Methods*. 2nd ed. New York: Springer-Verlag.

- Burnham, K. P., and Anderson, D. R. (1998). *Model Selection and Inference: A Practical Information-Theoretic Approach*. New York: Springer-Verlag.
- Cobb, G. W. (1978). "The Problem of the Nile: Conditional Solution to a Change Point Problem." *Biometrika* 65:243–251.
- De Jong, P., and Chu-Chun-Lin, S. (2003). "Smoothing with an Unknown Initial Condition." *Journal of Time Series Analysis* 24:141–148.
- De Jong, P., and Penzer, J. (1998). "Diagnosing Shocks in Time Series." *Journal of the American Statistical Association* 93:796–806.
- Durbin, J., and Koopman, S. J. (2012). *Time Series Analysis by State Space Methods*. 2nd ed. Oxford: Oxford University Press.
- Hannan, E. J., and Quinn, B. G. (1979). "The Determination of the Order of an Autoregression." *Journal of the Royal Statistical Society, Series B* 41:190–195.
- Harvey, A. C. (1989). *Forecasting, Structural Time Series Models, and the Kalman Filter*. Cambridge: Cambridge University Press.
- Harvey, A. C. (2001). "Testing in Unobserved Components Models." *Journal of Forecasting* 20:1–19.
- Hodrick, R. J., and Prescott, E. C. (1997). "Postwar U.S. Business Cycles: An Empirical Investigation." *Journal of Money, Credit, and Banking* 29:1–16.
- Hurvich, C. M., and Tsai, C.-L. (1989). "Regression and Time Series Model Selection in Small Samples." *Biometrika* 76:297–307.
- Jones, R. H. (1980). "Maximum Likelihood Fitting of ARMA Models to Time Series with Missing Observations." *Technometrics* 22:389–396.
- Kohn, R., and Ansley, C. F. (1986). "Estimation, Prediction, and Interpolation for ARIMA Models with Missing Data." *Journal of the American Statistical Association* 81:751–761.
- Pelagatti, M. M. (2015). *Time Series Modelling with Unobserved Components*. Boca Raton, FL: CRC Press.
- Schwarz, G. (1978). "Estimating the Dimension of a Model." *Annals of Statistics* 6:461–464.
- Trimbur, T. M. (2005). "Properties of Higher Order Stochastic Cycles." *Journal of Time Series Analysis* 27:1–17.
- West, M., and Harrison, J. (1999). *Bayesian Forecasting and Dynamic Models*. 2nd ed. New York: Springer-Verlag.

Chapter 31

Time Series Analysis Package

Contents

Overview	2242
Using CAS Sessions and CAS Engine Librefs	2242
CORR Class	2243
CORR Object	2243
OUTCORR Object	2247
CROSSCORR Class	2251
CROSSCORR Object	2251
OUTCROSSCORR Object	2255
DECOMP Class	2258
DECOMP Object	2258
OUTDECOMP Object	2262
OUTDECOMP2 Object	2264
Details	2266
DPF Class	2268
DPF Object	2268
OUTDPE Object	2271
OUTDPROB Object	2273
Example	2275
FREQ Class	2276
FREQ Object	2276
OUTFREQ Object	2279
Example	2281
SPECTRA Class	2281
SPECTRA Object	2281
OUTSPECTRA Object	2286
Details	2288
STATIONARITY Class	2292
STATIONARITY Object	2292
OUTSTATIONARITY Object	2295
TSAOUTLIER Class	2298
TSAOUTLIER Object	2298
TSAOUTOUTLIER Object	2300
Details	2302
TSA Object	2302
Functional Summary	2302

ACCUMULATE Method	2303
ACCUMULATE2 Method	2305
ACF Method	2308
ARMAORDERS Method	2309
CCF Method	2311
IACF Method	2313
INTERMITTENCYTEST Method	2314
MKTEST Method	2316
MOVINGSUMMARY Method	2318
PACF Method	2321
SCALE Method	2323
SEASONALDECOMP Method	2324
SEASONALINDICES Method	2326
SEASONTTEST Method	2328
SIMILARITY Method	2330
STATIONARITYTEST Method	2332
TRANSFORM Method	2334
UNBIASEDNESS Method	2336
WHITENOISE Method	2337
References	2338

Overview

The time series analysis (TSA) package contains a set of time series analysis functions that can be used as part of the programming statements in the TSMODEL procedure. This package provides a flexible way to analyze time series within the procedure.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

NOTE: The mylib.air data table that is used in the examples refers to SasHELP.AIR data.

CORR Class

CORR Object

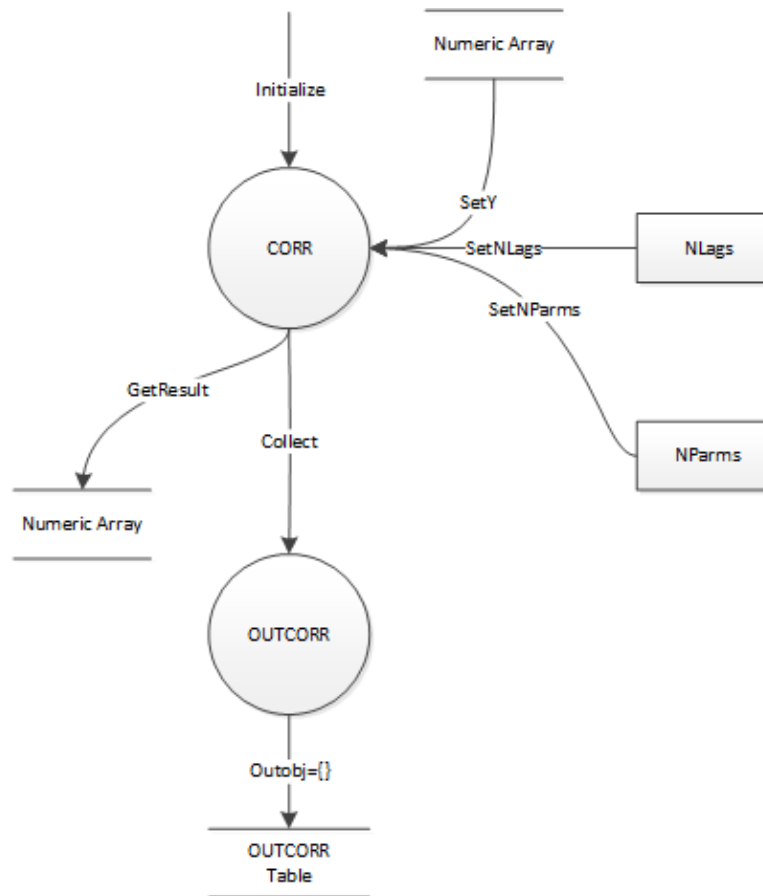
The CORR class object functions perform the autocorrelation and autocovariance analysis of a time series array and output all the values and results.

Table 31.1 summarizes the methods that are associated with the CORR object.

Table 31.1 Methods of the CORR Object

Method	Description
GetResult	Get the analysis result
Initialize	Initialize the CORR object
Run	Run the analysis
SetNLags	Set the number of lags
SetNParms	Set the number of parameters
SetY	Set the input time series

Figure 31.1 diagrams the methods of the CORR object.

Figure 31.1 CORR Data Flow

CORR Synopsis

DECLARE OBJECT *obj* (CORR) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetNLags (Value) ;
rc=obj.SetNParms (Value) ;
rc=obj.Run () ;
rc=obj.GetResult ('Name',OutputArray) ;

```

CORR Methods

CORR.GetResult Method

```
rc=obj.GetResult ('Name',OutputArray) ;
```

Outputs the selected results to an array.

Input Arguments You must specify the following input arguments:

<i>Name</i>	specifies a string that identifies the result series to return. You can specify only one of the following values within single quotation marks:
<i>acov</i>	returns an array of covariance estimates, with <i>nlag</i> + 1 entries.
<i>acf</i>	returns an array of autocorrelation estimates, with <i>nlag</i> + 1 entries.
<i>acfstd</i>	returns an array of standard errors, with <i>nlag</i> + 1 entries.
<i>acf2std</i>	returns an array of twice standard errors, with <i>nlag</i> + 1 entries.
<i>acfnorm</i>	returns an array of normalized autocorrelation, with <i>nlag</i> + 1 entries.
<i>acfprob</i>	returns an array of autocorrelation probabilities, with <i>nlag</i> + 1 entries.
<i>acflprob</i>	returns an array of autocorrelation log probabilities, with <i>nlag</i> + 1 entries.
<i>pacf</i>	returns an array of partial autocorrelation estimates, with <i>nlag</i> + 1 entries.
<i>pacfstd</i>	returns an array of partial autocorrelation standard errors, with <i>nlag</i> + 1 entries.
<i>pacf2std</i>	returns an array of twice standard errors, with <i>nlag</i> + 1 entries.
<i>pacfnorm</i>	returns an array of normalized partial autocorrelation, with <i>nlag</i> + 1 entries.
<i>pacfprob</i>	returns an array of partial autocorrelation probabilities, with <i>nlag</i> + 1 entries.
<i>pacflprob</i>	returns an array of partial autocorrelation log probabilities, with <i>nlag</i> + 1 entries.
<i>iacf</i>	returns an array of inverse autocorrelation estimates, with <i>nlag</i> + 1 entries.
<i>iacfstd</i>	returns an array of inverse autocorrelation standard errors, with <i>nlag</i> + 1 entries.
<i>iacf2std</i>	returns an array of twice standard errors, with <i>nlag</i> + 1 entries.
<i>iacfnorm</i>	returns an array of normalized inverse autocorrelation, with <i>nlag</i> + 1 entries.
<i>iacfprob</i>	returns an array of inverse autocorrelation probabilities, with <i>nlag</i> + 1 entries.
<i>iacflprob</i>	returns an array of inverse autocorrelation log probabilities, with <i>nlag</i> + 1 entries.
<i>wn</i>	returns an array of Ljung-Box white noise tests, with <i>nlag</i> + 1 entries.
<i>wnprob</i>	returns white noise probabilities.
<i>wnlprob</i>	returns white noise log probabilities.

OutputArray specifies a numeric array to receive the result series.

CORR.Initialize Method

rc=obj.Initialize () ;

Initializes a CORR object to an empty state. This method must be called before the time series arrays and other attributes are specified for the CORR object.

Arguments There are no arguments associated with this method.

CORR.Run Method

```
rc=obj.Run () ;
```

Runs the CORR object to perform the autocorrelation and autocovariance analysis by using the time series array *YSeries* that has been specified for it. Upon successful completion, various results can be extracted from the CORR object.

CORR.SetNLags Method

```
rc=obj.SetNLags (Value) ;
```

Sets a named option for the CORR object.

Input Arguments You must specify the following input argument:

Value specifies the number of lags to use in the calculation. The default value is 24.

CORR.SetNParms Method

```
rc=obj.SetNParms (Value) ;
```

Sets a named option for the CORR object.

Input Arguments You must specify the following input argument:

Value specifies the degrees of freedom associated with the Ljung-Box statistics. The default value is 0.

CORR.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the time series array (*YSeries*) for the CORR object.

Input Arguments You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the CORR object.

Arguments There are no arguments associated with this method.

OUTCORR Object

The OUTCORR object collects output from a CORR object.

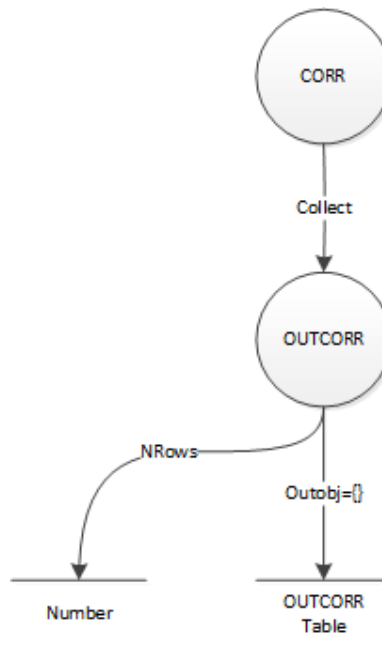
Table 31.2 summarizes the methods that are associated with the OUTCORR object.

Table 31.2 Methods of the OUTCORR Object

Method	Description
Collect	Collect the results of a CORR object
nrows	Get the current row count from the OUTCORR object

Figure 31.2 diagrams the methods of the OUTCORR object.

Figure 31.2 OUTCORR Data Flow



OUTCORR Methods

OUTCORR.Collect Method

```
rc=obj.Collect (CORRObj) ;
```

Collects the results from a CORR object and saves them to a table.

Arguments You must specify the following input argument:

CORRObj is a character string that specifies the name of the CORR object to use as the source of the autocorrelation and autocovariance analysis results.

OUTCORR.nrows Attribute

`nrows=obj.nrows ()` ;

Gets the current row count from the OUTCORR object.

Arguments There are no arguments associated with this method.

Output Table Schema

The collector object, OUTCORR, collects the results of a CORR object.

OUTCORR collects the results of a CORR object. The output table contains the columns that are shown in Table 31.3.

Table 31.3 OUTCORR Output Table Contents

Column Name	Column Content	Statistics Used to Calculate Content
NAME	Name of target variable	N/A
LAG	Array of time lags	$h \in \{0, \dots, H\}$
N	Number of variance products	N_h , the number of observed products at lag h , ignoring missing values
MU	Mean value	Mean value
ACOV	Array of covariance estimates, with $nlag + 1$ entries	$\hat{\gamma}(h) = \frac{1}{T} \sum_{t=h+1}^T (y_t - \bar{y})(y_{t-h} - \bar{y})$, or $\hat{\gamma}(h) = \frac{1}{N_h} \sum_{t=h+1}^T (y_t - \bar{y})(y_{t-h} - \bar{y})$ when embedded missing values are present
ACF	Array of autocorrelation estimates, with $nlag + 1$ entries	$\hat{\rho}(h) = \hat{\gamma}(h) / \hat{\gamma}(0)$
ACFSTD	Array of standard errors, with $nlag + 1$ entries	$\text{Std}(\hat{\rho}(h)) = \sqrt{\frac{1}{T} \left(1 + 2 \sum_{j=1}^{h-1} \hat{\rho}(j)^2 \right)}$
ACF2STD	Array of twice standard errors, with $nlag + 1$ entries	$\text{Flag}(\hat{\rho}(h)) = \begin{cases} 1 & \hat{\rho}(h) > 2\text{Std}(\hat{\rho}(h)) \\ 0 & -2\text{Std}(\hat{\rho}(h)) < \hat{\rho}(h) < 2\text{Std}(\hat{\rho}(h)) \\ -1 & \hat{\rho}(h) < -2\text{Std}(\hat{\rho}(h)) \end{cases}$
ACFNORM	Array of normalized autocorrelation, with $nlag + 1$ entries	$\text{Norm}(\hat{\rho}(h)) = \hat{\rho}(h) / \text{Std}(\hat{\rho}(h))$

Table 31.3 *continued*

Column Name	Column Content	Statistics Used to Calculate Content
ACFPROB	Array of autocorrelation probabilities, with $nlag + 1$ entries	$\text{Prob}(\hat{\rho}(h)) = 2 (1 - \Phi(\text{Norm}(\hat{\rho}(h))))$
ACFLPROB	Array of autocorrelation log probabilities, with $nlag + 1$ entries	$\text{LogProb}(\hat{\rho}(h)) = -\log_{10}(\text{Prob}(\hat{\rho}(h)))$
PACF	Array of partial autocorrelation estimates, with $nlag + 1$ entries	$\hat{\varphi}(h) = \Gamma_{(0,h-1)}\{\gamma_j\}_{j=1}^h$
PACFSTD	Array of partial autocorrelation standard errors, with $nlag + 1$ entries	$\text{Std}(\hat{\varphi}(h)) = 1/\sqrt{N_0}$
PACF2STD	Array of twice standard errors, with $nlag + 1$ entries	$\text{Flag}(\hat{\varphi}(h)) = \begin{cases} 1 & \hat{\varphi}(h) > 2\text{Std}(\hat{\varphi}(h)) \\ 0 & -2\text{Std}(\hat{\varphi}(h)) < \hat{\varphi}(h) < 2\text{Std}(\hat{\varphi}(h)) \\ -1 & \hat{\varphi}(h) < -2\text{Std}(\hat{\varphi}(h)) \end{cases}$
PACFNORM	Array of normalized partial autocorrelation, with $nlag + 1$ entries	$\text{Norm}(\hat{\rho}(h)) = \hat{\rho}(h)/\text{Std}(\hat{\rho}(h))$
PACFPROB	Array of partial autocorrelation probabilities, with $nlag + 1$ entries	$\text{Prob}(\hat{\varphi}(h)) = 2 (1 - \Phi(\text{Norm}(\hat{\varphi}(h))))$
PACFLPROB	Array of partial autocorrelation log probabilities, with $nlag + 1$ entries	$\text{LogProb}(\hat{\varphi}(h)) = -\log_{10}(\text{Prob}(\hat{\varphi}(h)))$
IACF	Array of inverse autocorrelation estimates, with $nlag + 1$ entries	$\hat{\theta}(h)$
IACFSTD	Array of inverse autocorrelation standard errors, with $nlag + 1$ entries	$\text{Std}(\hat{\theta}(h)) = 1/\sqrt{N_0}$

Table 31.3 *continued*

Column Name	Column Content	Statistics Used to Calculate Content
IACF2STD	Array of twice standard errors, with $nlag + 1$ entries	$\text{Flag}(\hat{\theta}(h)) = \begin{cases} 1 & \hat{\theta}(h) > 2\text{Std}(\hat{\theta}(h)) \\ 0 & -2\text{Std}(\hat{\theta}(h)) < \hat{\theta}(h) < 2\text{Std}(\hat{\theta}(h)) \\ -1 & \hat{\theta}(h) < -2\text{Std}(\hat{\theta}(h)) \end{cases}$
IACFNORM	Array of normalized inverse autocorrelation, with $nlag + 1$ entries	$\text{Norm}(\hat{\theta}(h)) = \hat{\theta}(h)/\text{Std}(\hat{\theta}(h))$
IACFPROB	Array of inverse autocorrelation probabilities, with $nlag + 1$ entries	$\text{Prob}(\hat{\theta}(h)) = 2 \left(1 - \Phi \left(\text{Norm}(\hat{\theta}(h)) \right) \right)$
IACFLPROB	Array of inverse autocorrelation log probabilities, with $nlag + 1$ entries	$\text{LogProb}(\hat{\varphi}(h)) = -\log_{10}(\text{Prob}(\hat{\varphi}(h)))$
WN	Array of Ljung-Box white noise tests, with $nlag + 1$ entries	$Q(h) = T(T+2) \sum_{j=1}^h \rho(j)^2 / (T-j)$
WNPROB	White noise probabilities	$\text{Prob}(Q(h)) = \chi_{\max(1, h-p)}(Q(h))$
WNLPROB	White noise log probabilities	$\text{LogProb}(Q(h)) = -\log_{10}(\text{Prob}(Q(h)))$

CROSSCORR Class

CROSSCORR Object

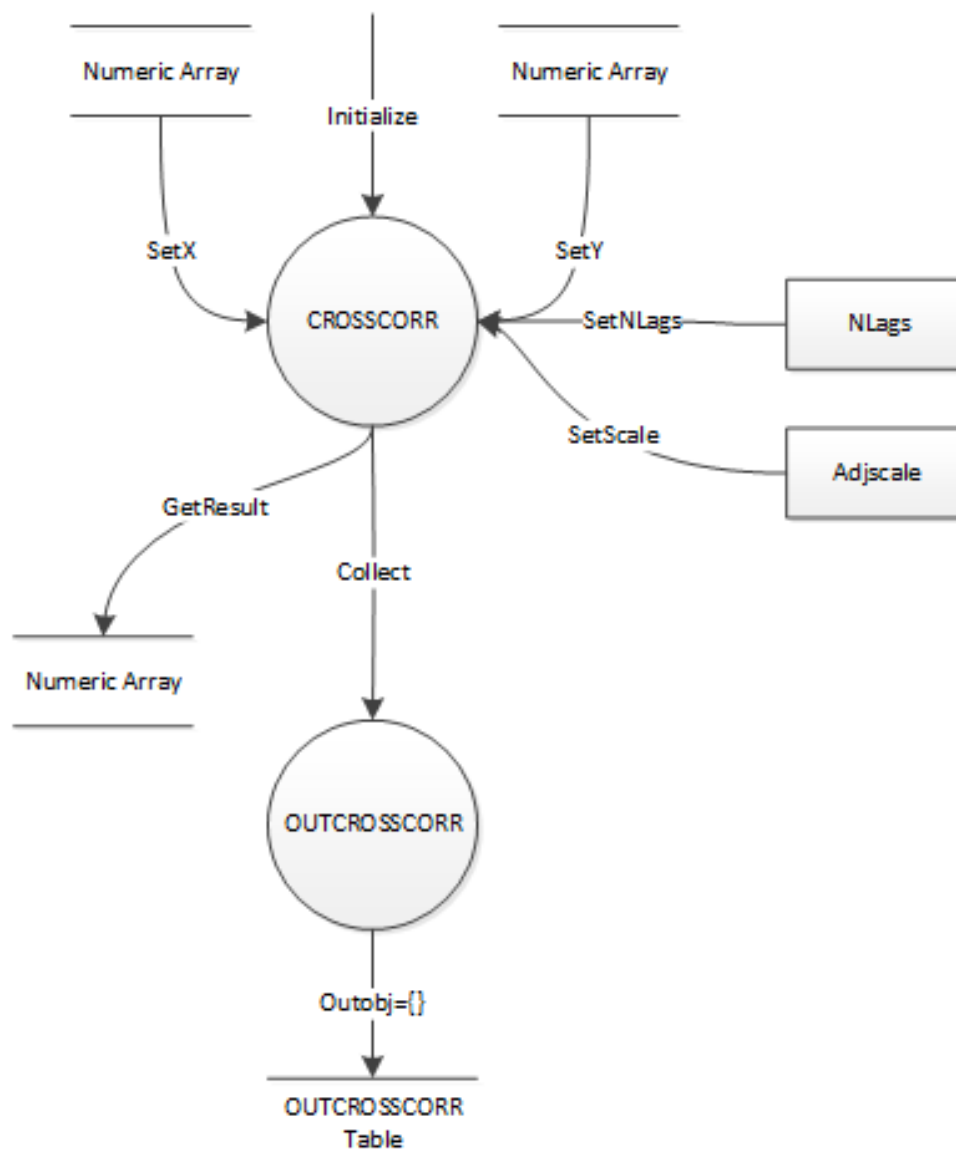
The CROSSCORR class object functions perform the cross-correlation and cross-covariance analysis of two time series arrays and output all the values and results.

Table 31.4 summarizes the methods that are associated with the CROSSCORR object.

Table 31.4 Methods of the CROSSCORR Object

Method	Description
GetResult	Get the analysis result
Initialize	Initialize the CROSSCORR object
Run	Run the analysis
SetNLags	Set the number of lags
SetScale	Set the normalization
SetX	Set the cross time series array
SetY	Set the primary time series array

Figure 31.3 diagrams the methods of the CROSSCORR object.

Figure 31.3 CROSSCORR Data Flow

CROSSCORR Synopsis

DECLARE OBJECT *obj* (CROSSCORR) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetX (XSeries) ;
rc=obj.SetNLags (NLags) ;
rc=obj.SetScale (Adjscale) ;
rc=obj.Run () ;
rc=obj.GetResult ('Name',OutputArray) ;

```

CROSSCORR Methods

CROSSCORR.GetResult Method

```
rc=obj.GetResult ('Name',OutputArray) ;
```

Outputs the selected results to an array.

Input Arguments You must specify the following input arguments:

<i>Name</i>	specifies a string that identifies the result series to return. You can specify the following values within single quotation marks:
<i>ccov</i>	returns an array of cross-covariance estimates, with $2 \times nlag + 1$ entries.
<i>ccf</i>	returns an array of cross-correlation estimates, with $2 \times nlag + 1$ entries.
<i>ccfst</i>	returns an array of standard errors, with $2 \times nlag + 1$ entries.
<i>ccf2std</i>	returns an array of double standard errors, with $2 \times nlag + 1$ entries.
<i>ccfnorm</i>	returns an array of normalized cross-correlation, with $2 \times nlag + 1$ entries.
<i>ccfprob</i>	returns an array of probabilities, with $2 \times nlag + 1$ entries.
<i>ccflprob</i>	returns an array of log probabilities, with $2 \times nlag + 1$ entries.

OutputArray specifies a numeric array to receive the result series.

CROSSCORR.Initialize Method

```
rc=obj.Initialize () ;
```

Initializes an CROSSCORR object to an empty state. This method must be called before the time series arrays and other attributes are specified for the CROSSCORR object.

Arguments There are no arguments associated with this method.

CROSSCORR.Run Method

```
rc=obj.Run () ;
```

Runs the CROSSCORR object to perform the cross-correlation and cross-covariance analysis by using the time series arrays *YSeries* and *XSeries* that have been specified for it. Upon successful completion, various results can be extracted from the CROSSCORR object.

Arguments There are no arguments associated with this method.

CROSSCORR.SetNLags Method

```
rc=obj.SetNLags (Nlags) ;
```

Sets a named option for the CROSSCORR object.

Input Arguments You must specify the following input argument:

Nlags specifies the number of lags to use in the calculation. The default value is 24.

CROSSCORR.SetScale Method

rc=obj.SetScale (Adjscale) ;

Specifies the normalization setting for the CROSSCORR object.

Input Arguments You must specify the following input argument:

Adjscale specifies *Adjscale*, which controls the normalization of the cross-covariance and cross-correlation. You can specify the following numeric values:

- 0 specifies the default normalization.
- 1 specifies the unbiased normalization.
- 2 specifies the biased normalization.

For more information, see [Table 31.6](#).

CROSSCORR.SetX Method

rc=obj.SetX (XSeries) ;

Specifies the cross time series array (*XSeries*) for the CROSSCORR object.

Input Arguments You must specify the following input argument:

XSeries specifies a numeric array that contains the dependent series for the CROSSCORR object.

CROSSCORR.SetY Method

rc=obj.SetY (YSeries) ;

Specifies the primary time series array (*YSeries*) for the CROSSCORR object.

Input Arguments You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the CROSSCORR object.

OUTCROSSCORR Object

The OUTCROSSCORR object collects output from a CROSSCORR object.

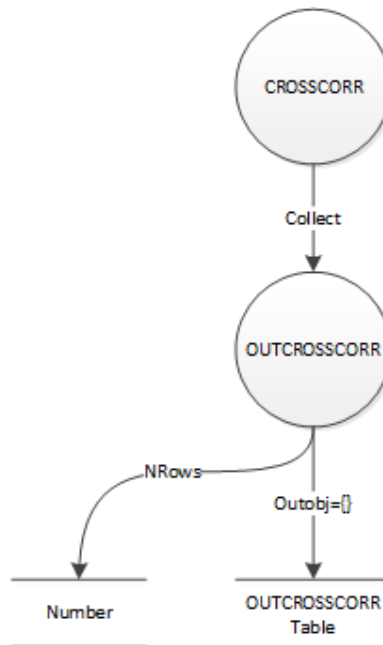
Table 31.5 summarizes the methods that are associated with the OUTCROSSCORR object.

Table 31.5 Methods of the OUTCROSSCORR Object

Method	Description
Collect	Collect the results of a CROSSCORR object
nrows	Get the current row count from the OUTCROSSCORR object

Figure 31.4 diagrams the methods of the OUTCROSSCORR object.

Figure 31.4 OUTCROSSCORR Data Flow



OUTCROSSCORR Methods

OUTCROSSCORR.Collect Method

rc=obj.Collect (CROSSCORRObj) ;

Collects the results from a CROSSCORR object and saves them to a table.

Arguments You must specify the following input argument:

CROSSCORRObj is a character string that specifies the name of the CROSSCORR object to use as the source of the cross-correlation and cross-covariance analysis results.

OUTCROSSCORR.nrows Attribute

nrows=obj.nrows () ;

Gets the current row count from the OUTCROSSCORR object.

Arguments There are no arguments associated with this method.

Output Table Schema

The collector object, OUTCROSSCORR, collects the results of a CROSSCORR object.

OUTCROSSCORR collects the results of a CROSSCORR object. The output table contains the columns that are shown in [Table 31.6](#).

Table 31.6 OUTCROSSCORR Output Table Contents

Column Name	Column Content	Statistics Used to Calculate Content
NAME	Variable name	N/A
CROSS	Cross variable name	N/A
LAG	Array of time lags	$h \in \{-H, \dots, 0, \dots, H\}$
N	Number of variance products	N_h , the number of observed products at lag h , ignoring missing values
MEAN	Mean of variable	
CROSSMEAN	Mean of cross variable	
CCOV	Array of cross-covariance estimates, with $2 \times nlag + 1$ entries	$\hat{\gamma}_{x,y}(h) = \begin{cases} \frac{1}{N} \sum_{t=h+1}^T (x_t - \bar{x})(y_{t-h} - \bar{y}) & 0 \leq h < T \\ \frac{1}{N} \sum_{t= h +1}^T (x_{t- h } - \bar{x})(y_t - \bar{y}) & -T < h < 0 \end{cases}$ where $N = T$ if adjscale is 0, $N = N_h$ if adjscale is 1, and $N = T$ if adjscale is 2. Or $\hat{\gamma}_{x,y}(h) = \begin{cases} \frac{1}{N} \sum_{t=h+1}^T (x_t - \bar{x})(y_{t-h} - \bar{y}) & 0 \leq h < T \\ \frac{1}{N} \sum_{t= h +1}^T (x_{t- h } - \bar{x})(y_t - \bar{y}) & -T < h < 0 \end{cases}$ when embedded missing values are present, where $N = N_h$ if adjscale is 0, $N = N_h$ if adjscale is 1, and $N = T$ if adjscale is 2.
CCF	Array of cross-correlation estimates, with $2 \times nlag + 1$ entries	$\hat{\rho}_{x,y}(h) = \hat{\gamma}_{x,y}(h) / \sqrt{\hat{\gamma}_x(0)\hat{\gamma}_y(0)}$
CCFSTD	Array of cross-correlation standard errors, with $2 \times nlag + 1$ entries	$\text{Std}(\hat{\rho}_{x,y}(h)) = 1/\sqrt{N_0}$
CCF2STD	Array of indicators of sample cross-correlation outside 2 standard errors, with $2 \times nlag + 1$ entries	$\text{Flag}(\hat{\rho}_{x,y}(h)) = \begin{cases} 1 & \hat{\rho}_{x,y}(h) > 2\text{Std}(\hat{\rho}_{x,y}(h)) \\ 0 & -2\text{Std}(\hat{\rho}_{x,y}(h)) < \hat{\rho}_{x,y}(h) < 2\text{Std}(\hat{\rho}_{x,y}(h)) \\ -1 & \hat{\rho}_{x,y}(h) < -2\text{Std}(\hat{\rho}_{x,y}(h)) \end{cases}$
CCFNORM	Array of normalized cross-correlations, with $2 \times nlag + 1$ entries	$\text{Norm}(\hat{\rho}_{x,y}(h)) = \hat{\rho}_{x,y}(h) / \text{Std}(\hat{\rho}_{x,y}(h))$

Table 31.6 *continued*

Column Name	Column Content	Statistics Used to Calculate Content
CCFPROB	Array of cross-correlation probabilities, with $2 \times nlag + 1$ entries	$\text{Prob}(\hat{\rho}_{x,y}(h)) = 2 (1 - \Phi(\text{Norm}(\hat{\rho}_{x,y}(h))))$
CCFLPROB	Array of cross-correlation log probabilities, with $2 \times nlag + 1$ entries	$\text{LogProb}(\hat{\rho}_{x,y}(h)) = -\log_{10}(\text{Prob}(\hat{\rho}_{x,y}(h)))$

DECOMP Class

DECOMP Object

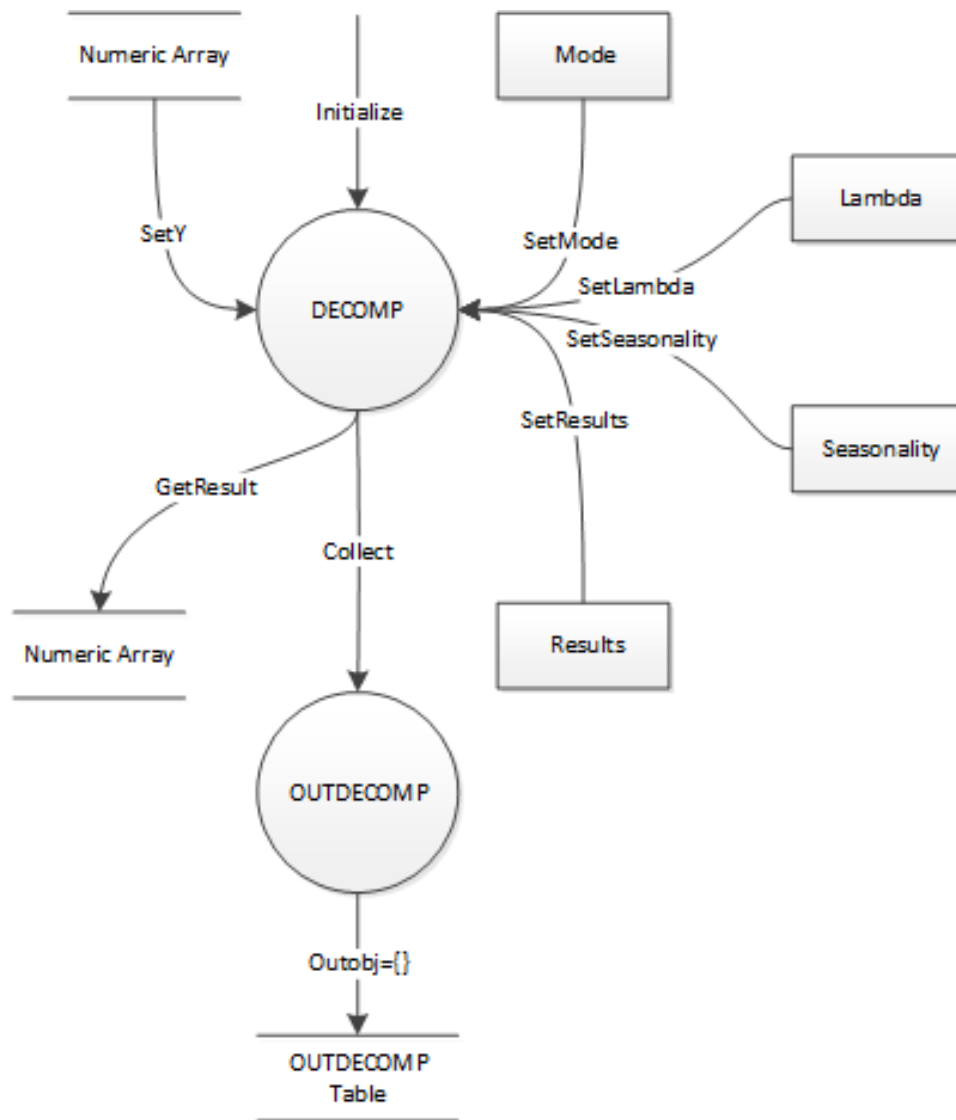
The DECOMP class object functions perform the seasonal decomposition or analysis of the time series arrays and output all the values and results.

Table 31.7 summarizes the methods that are associated with the DECOMP object.

Table 31.7 Methods of the DECOMP Object

Method	Description
GetResult	Get the analysis results
Initialize	Initialize the DECOMP object
Run	Run the analysis
SetLambda	Set the Hodrick-Prescott filter parameter for trend-cycle decomposition
SetMode	Set the type of decomposition
SetResults	Set the analysis results to be generated
SetSeasonality	Set the length of the seasonal cycles
SetY	Set the time series array

Figure 31.5 diagrams the methods of the DECOMP object.

Figure 31.5 DECOMP Data Flow

DECOMP Synopsis

DECLARE OBJECT *obj* (DECOMP) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetMode ('Name') ;
rc=obj.SetResults ('Name'<,'Name',... >) ;
rc=obj.SetSeasonality (Value) ;
rc=obj.SetLambda (Value) ;
rc=obj.Run () ;
rc=obj.GetResult ('Name',OutputArray) ;

```

DECOMP Methods

DECOMP.GetResult Method

```
rc=obj.GetResult ('Name',OutputArray) ;
```

Outputs the selected results to an array.

Input Arguments You must specify the following input arguments:

Name specifies a string that identifies the result series to return. You can specify the following values within single quotation marks:

<i>tcc</i>	returns the trend-cycle component.
<i>sic</i>	returns the seasonal-irregular component.
<i>sc</i>	returns the seasonal component.
<i>scstd</i>	returns the seasonal component standard errors.
<i>tcs</i>	returns the trend-cycle-seasonal component.
<i>ic</i>	returns the irregular component.
<i>sa</i>	returns the seasonally adjusted series.
<i>pcsa</i>	returns the percentage of change in seasonally adjusted series.
<i>tc</i>	returns the trend component.
<i>cc</i>	returns the cycle component.

OutputArray specifies a numeric array to receive the result series.

DECOMP.Initialize Method

```
rc=obj.Initialize () ;
```

Initializes a DECOMP object to an empty state. This method must be called before the time series array and other attributes are specified for the DECOMP object.

Arguments There are no arguments associated with this method.

DECOMP.Run Method

```
rc=obj.Run () ;
```

Runs the DECOMP object to perform the seasonal decomposition or analysis by using the time series array *YSeries* that has been specified for it. Upon successful completion, various results can be extracted from the DECOMP object.

Arguments There are no arguments associated with this method.

DECOMP.SetLambda Method

```
rc=obj.SetLambda (Value) ;
```

Sets the Hodrick-Prescott filter parameter for trend-cycle decomposition.

Input Arguments You must specify the following input argument:

Value specifies the length of the seasonal cycle. The default value is 1.

DECOMP.SetMode Method

rc=obj.SetMode (Value) ;

Sets the type of decomposition to be used to decompose the time series.

Input Arguments You can specify one of the following *Values*:

ADD | ADDITIVE specifies additive decomposition.
LOGADD | LOGADDITIVE specifies log-additive decomposition.
MULT | MULTIPLICATIVE specifies multiplicative decomposition.
MULTORADD specifies multiplicative or additive decomposition, depending on the data.
PSEUDOADD | PSEUDOADDITIVE specifies pseudo-additive decomposition.

DECOMP.SetResults Method

rc=obj.SetResults ('Name'<,'Name',... >) ;

Set a group of results to be generated in the DECOMP object.

Input Arguments You must specify the following input arguments:

Name specifies one or multiple strings that identify the result series to be generated. You can specify the following values within single quotation marks:

<i>tcc</i>	specifies the trend-cycle component.
<i>sic</i>	specifies the seasonal-irregular component.
<i>sc</i>	specifies the seasonal component.
<i>scstd</i>	specifies the seasonal component standard errors.
<i>tcs</i>	specifies the trend-cycle-seasonal component.
<i>ic</i>	specifies the irregular component.
<i>sa</i>	specifies the seasonally adjusted series.
<i>pcsa</i>	specifies the percentage of change in seasonally adjusted series.
<i>tc</i>	specifies the trend component.
<i>cc</i>	specifies the cycle component.

OutputArray specifies a numeric array to receive the result series.

DECOMP.SetSeasonality Method

rc=obj.SetSeasonality (Value) ;

Sets the length of seasonal cycles for the DECOMP object.

Input Arguments You must specify the following input argument:

Value specifies the length of the seasonal cycle, which must be either an integer greater than 1 or `_SEASONALITY_`. The predefined symbol `_SEASONALITY_` is the length of the seasonal cycle that is computed by using the seasonality or the time ID interval that you specify in a PROC TSMODEL step or a `runTimeCode` action call. For more information about predefined symbols, see the section “Predefined Symbols” in Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*). For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*). The default value is `_SEASONALITY_`.

DECOMP.SetY Method

`rc=obj.SetY (YSeries) ;`

Specifies the time series array (*YSeries*) for the DECOMP object.

Input Arguments You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the DECOMP object.

OUTDECOMP Object

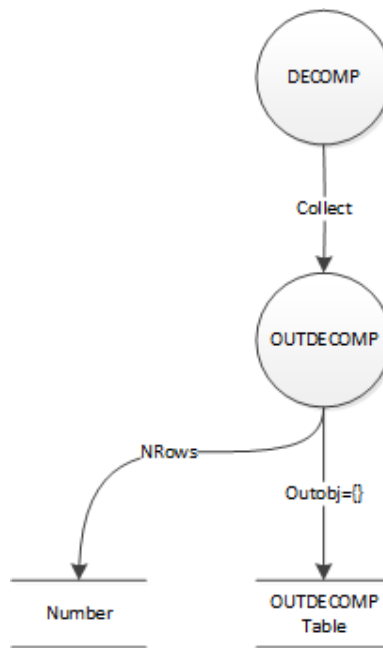
The OUTDECOMP object collects output from DECOMP object.

Table 31.8 summarizes the methods that are associated with the OUTDECOMP object.

Table 31.8 Methods of the OUTDECOMP Object

Method	Description
Collect	Collect the results of a DECOMP object
nrows	Get the current row count from the OUTDECOMP object

Figure 31.6 diagrams the methods of the OUTDECOMP object.

Figure 31.6 OUTDECOMP Data Flow

OUTDECOMP Methods

OUTDECOMP.Collect Method

rc=obj.Collect (DECOMPObj) ;

Collects the results from a DECOMP object and saves them to a table.

Arguments You must specify the following input argument:

DECOMPObj is a character string that specifies the name of the DECOMP object to use as the source of the seasonal decomposition or analysis results.

OUTDECOMP.nrows Attribute

nrows=obj.nrows () ;

Gets the current row count from the OUTDECOMP object.

Arguments

There are no arguments associated with this method.

Output Table Schema

The collector object, OUTDECOMP, collects the results of a DECOMP object.

OUTDECOMP collects the results of a DECOMP object. The output table contains the columns that are shown in [Table 31.9](#).

Table 31.9 OUTDECOMP Output Table Contents

Column Name	Column Content
NAME	Variable name
MODE	Mode of decomposition
TIMEID	Time ID values
SEASON	Seasonal index
ORIGINAL	Original series
TCC	Trend-cycle component
SIC	Seasonal-irregular component
SC	Seasonal component
SCSTD	Seasonal component standard error
TCS	Trend-cycle-seasonal component
IC	Irregular component
SA	Seasonally adjusted series
PCSA	Percentage change in seasonally adjusted series
TC	Trend component
CC	Cycle component

For more information about each components, see [Table 31.12](#).

OUTDECOMP2 Object

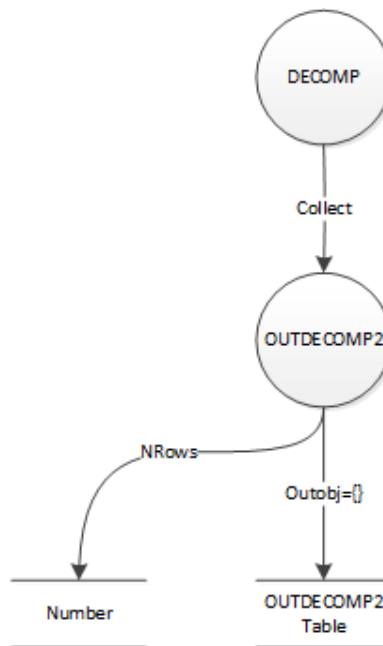
The OUTDECOMP2 object collects output from DECOMP object. In this object, the time periods are recorded as the column names instead of decomposition components.

[Table 31.10](#) summarizes the methods that are associated with the OUTDECOMP2 object.

Table 31.10 Methods of the OUTDECOMP2 Object

Method	Description
Collect	Collect the results of a DECOMP object
nrows	Get the current row count from the OUTDECOMP2 object

[Figure 31.7](#) diagrams the methods of the OUTDECOMP2 object.

Figure 31.7 OUTDECOMP2 Data Flow

OUTDECOMP2 Methods

OUTDECOMP2.Collect Method

`rc=obj.Collect (DECOMPObj) ;`

Collects the results from a DECOMP object and saves them to a table.

Arguments You must specify the following input argument:

DECOMPObj is a character string that specifies the name of the DECOMP object to use as the source of the seasonal decomposition or analysis results.

OUTDECOMP2.nrows Attribute

`nrows=obj.nrows () ;`

Gets the current row count from the OUTDECOMP2 object.

Arguments

There are no arguments associated with this method.

Output Table Schema

The collector object, OUTDECOMP2, collects the results of a DECOMP object.

OUTDECOMP2 collects the results of a DECOMP object. The output table contains the columns that are shown in [Table 31.11](#).

Table 31.11 OUTDECOMP2 Output Table Contents

Column Name	Column Content
NAME	Variable name
MODE	Mode of decomposition
COMPONENT	Component Name
PERIOD	Period

Details

You can use the DECOMP object to perform seasonal decomposition (also called seasonal adjustment) on the working series. The DECOMP object uses classical decomposition. More complex seasonal decomposition or adjustment analysis can be performed by using the X11 or the X12 procedure of SAS/ETS.

The SetMode method determines the mode of the seasonal adjustment decomposition to be performed. There are four modes: multiplicative (MULT), additive (ADD), pseudo-additive (PSEUDOADD), and log-additive (LOGADD) decomposition (in which the components are exponentiated to the original metric). The default value for the SetMode method is MULTORADD, which uses multiplicative mode for series that are strictly positive, pseudo-additive mode for series that are nonnegative, and additive mode for series that are not nonnegative.

The SetLambda method specifies the Hodrick-Prescott filter parameter (Hodrick and Prescott 1980) for trend-cycle decomposition. The Hodrick-Prescott filter is used to decompose the trend-cycle component into the trend component and the cycle component in an additive fashion. The SetLambda method takes a single argument *Value*, which specifies the length of the seasonal cycle. A smaller *Value* assigns less significance to the cycle; that is, a value of 0 implies no cycle component. The default value is 1600.

Table 31.12 shows the notation and keywords that are associated with seasonal decomposition (adjustment) analysis.

Table 31.12 Seasonal Adjustment Formulas

Component	Input Argument	SetMode Value	Formula
Original series	original	MULT	$O_t = TC_t S_t I_t$
		ADD	$O_t = TC_t + S_t + I_t$
		LOGADD	$\log(O_t) = TC_t + S_t + I_t$
		PSEUDOADD	$O_t = TC_t (S_t + I_t - 1)$
Trend-cycle component	tcc	MULT	Centered moving average of O_t
		ADD	Centered moving average of O_t
		LOGADD	Centered moving average of $\log(O_t)$
		PSEUDOADD	Centered moving average of O_t
Seasonal-irregular component	sic	MULT	$SI_t = S_t I_t = O_t / TC_t$
		ADD	$SI_t = S_t + I_t = O_t - TC_t$
		LOGADD	$SI_t = S_t + I_t = \log(O_t) - TC_t$
		PSEUDOADD	$SI_t = S_t + I_t - 1 = O_t / TC_t$

Table 31.12 *continued*

Component	Keyword	MODE= Option	Formula
Seasonal component	sc	MULT	Seasonal averages of SI_t
		ADD	Seasonal averages of SI_t
		LOGADD	Seasonal averages of SI_t
		PSEUDOADD	Seasonal averages of SI_t
Irregular component	ic	MULT	$I_t = SI_t/S_t$
		ADD	$I_t = SI_t - S_t$
		LOGADD	$I_t = SI_t - S_t$
		PSEUDOADD	$I_t = SI_t - S_t + 1$
Trend-cycle-seasonal component	tcs	MULT	$TCS_t = TC_t S_t = O_t/I_t$
		ADD	$TCS_t = TC_t + S_t = O_t - I_t$
		LOGADD	$TCS_t = TC_t + S_t = O_t - I_t$
		PSEUDOADD	$TCS_t = TC_t S_t$
Trend component	tc	MULT	$T_t = TC_t - C_t$
		ADD	$T_t = TC_t - C_t$
		LOGADD	$T_t = TC_t - C_t$
		PSEUDOADD	$T_t = TC_t - C_t$
Cycle component	cc	MULT	$C_t = TC_t - T_t$
		ADD	$C_t = TC_t - T_t$
		LOGADD	$C_t = TC_t - T_t$
		PSEUDOADD	$C_t = TC_t - T_t$
Seasonally adjusted series	sa	MULT	$SA_t = O_t/S_t = TC_t I_t$
		ADD	$SA_t = O_t - S_t = TC_t + I_t$
		LOGADD	$SA_t = O_t/\exp(S_t) = \exp(TC_t + I_t)$
		PSEUDOADD	$SA_t = TC_t I_t$

When s is odd, the trend-cycle component is computed from the s -period-centered moving average as follows:

$$TC_t = \sum_{k=-\lfloor s/2 \rfloor}^{\lfloor s/2 \rfloor} y_{t+k}/s$$

When s is even, the trend-cycle component is computed from the s -period-centered moving average as follows:

$$TC_t = \sum_{k=-s/2}^{s/2-1} (y_{t+k} + y_{t+1+k})/2s$$

The seasonal component is obtained by averaging the seasonal-irregular component for each season,

$$S_{k+js} = \sum_{t=k \bmod s} \frac{SI_t}{T/s}$$

where $0 \leq j \leq T/s$ and $1 \leq k \leq s$. The seasonal components are normalized to sum to 1 (multiplicative) or 0 (additive).

DPF Class

DPF Object

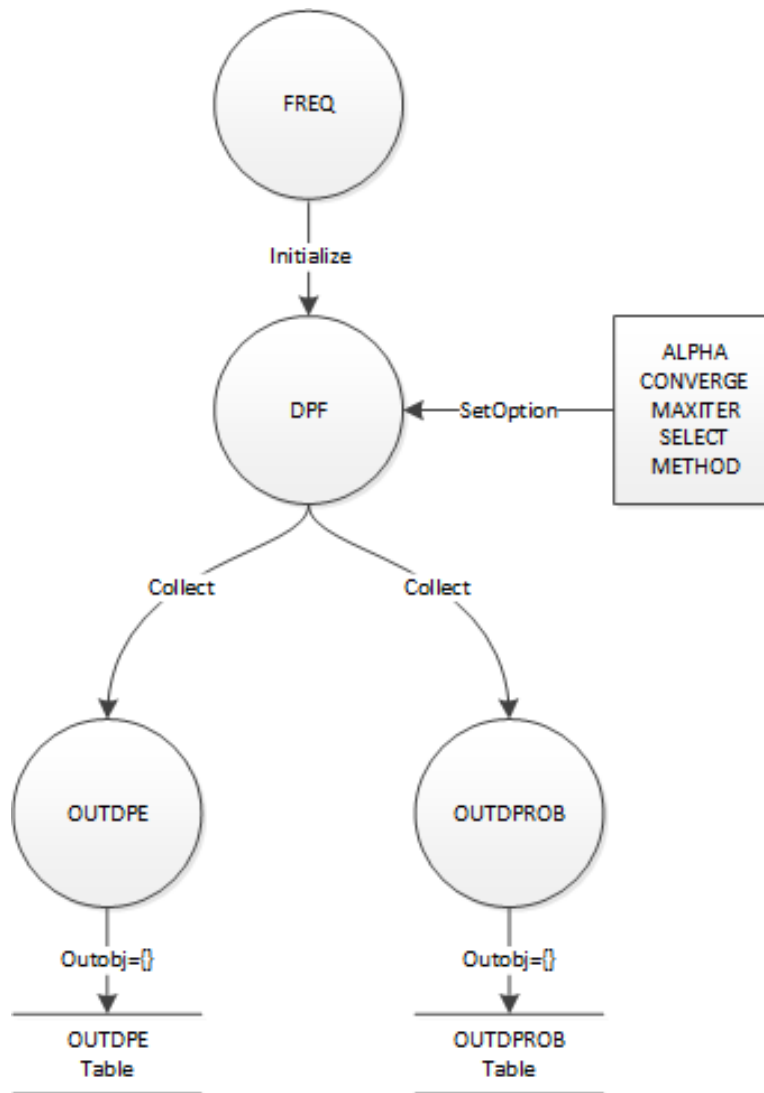
The DPF class object functions take the output of the FREQ class object as input. The DPF class object performs count distribution analysis for a time series.

Table 31.13 summarizes the methods that are associated with the DPF object.

Table 31.13 Methods of the DPF Object

Method	Description
Initialize	Initialize the DPF object
Run	Run the analysis
SetOption	Set the specifications

Figure 31.8 diagrams the methods of the DPF object.

Figure 31.8 DPF Data Flow

DPF Synopsis

DECLARE OBJECT *obj* (DPF) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize (FREQObj) ;
rc=obj.SetOption ('Name',Value) ;
rc=obj.Run () ;

```

DPF Methods

DPF.Initialize Method

```
rc=obj.Initialize (FREQObj) ;
```

Specifies a FREQ object, which includes frequency analysis results. This method must be called before the time series arrays and other attributes are specified for the DPF object.

Input Arguments

You must specify a FREQ object, which includes frequency analysis results.

FREQObj specifies the FREQ object as input.

DPF.Run Method

```
rc=obj.Run () ;
```

Runs the DPF object to perform the count distribution analysis. Upon successful completion, various results can be extracted from the DPF object.

DPF.SetOption Method

```
rc=obj.SetOption ('Name', Value) ;
```

Sets a named option for the DPF object.

Input Arguments

You can specify one of the following for '*Name*' and its associated *Value*:

ALPHA	specifies the confidence level size, where alpha must be between 0 and 1. The default value is 0.05.								
CONVERGE	specifies the convergence criterion.								
MAXITER	specifies the maximum number of iterations, where <i>Value</i> is an integer.								
'SELECT'	takes a string <i>Value</i> that specifies the distribution selection criterion. You can specify one of the following <i>Values</i> : <table data-bbox="500 1394 1240 1528"> <tr> <td>AIC</td><td>specifies Akaike's information criterion.</td></tr> <tr> <td>BIC</td><td>specifies the Bayesian information criterion.</td></tr> <tr> <td>LOGLIK</td><td>specifies the log-likelihood.</td></tr> </table> The default value is LOGLIK.	AIC	specifies Akaike's information criterion.	BIC	specifies the Bayesian information criterion.	LOGLIK	specifies the log-likelihood.		
AIC	specifies Akaike's information criterion.								
BIC	specifies the Bayesian information criterion.								
LOGLIK	specifies the log-likelihood.								
'METHOD'	takes a string <i>Value</i> that specifies the candidate distribution to use in the analysis. You can specify one of the following <i>Values</i> : <table data-bbox="500 1701 1435 1913"> <tr> <td>BEST</td><td>specifies the best distribution, based on the value of the select argument.</td></tr> <tr> <td>BINOMIAL</td><td>specifies the binomial distribution.</td></tr> <tr> <td>GEOMETRIC</td><td>specifies the geometric distribution.</td></tr> <tr> <td>NEGBINOMIAL</td><td>specifies the negative binomial distribution.</td></tr> </table>	BEST	specifies the best distribution, based on the value of the select argument.	BINOMIAL	specifies the binomial distribution.	GEOMETRIC	specifies the geometric distribution.	NEGBINOMIAL	specifies the negative binomial distribution.
BEST	specifies the best distribution, based on the value of the select argument.								
BINOMIAL	specifies the binomial distribution.								
GEOMETRIC	specifies the geometric distribution.								
NEGBINOMIAL	specifies the negative binomial distribution.								

- POISSON** specifies the Poisson distribution.
- ZMBINOMIAL** specifies the zero-modified binomial distribution.
- ZMGEOMETRIC** specifies the zero-modified geometric distribution.
- ZMNEGBINOMIAL** specifies the zero-modified negative binomial distribution.
- ZMPOISSON** specifies the zero-modified Poisson distribution.

The default distribution is BEST.

OUTDPE Object

The OUTDPE object collects output from a DPF object.

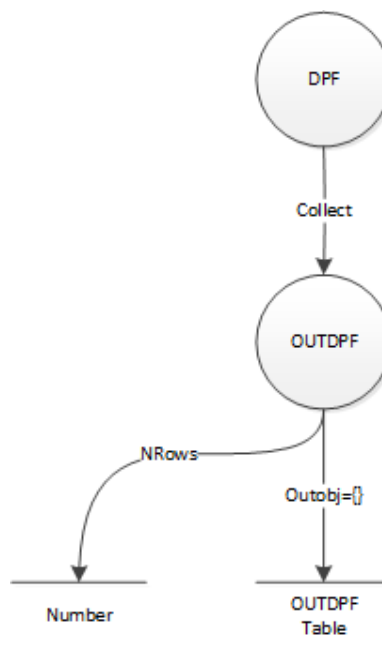
Table 31.14 summarizes the methods that are associated with the OUTDPE object.

Table 31.14 Methods of the OUTDPE Object

Method	Description
<code>Collect</code>	Collect the results of a DPF object
<code>nrows</code>	Get the current row count from the OUTDPE object

Figure 31.9 diagrams the methods of the OUTDPF object.

Figure 31.9 OUTDPE Data Flow



OUTDPE Methods

OUTDPE.Collect Method

```
rc=obj.Collect (DPFObj) ;
```

Collects the results from a DPF object and saves them to a table.

Arguments You must specify the following input argument:

DPFObj is a character string that specifies the name of the DPF object to use as the source of the parameter estimation of the count distribution.

OUTDPE.nrows Attribute

```
nrows=obj.nrows () ;
```

Gets the current row count from the OUTDPE object.

Arguments There are no arguments associated with this method.

Output Table Schema

OUTDPE collects predictions using the count distribution that has the best fit. The output table contains the columns that are shown in [Table 31.15](#).

Table 31.15 OUTDPE Output Table Contents

Column Name	Column Content
NAME	Name of target variable
Distribution	Distribution
Parameter	Name of parameter
Estimate	Parameter estimate
StdErr	Standard error
Tvalue	<i>t</i> value
Probt	Approximate probability > <i>t</i>
Lower	Lower 95% of parameter estimate
Upper	Upper 95% of parameter estimate

OUTDPROB Object

The OUTDPROB object collects output from a DPF object.

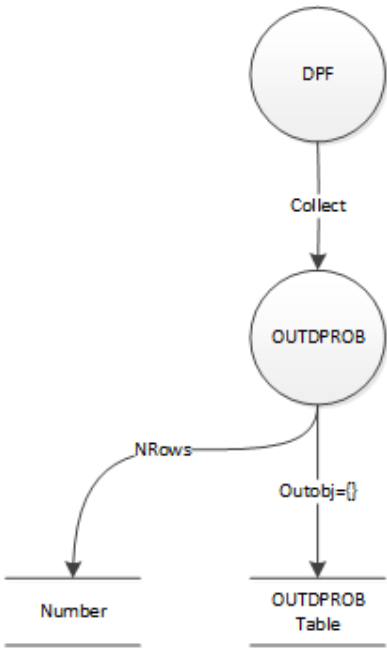
Table 31.16 summarizes the methods that are associated with the OUTDPROB object.

Table 31.16 Methods of the OUTDPROB Object

Method	Description
Collect	Collect the results of a DPF object
nrows	Get the current row count from the OUTDPROB object

Figure 31.10 diagrams the methods of the OUTDPROB object.

Figure 31.10 OUTDPROB Data Flow



OUTDPROB Methods

OUTDPROB.Collect Method

rc=obj.Collect (DPFObj) ;

Collects the results from a DPF object and saves them to a table.

Arguments You must specify the following input argument:

DPFObj is a character string that specifies the name of the DPF object to use as the source of the prediction of the count distribution.

OUTDPROB.nrows Attribute

nrows=obj.nrows () ;

Gets the current row count from the OUTDPROB object.

Arguments There are no arguments associated with this method.

Output Table Schema

OUTDPROB collects predictions using the count distribution that has the best fit. The output table contains the columns that are shown in [Table 31.17](#).

Table 31.17 OUTDPROB Output Table Contents

Column Name	Column Content
<u>_NAME_</u>	Name of target variable
Index	Index
Value	Values
Observedzeros	Observed zeros
Expectedzeros	Expected zeros
Observed	Observed counts
Expected	Expected counts
Expectedlower	Expected lower confidence
Expectedupper	Expected upper confidence
Probability	Probability
Probabilitylower	Probability lower confidence
Probabilityupper	Probability upper confidence
Chisquare	Chi-square statistic
Chisquareprob	Chi-square probability
Chisquarelogprob	Chi-square log probability

Example

This example uses the TSMODEL procedure to analyze the frequency of the time series Air and then conducts a count distribution analysis.

The following DATA step loads the Sashelp.Air data set into the library. This DATA step assumes that your library is named mylib, but you can substitute any appropriately defined library.

```
data mylib.air (replace=yes);
    set Sashelp.air;
run;

proc tsmodel data=mylib.air outobj=(of=mylib.outfreq(replace=YES)
                                   odpe=mylib.dpe(replace=YES)
                                   odprob=mylib.dprob(replace=YES)
                                   );
    var air;
    id date interval=month;
    require tsa;
    submit;
        declare object f(FREQ);
        declare object of(OUTFREQ);
        declare object dpf(DPF);
        declare object odpe(OUTDPE);
        declare object odprob(OUTDPROB);
        rc = f.Initialize();
        rc = f.SetOption("SEASONALITY",12);
        rc = f.SetY(air);
        rc = f.Run();
        rc = of.Collect(f);

        rc = dpf.Initialize(f);
        rc = dpf.SetOption("SELECT","AIC","METHOD","POISSON");
        rc = dpf.Run();
        rc = odpe.Collect(dpf);
        rc = odprob.Collect(dpf);
    endsubmit;
run;
```

FREQ Class

FREQ Object

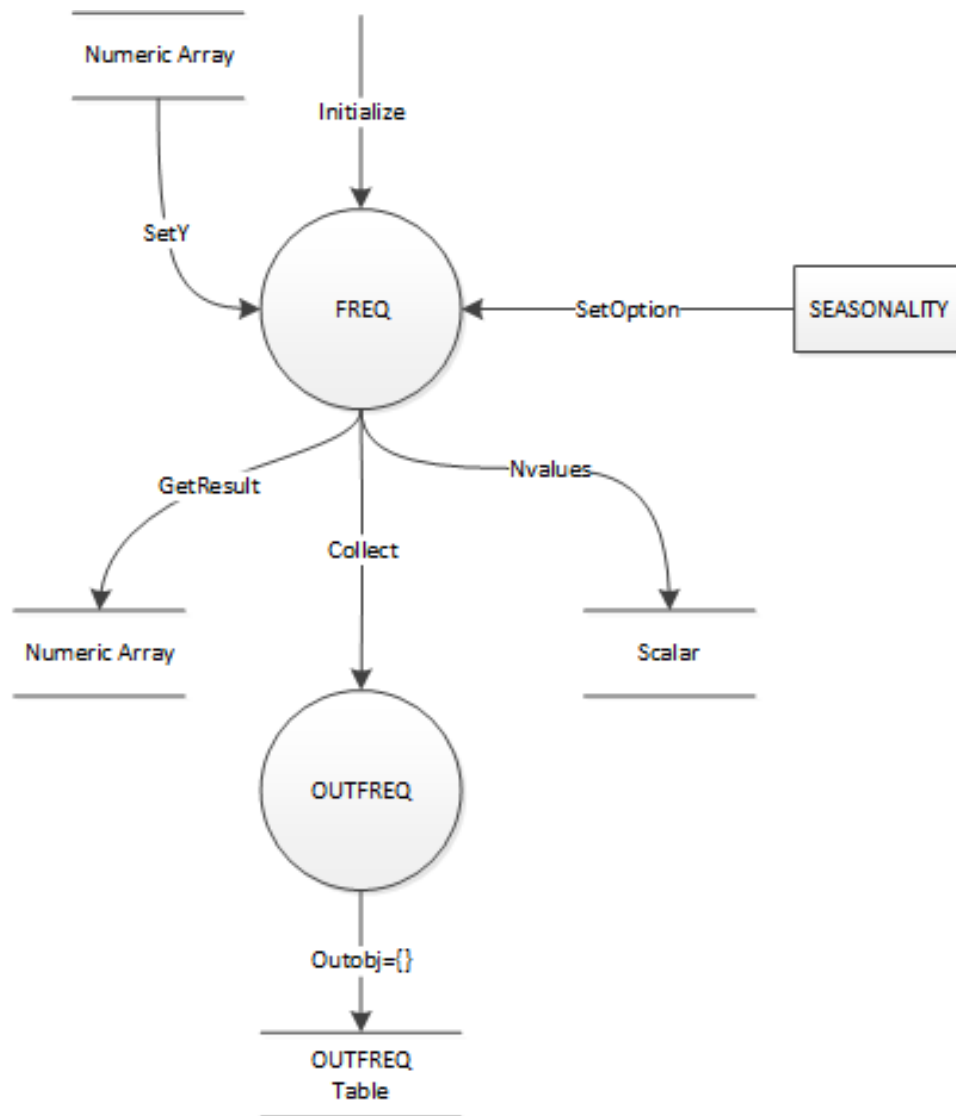
The FREQ class object function analyzes the frequency of a time series and outputs all unique values and corresponding counts for the time series.

Table 31.18 summarizes the methods that are associated with the FREQ object.

Table 31.18 Methods of the FREQ Object

Method	Description
GetResult	Get the analysis result
Initialize	Initialize the FREQ object
Nvalues	Return the number of unique values in the input series
Run	Run the analysis
SetOption	Set the specifications
SetY	Set the input time series

Figure 31.11 diagrams the methods of the FREQ object.

Figure 31.11 FREQ Data Flow

FREQ Synopsis

DECLARE OBJECT *obj* (FREQ) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetOption ('Name', Value) ;
rc=obj.Run () ;
rc=obj.GetResult ('Name', OutputArray) ;
rc=obj.Nvalues () ;

```

FREQ Methods

FREQ.GetResult Method

```
rc=obj.GetResult ('Name',OutputArray) ;
```

Outputs the selected results to an array.

Input Arguments You must specify the following input arguments:

Name specifies a string that identifies the result series to return. You can specify the following values within single quotation marks:

'VALUES'	retrieves a monotonically increasing list of the unique values that are found in the input series that is specified by using the SetY method.
'COUNTS'	retrieves the number of occurrences of each unique value.
'PCTS'	retrieves the percentage of total occurrences of each unique value.

OutputArray specifies a numeric array to receive the result series.

FREQ.Initialize Method

```
rc=obj.Initialize () ;
```

Initializes an FREQ object to an empty state. This method must be called before the time series arrays and other attributes are specified for the FREQ object.

Arguments There are no arguments associated with this method.

FREQ.Nvalues Method

```
rc=obj.Nvalues () ;
```

Returns the number of unique values in the input series to *rc*. This number is also the length of all result series that are returned by the GetResult method. It is set to a missing value if the Run method did not complete successfully.

Arguments There are no arguments associated with this method.

FREQ.Run Method

```
rc=obj.Run () ;
```

Runs the FREQ object to perform the autocorrelation and autocovariance analysis by using the time series array *YSeries* that has been specified for it. Upon successful completion, various results can be extracted from the FREQ object.

FREQ.SetOption Method

```
rc=obj.SetOption ('Name',Value) ;
```

Sets a named option for the FREQ object.

Input Arguments

You can specify the following for *'Name'* and its associated *Value*:

'SEASONALITY' specifies the seasonality of the time series, where *Value* is an integer. You can also specify `_SEASONALITY_`. The predefined symbol `_SEASONALITY_` is the length of the seasonal cycle that is computed by using the seasonality or the time ID interval that you specify in a PROC TSMODEL step or a **runTimeCode** action call. For more information about predefined symbols, see the section “Predefined Symbols” in Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*). For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*). The default value is 0.

FREQ.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the time series array (*YSeries*) for the FREQ object.

Input Arguments You must specify the following input argument:

YSeries specifies the time series array to analyze.

OUTFREQ Object

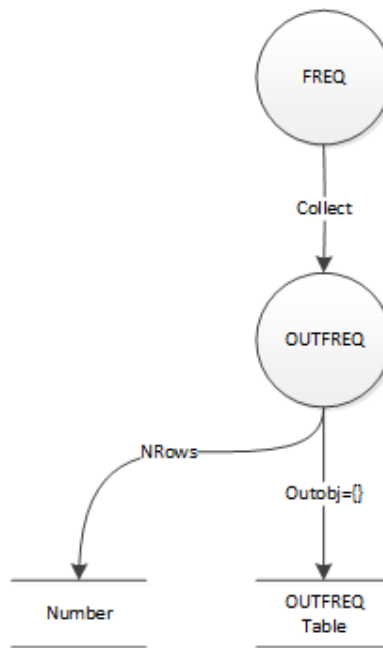
The OUTFREQ object collects output from a FREQ object.

Table 31.19 summarizes the methods that are associated with the OUTFREQ object.

Table 31.19 Methods of the OUTFREQ Object

Method	Description
<code>Collect</code>	Collect the results of a FREQ object
<code>nrows</code>	Get the current row count from the OUTFREQ object

Figure 31.12 diagrams the methods of the OUTFREQ object.

Figure 31.12 OUTFREQ Data Flow

OUTFREQ Methods

OUTFREQ.Collect Method

`rc=obj.Collect (FREQObj) ;`

Collects the results from a FREQ object and saves them to a table.

Arguments You must specify the following input argument:

FREQObj is a character string that specifies the name of the FREQ object to use as the source of the frequency analysis.

OUTFREQ.nrows Attribute

`nrows=obj.nrows () ;`

Gets the current row count from the OUTFREQ object.

Arguments There are no arguments associated with this method.

Output Table Schema

The collector object, OUTFREQ, collects the results of frequency function.

OUTFREQ collects the results of frequency function. The output table contains the columns that are shown in [Table 31.20](#).

Table 31.20 OUTFREQ Output Table Contents

Column Name	Column Content
NAME	Name of target variable
Values	Array of series values
Counts	Array of frequency series counts
Percent	Percentage of total frequency

Example

This example uses the TSMODEL procedure to analyze the frequency of the time series Air:

```
proc tsmode1 data=mylib.air outobj=(of=mylib.outfreq(replace=YES));
  var air;
  id date interval=month;
  require tsa;
  submit;
  declare object f(FREQ);
  declare object of(OUTFREQ);
  rc = f.Initialize();
  rc = f.SetOption("SEASONALITY",12);
  rc = f.SetY(air);
  rc = f.Run();
  rc = of.Collect(f);
  endsubmit;
run;
```

SPECTRA Class

SPECTRA Object

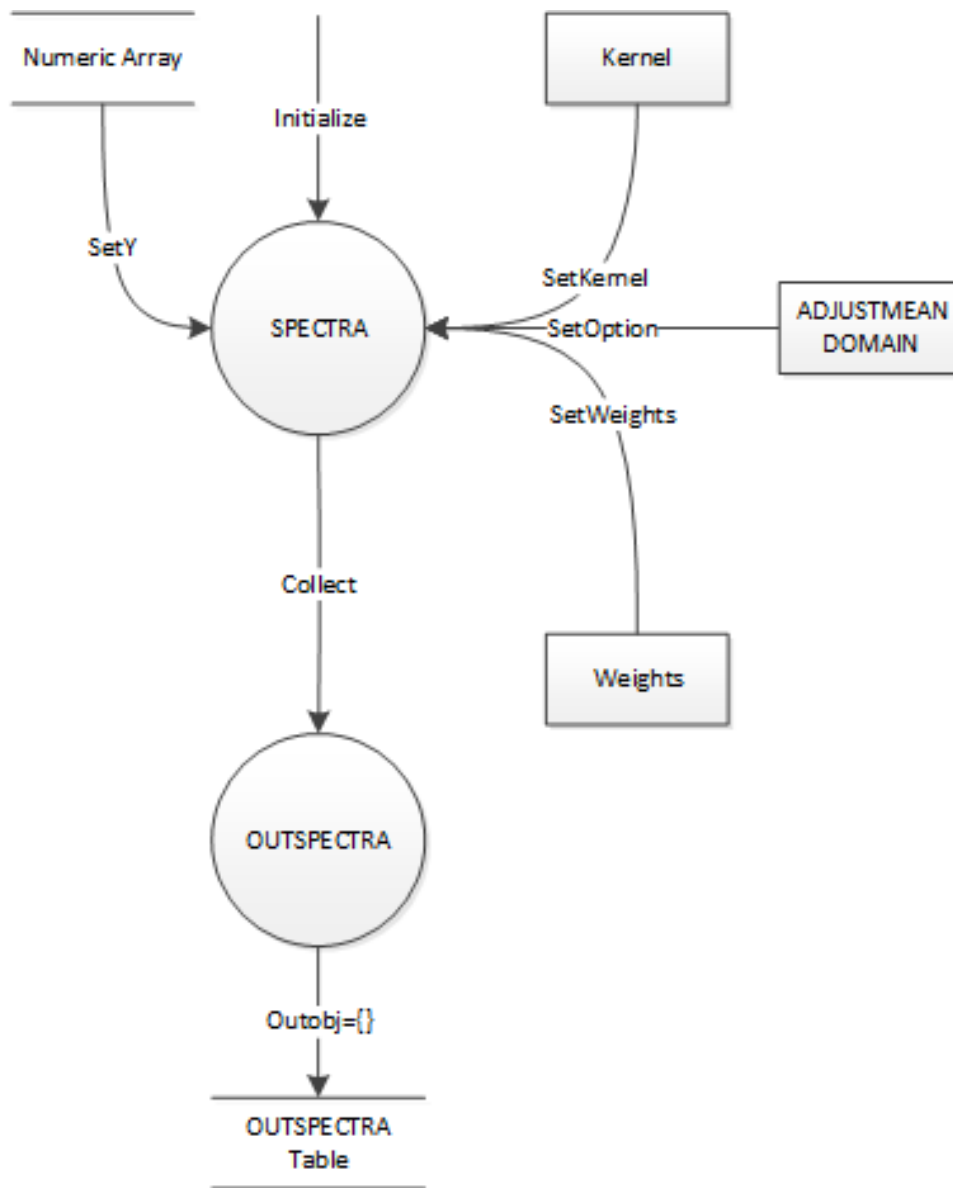
The SPECTRA object performs the spectral analysis of a time series array and outputs the results.

Table 31.21 summarizes the methods that are associated with the SPECTRA object.

Table 31.21 Methods of the SPECTRA Object

Method	Description
Initialize	Initialize the SPECTRA object
Run	Run the analysis
SetKernel	Set the kernel
SetOption	Set the specifications
SetWeights	Set the weights
SetY	Set the input time series

Figure 31.13 diagrams the methods of the SPECTRA object.

Figure 31.13 SPECTRA Data Flow

SPECTRA Synopsis

DECLARE OBJECT *obj* (SPECTRA) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetOption ('Name',Value <,'Name',Value, ... > ) ;
rc=obj.SetKernel ('Kernel' < ,Value,Value> ) ;
rc=obj.SetWeights (NumericArray) ;
rc=obj.Run () ;

```

SPECTRA Methods

SPECTRA.Initialize Method

`rc=obj.Initialize () ;`

Initializes a SPECTRA object to an empty state. This method must be called before the time series arrays and other attributes for the SPECTRA object are specified.

Arguments

There are no arguments associated with this method.

SPECTRA.SetKernel Method

`rc=obj.SetKernel ('Kernel' < ,C,E>) ;`

Sets the kernel options for the SPECTRA object.

Input Arguments

You can specify one of the following input arguments for '*Kernel*':

'NONE'	uses no kernel.
'PARZEN'	uses a Parzen kernel.
'BARTLETT'	uses a Bartlett kernel.
'TUKEY'	uses a Tukey-Hanning kernel.
'TRUNC TRUNCAT'	uses a truncated kernel.
'QS QUADR'	uses a quadratic spectral kernel.

If neither WEIGHTS nor a kernel function is specified, the spectral density estimate is identical to the unmodified periodogram.

The *C* and *E* arguments are optional. You can specify them as follows:

<i>C</i>	specifies a positive scale coefficient for the kernel function. The specified value must be within the range 0 to 32,000.
<i>E</i>	specifies a positive exponent for the kernel function. The specified value must be within the range 0 to 1.

Kernel Option Details

You can further parameterize each of the kernel functions with a kernel scale factor by using the *C* and *E* arguments. Table 31.22 shows the default values of the kernel scale parameters *c* and *e* that are associated with each of the kernel functions together with their kernel scale factor values *M* for a series that has 100 periodogram ordinates. The formula that is used to generate the table entries is $M = cK^e$, where *K* is the number of Fourier component frequencies.

For example, to apply the truncated kernel by using default scale factor parameters in the frequency domain, you could specify the following:

Table 31.22 Default Kernel Scale Factor Parameters

Kernel	c	e	M
Bartlett	1/2	1/3	2.32
Parzen	1	1/5	2.51
Quadratic	1/2	1/5	1.26
Tukey-Hanning	2/3	1/5	1.67
Truncated	1/4	1/5	0.63

```
rc = SetKernel('TRUNCAT');
```

For more information about the kernel function parameterization, see the section “[Using Kernel Specifications](#)” on page 2290.

SPECTRA.SetOption Method

```
rc=obj.SetOption('Name', Value <,'Name', Value, ... > );
```

Sets a named option for the SPECTRA object.

Input Arguments

You can specify one of the following for '*Name*' and its associated *Value*:

'ADJUSTMEAN'

takes a string *Value* that specifies whether to adjust the series by its mean before performing the spectral analysis. You can specify one of the following *Values*:

TRUE | T | YES | Y adjusts the mean before performing spectral analysis.

FALSE | F | NO | N does not adjust the mean before performing spectral analysis.

The default is NO.

'DOMAIN'

takes a string *Value* that specifies how to interpret the smoothing function. You can specify one of the following *Values*:

FREQUENCY smooths the periodogram ordinate.

TIME applies the kernel as a filter to the time series autocovariance function.

The default is FREQUENCY. For more information about the DOMAIN option, see the section “[Using Kernel Specifications](#)” on page 2290.

'ALPHA'

specifies the width of a window that is drawn around the spectral density estimate in a spectral density versus frequency plot. Based on approximations proposed by Brockwell and Davis (1991), periodogram ordinates fall within this window with a confidence level of $1 - \alpha$. The value α must be between 0 and 1; the default is 0.5.

SPECTRA.SetWeights Method

```
rc=obj.SetWeights (NumericArray) ;
```

Sets the relative weights for the SPECTRA object.

Input Arguments

You must specify the following input argument:

NumericArray specifies a series of relative weights that are used to compute a spectral density estimate as the moving average smoothing of periodogram ordinates.

If neither WEIGHTS nor a kernel function is specified, the spectral density estimate is identical to the unmodified periodogram.

SPECTRA.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the time series array (*YSeries*) for the SPECTRA object.

Input Arguments

You must specify the following input argument:

YSeries specifies a numeric array that contains the series for the SPECTRA object.

SPECTRA.Run Method

```
rc=obj.Run () ;
```

Runs the SPECTRA object to perform the spectral analysis by using the time series array *YSeries* that has been specified for it. Upon successful completion, various results can be extracted from the SPECTRA object.

Arguments

There are no arguments associated with this method.

OUTSPECTRA Object

The OUTSPECTRA object collects output from a SPECTRA object.

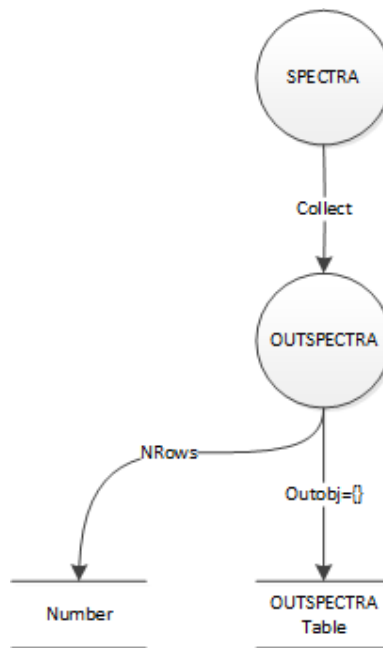
Table 31.23 summarizes the methods that are associated with the OUTSPECTRA object.

Table 31.23 Methods of the OUTSPECTRA Object

Method	Description
Collect	Collect the results of a SPECTRA object
nrows	Get the current row count from the OUTSPECTRA object

Figure 31.14 diagrams the methods of the OUTSPECTRA object.

Figure 31.14 OUTSPECTRA Data Flow



OUTSPECTRA Methods

OUTSPECTRA.Collect Method

rc=obj.Collect (SPECTRAObj) ;

Collects the results from a SPECTRA object and saves them to a table.

Arguments You must specify the following input argument:

SPECTRAObj is a character string that specifies the name of the SPECTRA object to use as the source of the spectral analysis.

OUTSPECTRA.nrows Attribute

nrows=obj.nrows () ;

Gets the current row count from the OUTSPECTRA object.

Arguments There are no arguments associated with this method.

Output Table Schema

OUTSPECTRA collects the results of a SPECTRA object. The output table contains the following columns:

Column Name	Column Content
NAME	Name of the variable
FREQ	Frequency in radians from 0 to π
PERIOD	Period or wavelength
COS	Cosine transform
SIN	Sine transform
P	Periodogram
S	Spectral density estimates
S_LOWER	Lower bound of confidence interval of spectral density
S_UPPER	Upper bound of confidence interval of spectral density

Details

The SPECTRA object uses the finite Fourier transform to decompose data series into a sum of sine and cosine terms of different amplitudes and wavelengths. The finite Fourier transform decomposition of the series x_t is

$$x_t = \frac{a_0}{2} + \sum_{k=1}^{K-1} f_k (a_k \cos \omega_k t + b_k \sin \omega_k t)$$

$$f_k = \begin{cases} 1/2 & \text{if } T \text{ is even and } k = K - 1 \\ 1 & \text{otherwise} \end{cases}$$

where

t	is the time subscript, $t = 0, 1, 2, \dots, T - 1$
x_t	are the equally spaced time series data
T	is the number of observations in the time series
K	is the number of frequencies in the Fourier decomposition: $K = \frac{T+2}{2}$ if T is even; $K = \frac{T+1}{2}$ if T is odd
k	is the frequency subscript, $k = 0, 1, 2, \dots, K - 1$
a_0	is the mean term: $a_0 = 2\bar{x}$
a_k	are the cosine coefficients
b_k	are the sine coefficients
ω_k	are the Fourier frequencies: $\omega_k = \frac{2\pi k}{T}$

Functions of the Fourier coefficients a_k and b_k can be plotted against frequency or against wavelength to form *periodograms*. The amplitude periodogram I_k is defined as

$$I_k = \frac{T}{2} (a_k^2 + b_k^2)$$

The periodogram I_k is an estimate at the discrete frequencies ω_k of the spectral density function that characterizes the series x_t . Smoothing the periodogram enables an improved spectral density estimate with

reduced variance and bias to be achieved at these points. Smoothing can be accomplished either by using a spectral window smoothing function or by applying a lag window filter to the series autocovariance function.

When the *Value* of the DOMAIN argument is FREQUENCY, the spectral density estimates are computed by using the following equation to smooth the periodogram ordinates:

$$S_k(M) = \sum_{\tau=K-T}^{K-1} w\left(\frac{\tau}{M}\right) I_{k+\tau}$$

where $w(\theta)$ is the spectral window function whose form is specified by either the SetKernel method or the SetWeights method; M is the kernel scale parameter, which acts as a frequency scaling factor in the spectral window smoothing function; and values of $I_{k+\tau}$ that fall outside $0 \leq k + \tau \leq K - 1$ are mapped to values inside this range by the equations presented previously.

When the *Value* of the DOMAIN argument is TIME, the spectral density values are estimated by applying a lag window filter, $\lambda(h, M)$, to the series autocovariance function. Then, the spectral density estimate can be computed from the filtered autocovariance function by using the following equation:

$$S_k(M) = \sum_{h=-(T-1)}^{T-1} \lambda(h, M) \hat{\gamma}(h) \cos h\omega_k$$

In this case, the kernel scale parameter, M , serves as a scale factor for the lag length, h , in the time domain. In the lag window formulation, the spectral density estimate is a consistent estimator as $T, M \rightarrow \infty$ under the conditions $\lambda(h, M) = 0$ for $|h| > M$ and $\lim_{T \rightarrow \infty} M/T = 0$. These conditions lead to the following parameterization of M :

$$M = cK^e$$

where the values $c > 0$ and $0 < e < 1$ satisfy the consistency conditions. To specify the kernel scale parameter explicitly, set c to the desired scale factor and set $e = 0$.

For uniformity and computational efficiency, all spectral density estimates are calculated using a spectral window weighting function, $w(\theta)$, which is applied to the periodogram ordinates. In the case where the *Value* of the DOMAIN argument is TIME, the effective spectral window weighting function is computed by the following equation:

$$w_{\text{TIME}}(\theta) = \sum_{h=-(T-1)}^{T-1} \lambda(h, M) \cos h\theta$$

Because the kernel scale parameter, M , serves as a lag scale factor in the time domain and as a bandwidth scale factor in the frequency domain, the impact of M on spectral density estimates depends on the *Value* of the DOMAIN argument. When the *Value* of the DOMAIN argument is specified to be FREQUENCY, increasing values of M decrease variance and increase bias in the spectral density estimates; when the *Value* of the DOMAIN argument is TIME, increasing values of M increase variance and decrease bias.

Using Kernel Specifications

You can specify one of 10 different kernel smoothing functions in the SPECTRA object. Five smoothing functions are available in the SetKernel method, and five complementary smoothing functions that correspond to lag window filters are available when the SetKernel method is used and the *Value* of the DOMAIN argument is TIME.

For example, a Parzen kernel with a support of 11 periodogram ordinates in the frequency domain can be specified using the SetKernel method:

```
rc = SetKernel('PARZEN', 5, 0);
```

The SPECTRA class of the TSA package supports the following spectral window kernel functions in the frequency domain, where $x = \tau/M$:

BARTLETT (Bartlett kernel):

$$w(x) = \begin{cases} 1 - |x| & |x| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

PARZEN (Parzen kernel):

$$w(x) = \begin{cases} 1 - 6|x|^2 + 6|x|^3 & 0 \leq |x| \leq \frac{1}{2} \\ 2(1 - |x|)^3 & \frac{1}{2} \leq |x| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

QS (quadratic spectral kernel):

$$w(x) = \frac{3}{(2\pi x)^2} \left(\frac{\sin 2\pi x}{2\pi x} - \cos 2\pi x \right)$$

TUKEY (Tukey-Hanning kernel):

$$w(x) = \begin{cases} (1 + \cos(\pi x))/2 & |x| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

TRUNCAT (truncated kernel):

$$w(x) = \begin{cases} 1 & |x| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

When the *Value* of the DOMAIN argument is TIME, the five preceding kernel functions are interpreted as lag window filters on the autocovariance function. The lag window kernel functions correspond to the following spectral window smoothing functions, where $\theta = 2\pi\tau/T$:

BARTLETT (Bartlett equivalent lag window filter):

$$w(\theta) = \frac{1}{2\pi M} \left(\frac{\sin(M\theta/2)}{\sin(\theta/2)} \right)^2$$

PARZEN (Parzen equivalent lag window filter):

$$w(\theta) = \frac{6}{\pi M^3} \left(\frac{\sin(M\theta/4)}{\sin(\theta/2)} \right)^4 \left(1 - \frac{2}{3} \sin^2(\theta/2) \right)$$

QS (quadratic spectral equivalent lag window filter):

$$w(\theta) = \begin{cases} \frac{3M}{4\pi} (1 - (M\theta/\pi)^2) & |\theta| \leq \pi/M \\ 0 & |\theta| > \pi/M \end{cases}$$

TUKEY (Tukey-Hanning equivalent lag window filter):

$$w(\theta) = \frac{1}{4} D_M(\theta - \pi/M) + \frac{1}{2} D_M(\theta) + \frac{1}{4} D_M(\theta + \pi/M)$$

$$D_M(\theta) = \frac{1}{2\pi} \frac{\sin[(M + 1/2)\theta]}{\sin(\theta/2)}$$

TRUNC (truncated equivalent lag window filter):

$$w(\theta) = D_M(\theta)$$

Using Specification of Weighting Constants

You can specify any number of weighting constants. The constants are interpreted symmetrically around the middle weight. The middle constant (or the constant to the right of the middle if an even number of weight constants is specified) is the relative weight of the current periodogram ordinate. The constant immediately to the right of the middle constant is the relative weight of the next periodogram ordinate, and so on. The actual weights that are used in the smoothing process are the weights that are specified in the `SetWeights` method, which are scaled so that they sum to 1.

The moving average calculation reflects at each end of the periodogram to accommodate the periodicity of the periodogram function when there are not enough periodogram ordinates.

For example, a simple triangular weighting can be specified using the following `WEIGHTS` option:

```
array group1[5]/nosymbols; group1[1]=1; group1[2]=2; group1[3]=3;
    group1[4]=2; group1[5]=1;
rc = spectra.SetWeights(group1);
```

Computational Method

If the number of observations, T , factors into prime integers that are less than or equal to 23 and the product of the square-free factors of T is less than 210, then the SPECTRA object uses the fast Fourier transform that was developed by Cooley and Tukey (1965) and implemented by Singleton (1969). If T cannot be factored in this way, then the SPECTRA object uses a Chirp-Z algorithm similar to that proposed by Monro and Branch (1977).

Missing Values

Missing values are replaced with an estimate of the mean to perform spectral analyses. This treatment of a series with missing values is consistent with the approach of Priestley (1981).

STATIONARITY Class

STATIONARITY Object

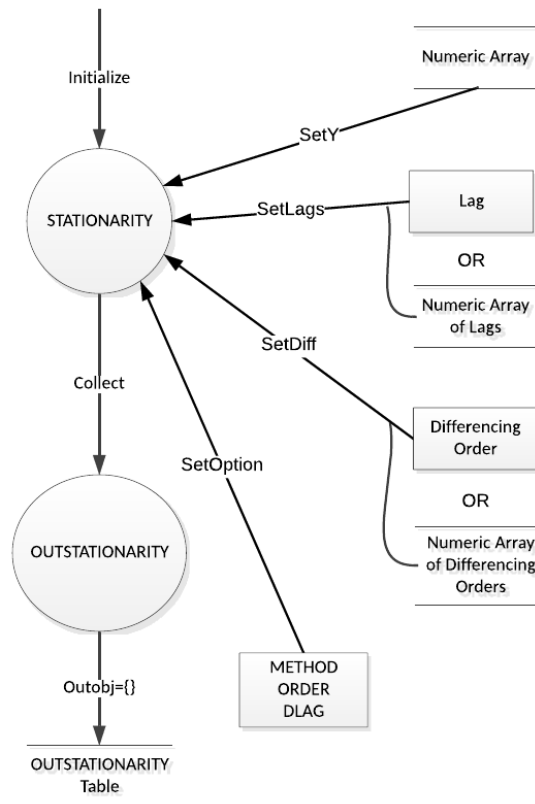
The STATIONARITY class object functions perform the stationarity tests on a time series and store all the results that you can retrieve later by using the OUTSTATIONARITY object.

Table 31.24 summarizes the methods that are associated with the STATIONARITY object.

Table 31.24 Methods of the STATIONARITY Object

Method	Description
Initialize	Initialize the STATIONARITY object
SetY	Set the input time series
SetOption	Set the specifications
SetLags	Set the number of lags
SetDiff	Set the differences of series
Run	Run the analysis

Figure 31.15 diagrams the methods of the STATIONARITY object.

Figure 31.15 STATIONARITY Data Flow

STATIONARITY Synopsis

DECLARE OBJECT *obj* (STATIONARITY) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY ( YSeries ) ;
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... > ) ;
rc=obj.SetLags ( Value ) ;
rc=obj.SetDiff ( Value ) ;
rc=obj.Run () ;

```

STATIONARITY Methods

STATIONARITY.Initialize Method

rc=obj.Initialize () ;

Initializes a STATIONARITY object to an empty state. This method must be called before the time series array and other attributes are specified for the STATIONARITY object.

Arguments There are no arguments associated with this method.

STATIONARITY.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the time series array (*YSeries*) for the STATIONARITY object.

Input Arguments You must specify the following input argument:

YSeries specifies a numeric array that contains the time series for the STATIONARITY object to analyze.

STATIONARITY.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... > ) ;
```

Sets a named option for the STATIONARITY object.

Input Arguments

You must specify at least one of the following for '*Name*' and its associated *Value*:

'METHOD' takes a string *Value* that specifies the stationarity test method, which can be the augmented Dickey-Fuller (ADF) test, Phillips-Perron (PP) test, or random-walk-with-drift (RW) test. You can specify one of the following *Values*:

ADF | Dickey-Fuller specifies the augmented Dickey-Fuller test.

PP | Phillips-Perron specifies the Phillips-Perron test.

RW | Random-Walk specifies the random-walk-with-drift test.

The default is ADF.

'ORDER' | 'DLAG' takes an integer *Value* that specifies lags for the stationarity test method.

The *Value* must be a positive integer between 1 and 12 (inclusive), denoting the order that the STATIONARITY object uses for the test methods. If the value is between 2 and 12 (inclusive) and the method is ADF, the seasonal ADF test is performed. If the value is 1 and the method is ADF, the nonseasonal ADF test is performed. For the PP or RW method, values between 2 and 12 (inclusive) are discarded and the default order of 1 is used.

The default value is 1.

STATIONARITY.SetLags Method

```
rc=obj.SetLags (Value) ;
```

Sets the lags that the STATIONARITY object uses in the stationarity test calculation.

Input Arguments You must specify the following input argument:

Value specifies the number of lags to use in the calculation. The allowable value type is an array or a numeric scalar. Specifying a scalar *Value* is equivalent to specifying an array that contains consecutive numbers from 0 to *Value* (inclusive).

The default value is 2, which is equivalent to an array {0, 1, 2}.

STATIONARITY.SetDiff Method

```
rc=obj.SetDiff (Value) ;
```

Sets a differencing option for the STATIONARITY object.

Input Arguments You must specify the following input argument:

Value specifies a nonnegative integer scalar or an array of nonnegative integers. The STATIONARITY object differences the time series for each integer value in succession prior to performing the test calculations. For example, if you specify the array { 1, 3 }, then the object first takes the first-order difference and then applies a third-order difference to that first-order-differenced series.

The default value is 0.

STATIONARITY.Run Method

```
rc=obj.Run () ;
```

Runs the STATIONARITY object to perform various stationarity test calculations by using the time series array that you specify in the SetY method according to the options that you specify in other setter methods, such as SetOption. Upon successful completion, you can extract various results from the OUTSTATIONARITY object.

OUTSTATIONARITY Object

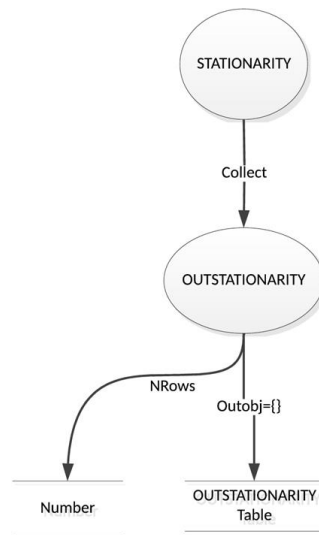
The OUTSTATIONARITY object collects output from a STATIONARITY object.

Table 31.25 summarizes the methods that are associated with the OUTSTATIONARITY object.

Table 31.25 Methods of the OUTSTATIONARITY Object

Method	Description
Collect	Collect the results of a STATIONARITY object
nrows	Get the current row count from the OUTSTATIONARITY object

Figure 31.16 diagrams the methods of the OUTSTATIONARITY object.

Figure 31.16 OUTSTATIONARITY Data Flow

OUTSTATIONARITY Methods

OUTSTATIONARITY.Collect Method

```
rc=obj.Collect (STATIONARITYObj) ;
```

Collects the results from a STATIONARITY object and saves them to a table.

Arguments You must specify the following input argument:

STATIONARITYObj is a character string that specifies the name of the STATIONARITY object to use as the source of the stationarity test analysis results.

OUTSTATIONARITY.nrows Attribute

```
nrows=obj.nrows () ;
```

Gets the current row count from the OUTSTATIONARITY object.

Arguments There are no arguments associated with this method.

Output Table Schema

The collector object, OUTSTATIONARITY, collects the results of a STATIONARITY object.

OUTSTATIONARITY The output table contains the columns shown in [Table 31.26](#).

Table 31.26 OUTSTATIONARITY Output Table Contents

Column	Description
Method	Unit root test method
MethodCode	Short code for the unit root test method
Type	Unit root test type
TypeCode	Short code for the unit root test type
Lags	Time lag
Rho	Regression test statistic. This column has missing values when the method is random walk with drift.
ProbRho	Regression test probability value. This column has missing values when the method is random walk with drift
Tau	Studentized test statistic
ProbTau	Studentized test probability value
F	<i>F</i> -test statistic. This column has missing values when the method is seasonal augmented Dickey-Fuller, random walk with drift, or Phillips-Perron. It also has missing values for the (nonseasonal) augmented Dickey-Fuller test when the test type is zero mean.
ProbF	<i>F</i> -test probability value. This column has missing values when the method is seasonal augmented Dickey-Fuller, random walk with drift, or Phillips-Perron. It also has missing values for the (nonseasonal) augmented Dickey-Fuller test when the test type is zero mean.

TSAOUTLIER Class

TSAOUTLIER Object

The TSAOUTLIER object detects outliers for time series data on the basis of the Hampel identifier and outputs the results.

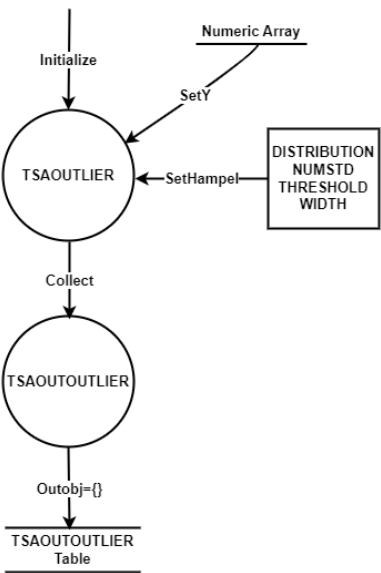
Table 31.27 summarizes the methods that are associated with the TSAOUTLIER object.

Table 31.27 Methods of the TSAOUTLIER Object

Method	Description
Initialize	Initialize the TSAOUTLIER object
Run	Run the analysis
SetY	Set the input time series
SetHampel	Set outlier detection options for the Hampel-based method

Figure 31.17 diagrams the methods of the TSAOUTLIER object.

Figure 31.17 TSAOUTLIER Data Flow



TSAOUTLIER Synopsis

DECLARE OBJECT *obj* (TSAOUTLIER) ;

Method syntax, in order of typical usage:

```
rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetHampel ('Name', Value < , 'Name', Value, ... >) ;
rc=obj.Run () ;
```

TSAOUTLIER Methods

TSAOUTLIER.Initialize Method

```
rc=obj.Initialize () ;
```

Initializes a TSAOUTLIER object to an empty state. You must call this method before you specify the time series arrays and other attributes for this object.

Arguments

There are no arguments associated with this method.

TSAOUTLIER.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the time series array (*YSeries*) for the TSAOUTLIER object.

Input Arguments

You must specify the following input argument:

YSeries specifies a numeric array that contains the series for the TSAOUTLIER object. This object detects outliers by using the raw values of this series, which means that it keeps each missing value as-is while detecting outliers. The result is that missing values can be detected as outliers.

TSAOUTLIER.SetHampel Method

```
rc=obj.SetHampel (< 'Name', Value, 'Name', Value, ... >) ;
```

Specifies named options for the TSAOUTLIER object.

Input Arguments

You can specify one or more of the following '*Names*' and the associated *Value*:

'DISTRIBUTION' takes a string *Value* that specifies the data distribution that the TSAOUTLIER object uses to compute the threshold for detecting outliers. You can specify the following distribution *Values*:

EXP	specifies an exponential distribution.
LAPLACE	specifies a Laplace distribution.
LOGISTIC	specifies a logistic distribution.

	NORM	specifies a normal distribution. This is the default distribution.
	UNIFORM	specifies a uniform distribution.
'NUMSTD'		takes a positive numeric <i>Value</i> that specifies the number of standard deviations. The TSAOUTLIER object computes the outlier detection threshold by multiplying this <i>Value</i> by a scaling factor that corresponds to the distribution that you specify in the 'DISTRIBUTION' option. The default value is 3.
'THRESHOLD'		takes a positive numeric <i>Value</i> that specifies the threshold for detecting outliers. If you specify this option, then the 'DISTRIBUTION' and 'NUMSTD' options are ignored.
'WIDTH'		specifies a positive integer <i>Value</i> that determines the number of input observations to include in the sample window for outlier detection. If <i>Value</i> is equal to k , then a window of a maximum size of $2 \cdot k + 1$ is centered around each point of the time series, and outliers are detected by using the data in that window. The default value is 3.

TSAOUTLIER.Run Method

```
rc=obj.Run () ;
```

Runs the TSAOUTLIER object to perform the outlier analysis.

Arguments

There are no arguments associated with this method.

TSAOUTOUTLIER Object

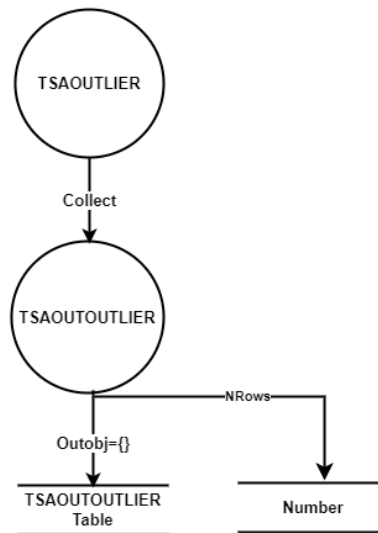
The TSAOUTOUTLIER object collects output from a TSAOUTLIER object.

Table 31.28 summarizes the methods that are associated with the TSAOUTOUTLIER object.

Table 31.28 Methods of the TSAOUTOUTLIER Object

Method	Description
Collect	Collect the results of a TSAOUTLIER object
nrows	Get the current row count from the TSAOUTOUTLIER object

Figure 31.18 diagrams the methods of the TSAOUTOUTLIER object.

Figure 31.18 TSAOUTOUTLIER Data Flow

TSAOUTOUTLIER Methods

TSAOUTOUTLIER.Collect Method

rc=obj.Collect (TSAOUTLIERObj) ;

Collects the results from a TSAOUTLIER object and saves them to a table.

Arguments You must specify the following input argument:

TSAOUTLIERObj is a character string that specifies the name of the TSAOUTLIER object to use as the source of the outlier analysis.

TSAOUTOUTLIER.nrows Attribute

nrows=obj.nrows () ;

Gets the current row count from the TSAOUTOUTLIER object.

Arguments There are no arguments associated with this method.

Output Table Schema

The collector object, TSAOUTOUTLIER, collects the results of a TSAOUTLIER object.

Table 31.29 shows the contents of the TSAOUTOUTLIER object.

Table 31.29 Contents of the TSAOUTOUTLIER Object

Column	Type	Description
EVENT	String	Name of the outlier event
STATISTIC	Numeric	Values of the standardized distance from the median
TIMEID	Numeric	Time ID values for the series

Details

For more information about the TSAOUTLIER object, see the section “Hampel Method of Outlier Detection” (Chapter 2, *SAS Visual Forecasting: Time Series Packages*).

TSA Object

Functional Summary

NOTE: Each function in this section has a prefix of “TSA.”; however, the prefixes are omitted in descriptions for better readability.

Table 31.30 summarizes the functions in the TSA package.

Table 31.30 Functions in Time Series Analysis Package

TSA Function	Description
ACCUMULATE	Accumulates a univariate time series to a particular frequency
ACCUMULATE2	Accumulates a high-frequency time series to a lower frequency and expands the lower-frequency series to have the same length as the high-frequency series
ACF	Computes autocorrelation and autocovariance for a time series array
ARMAORDERS	Performs tests to tentatively identify the autoregressive and moving average orders of mixed autoregressive moving average models
CCF	Computes the cross-correlation and cross-covariance for two time series arrays
DPF Class Object	Performs count distribution analysis for time series
FREQ Class Object	Performs frequency analysis of a time series
INTERMITTENCYTEST	Tests for intermittency of a univariate time series
IACF	Computes the inverse autocorrelation for a time series array

Table 31.30 *continued*

TSA Function	Description
MOVINGSUMMARY	Computes statistics for a set of values within a moving time window
PACF	Computes the partial autocorrelation for a time series array
SCALE	Scales a time series between the minimum value and the maximum value of the original time series
SEASONALDECOMP	Computes the seasonal indices of a univariate time series using classical decomposition
SEASONALINDICES	Computes the seasonal indices of a univariate time series by using regression seasonal dummies
SEASONTTEST	Tests for seasonality of a univariate time series
SIMILARITY	Performs similarity analysis for time series
STATIONARITYTEST	Tests for stationarity of a univariate time series
TRANSFORM	Transforms time series according to the specified transformation type
UNBIASEDNESS	Tests whether a univariate time series is unbiased
WHITENOISE	Tests for white noise of a time series array

ACCUMULATE Method

`rc = TSA.ACCUMULATE (time, y, 'interval', id, z, <'accumulate'>, <'setmiss'>, <'zeromiss'>);`

The ACCUMULATE function accumulates a univariate time series to a particular frequency.

Required Arguments

You must specify the following arguments, separated by commas:

time specifies the time ID array for the time series.

y specifies the times series array to accumulate.

'interval' specifies the time interval.

You can specify any valid time interval name as a quoted string. Here are some typical intervals:

DAY specifies a seasonal cycle of length 7.

MONTH specifies a seasonal cycle of length 12.

QTR specifies a seasonal cycle of length 4.

YEAR specifies a seasonal cycle of length 1.

See Chapter 4, “Date Intervals, Formats, and Functions” (*SAS/ETS User’s Guide*), for other kinds of time intervals that you can specify.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

'accumulate' specifies the accumulation statistic.

You can specify the following values within single quotation marks:

AVERAGE AVG	specifies the average of the values in the time series.
CSS	specifies the corrected sum of squares of the values in the time series.
FIRST	specifies the first value of the time series.
LAST	specifies the last value of the time series.
MAXIMUM MAX	specifies the maximum value in the time series.
MEDIAN MED	specifies the median of the values in the time series.
MINIMUM MIN	specifies the minimum value in the time series.
N	specifies the number of nonmissing observations.
NMISS	specifies the number of missing observations.
NOBS	specifies the number of observations.
STDDEV STD	specifies the standard deviation of the values in the time series.
TOTAL	specifies the total sum of the values in the time series.
USS	specifies the uncorrected sum of squares of the values in the time series.

The default is TOTAL.

'setmiss' specifies the missing value interpretation.

You can specify the following values within single quotation marks:

AVERAGE AVG	specifies the accumulated average value.
FIRST	specifies the accumulated first nonmissing value.
MAXIMUM MAX	specifies the accumulated maximum value.
MEDIAN MED	specifies the accumulated median value.
MINIMUM MIN	specifies the accumulated minimum value.
MISSING	specifies a missing value.
NEXT	specifies the next period’s accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.
PREVIOUS PREV	specifies the previous period’s accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.

The default is MISSING.

'zeromiss' specifies the zero value interpretation.

You can specify the following values within single quotation marks:

- BOTH** sets both beginning and ending zeros to missing.
- LEFT** sets beginning zeros to missing.
- NONE** leaves beginning and ending zeros unchanged.
- RIGHT** sets ending zeros to missing.

The default is NONE.

Returned Values

The ACCUMULATE function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

- id* returns the time ID array for the accumulated time series.
- z* returns the accumulated time series array.

Example

This example uses the TSMODEL procedure to accumulate a time series:

```
proc tsmodel data=mylib.air outarray=mylib.outarray;
  id date interval=month;
  var air;
  outarrays qtravg new_id;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.ACCUMULATE(date, air, 'QTR', new_id, qtravg, 'AVERAGE', , );
  endsubmit;
run;
```

ACCUMULATE2 Method

```
rc = TSA.ACCUMULATE2 (time, y, 'interval', id, z, <'accumulate'>, <'setmiss'>, <'zeromiss'> ) ;
```

The ACCUMULATE2 function accumulates a high-frequency time series to a lower frequency and expands the lower-frequency time series to the same length as the high-frequency series.

Required Arguments

You must specify the following arguments, separated by commas:

time specifies the time ID array for the time series.

y specifies the times series array to accumulate.

'interval' specifies the time interval.

You can specify any valid time interval name as a quoted string. Here are some typical intervals:

DAY specifies a seasonal cycle of length 7.

MONTH specifies a seasonal cycle of length 12.

QTR specifies a seasonal cycle of length 4.

YEAR specifies a seasonal cycle of length 1.

See Chapter 4, “Date Intervals, Formats, and Functions” (*SAS/ETS User’s Guide*), for other kinds of time intervals that you can specify.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

'accumulate' specifies the accumulation statistic.

You can specify the following values within single quotation marks:

AVERAGE | AVG specifies the average of the values in the time series.

CSS specifies the corrected sum of squares of the values in the time series.

FIRST specifies the first value of the time series.

LAST specifies the last value of the time series.

MAXIMUM | MAX specifies the maximum value in the time series.

MEDIAN | MED specifies the median of the values in the time series.

MINIMUM | MIN specifies the minimum value in the time series.

N specifies the number of nonmissing observations.

NMISS specifies the number of missing observations.

NOBS specifies the number of observations.

STDDEV | STD specifies the standard deviation of the values in the time series.

TOTAL specifies the total sum of the values in the time series.

USS specifies the uncorrected sum of squares of the values in the time series.

The default is TOTAL.

'setmiss' specifies the missing value interpretation.

You can specify the following values within single quotation marks:

AVERAGE AVG	specifies the accumulated average value.
FIRST	specifies the accumulated first nonmissing value.
LAST	specifies the accumulated last nonmissing value.
MAXIMUM MAX	specifies the accumulated maximum value.
MEDIAN MED	specifies the accumulated median value.
MINIMUM MIN	specifies the accumulated minimum value.
MISSING	specifies a missing value.
NEXT	specifies the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.
PREVIOUS PREV	specifies the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.

The default is MISSING.

'zeromiss' specifies the zero value interpretation.

You can specify the following values within single quotation marks:

BOTH	sets both beginning and ending zeros to missing.
LEFT	sets beginning zeros to missing.
NONE	leaves beginning and ending zeros unchanged.
RIGHT	sets ending zeros to missing.

The default is NONE.

Returned Values

The ACCUMULATE2 function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success.
< 0	Computational failure.

id returns the time ID array for the accumulated time series.

z returns the accumulated time series array.

Example

This example uses the TSMODEL procedure to accumulate a monthly time series into a yearly time series:

```
proc tsmode1 data=mylib.air outarray=mylib.outarray;
  id date interval=month;
  var air;
  outarrays yearavg_expand yearid_expand;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.ACCUMULATE2(date, air, 'YEAR', yearid_expand, yearavg_expand,
                    'AVERAGE', , );
  endsubmit;
run;
```

ACF Method

```
rc = TSA.ACF (y, nlag, <lags>, <df>, <mu>, <acov>, <acf>, <acfstd>, <acf2std>, <acfnorm>,
              <acfprob> <acflprob>);
```

The ACF function computes autocorrelation and autocovariance for a time series array.

Required Arguments

You must specify the following arguments, separated by a comma:

y specifies the times series array.
nlag specifies the number of the lag to use in the calculation.

Returned Values

The ACF function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

lags returns the numbers of the lags that were used in the calculation.
df returns the number of observations used to compute *acov* and *acf*.

<i>mu</i>	returns the mean estimate.
<i>acov</i>	returns an array of covariance estimates, with <i>nlag</i> + 1 entries.
<i>acf</i>	returns an array of autocorrelation estimates, with <i>nlag</i> + 1 entries.
<i>acfstd</i>	returns an array of standard errors, with <i>nlag</i> + 1 entries.
<i>acf2std</i>	returns an array of twice standard errors, with <i>nlag</i> + 1 entries.
<i>acfnorm</i>	returns an array of normalized autocorrelation, with <i>nlag</i> + 1 entries.
<i>acfprob</i>	returns an array of autocorrelation probabilities, with <i>nlag</i> + 1 entries.
<i>acflprob</i>	returns an array of autocorrelation log probabilities, with <i>nlag</i> + 1 entries.

Example

This example uses the TSMODEL procedure to compute the autocorrelation of lag 3 of the time series Air:

```
proc tsmodel data=mylib.air outscalar=mylib.outscalars
      outarray=mylib.outarray;
  id date interval=month;
  var air;
  outscalars mu;
  outarrays acf acov lags df acfstd;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.ACF(air, 3, lags, df, mu, acov, acf, acfstd, , , , );
  endsubmit;
run;
```

ARMAORDERS Method

rc = TSA.ARMAORDERS (*y*, *dif*, <'method'>, <*p*>, <*q*>, <*perror*>, <*porders*>, <*qorders*>);

The ARMAORDERS function performs tests to tentatively identify the autoregressive and moving average orders of mixed autoregressive moving average models.

Required Arguments

You must specify the following argument:

<i>y</i>	specifies the times series array to test.
<i>dif</i>	specifies either an array of positive integers or a positive integer that is used for differencing. The default value is 0.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

'method' specifies the method of tentative order selection.
You can specify the following values within single quotation marks:

- ESACF** specifies the extended sample autocorrelation function.
- MINIC** specifies the minimum information criterion.
- SCAN** specifies the squared canonical correlations.

The default value is MINIC.

p specifies the autoregressive order range, where *p* is an array of two nonnegative integers that define the minimum and maximum values.
By default, *p* is the array [0,5].

q specifies the moving average order range, where *q* is an array of two nonnegative integers that define the minimum and maximum values.
By default, *q* is the array [0,5].

perror specifies the autoregressive orders used to estimate the error series for the MINIC method, where *perror* is an array of two nonnegative integers that define the minimum and maximum values.
By default, *perror* is the array [max(*p*), max(*p*)+max(*q*)].

Returned Values

The ARMAORDERS function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

- porders* returns the recommended autoregressive orders.
- qorders* returns the recommended moving average orders.

Example

This example uses the TSMODEL procedure to tentatively identify the autoregressive and moving average orders for the time series Air:

```
proc tsmodel data=mylib.air outscalar=mylib.outscalars;
  id date interval=month;
  var air;
  outscalars porders qorders;
  require tsa;
  submit;
  declare object TSA(tsa);
  porders = 0;
  qorders = 0;
  Array P[2]/nosymbols; P[1]=0; P[2]=5;
  Array Q[2]/nosymbols; Q[1]=0; Q[2]=5;
  rc=TSA.ARMAORDERS(air, 0, 'SCAN', P, Q, , porders, qorders);
  rc=TSA.ARMAORDERS(air, 1, 'ESACF', P, Q, , porders, qorders);
  rc=TSA.ARMAORDERS(air, 1, 'MINIC', P, Q, , porders, qorders);
  endsubmit;
run;
```

CCF Method

$rc = \text{TSA.CCF}(y, x, nlag, \langle lags \rangle, \langle df \rangle, \langle ymu \rangle, \langle xmu \rangle, \langle ccov \rangle, \langle ccf \rangle, \langle ccfstd \rangle, \langle ccf2std \rangle, \langle ccfnorm \rangle, \langle ccfprob \rangle, \langle ccflprob \rangle, \langle adjscale \rangle);$

The CCF function computes the cross-correlation and cross-covariance for two time series arrays.

Required Arguments

You must specify the following arguments, separated by commas:

- y* specifies one times series array.
- x* specifies the other time series array.
- nlag* specifies the number of the lag to compute.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

- adjscale* specifies *adjscale*, which controls the normalization of the cross-covariance and cross-correlation. You can specify the following numeric values:
 - 0 specifies the default normalization.
 - 1 specifies the unbiased normalization.
 - 2 specifies the biased normalization.

For more information, see [Table 31.6](#).

Returned Values

The CCF function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

<i>lags</i>	returns an array of lags that were computed, with $nlag + 1$ entries.
<i>df</i>	returns an array of number of products for which to compute the cross-correlation, with $nlag + 1$ entries.
<i>ymu</i>	returns the mean estimate of input time series <i>y</i> .
<i>xmu</i>	returns the mean estimate of input time series <i>x</i> .
<i>ccov</i>	returns an array of cross-covariance estimates, with $2 \times nlag + 1$ entries.
<i>ccf</i>	returns an array of cross-correlation estimates, with $2 \times nlag + 1$ entries.
<i>ccfstd</i>	returns an array of standard errors, with $2 \times nlag + 1$ entries.
<i>ccf2std</i>	returns an array of double standard errors, with $2 \times nlag + 1$ entries.
<i>ccfnorm</i>	returns an array of normalized cross-correlation, with $2 \times nlag + 1$ entries.
<i>ccfprob</i>	returns an array of probabilities, with $2 \times nlag + 1$ entries.
<i>ccflprob</i>	returns an array of log probabilities, with $2 \times nlag + 1$ entries.

Example

This example uses the TSMODEL procedure to compute the cross-correlation and cross-covariance of two time series arrays (Price and Sale) with lag 20:

```
data mylib.pricedata (replace=yes);
  set Sashelp.pricedata;
run;

proc tsmodel data=mylib.pricedata outarray=mylib.ccf_array
  outscalar=mylib.ccf_scalar;
  id date interval=month;
  var price sale;
  by region line product;
  outscalars ymu xmu;
  outarrays lags df ccov ccf ccfstd ccf2std ccfnorm ccfprob ccflprob;
  require tsa;
```

```

submit;
declare object TSA(tsa);
rc=TSA.CCF(price, sale, 20, lags, df, ymu, xmu, ccov, ccf,
           ccfst, ccf2std, ccfnorm, ccflprob, ccflprob);
endsubmit;
run;

```

IACF Method

`rc = TSA.IACF (y, nlag, <lags>, <df>, <mu>, <iacf>, <iacfstd>, <iacf2std>, <iacfnorm>, <iacfprob>, <iacflprob>);`

The IACF function computes the inverse autocorrelation for a time series array.

Required Arguments

You must specify the following arguments, separated by a comma:

y specifies the times series array.
nlag specifies the number of the lag to use in the calculation.

Returned Values

The IACF function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

lags returns the number of the lag that was used in the calculation.
df returns the number of observations used to compute *iacf*.
mu returns the mean estimate.
iacf returns an array of inverse autocorrelation estimates, with *nlag* + 1 entries.
iacfstd returns an array of inverse autocorrelation standard errors, with *nlag* + 1 entries.
iacf2std returns an array of twice standard errors, with *nlag* + 1 entries.
iacfnorm returns an array of normalized inverse autocorrelation, with *nlag* + 1 entries.
iacfprob returns an array of inverse autocorrelation probabilities, with *nlag* + 1 entries.
iacflprob returns an array of inverse autocorrelation log probabilities, with *nlag* + 1 entries.

Example

This example uses the TSMODEL procedure to compute the inverse autocorrelation of lag 3 of the time series Air:

```
proc tsmodel data=mylib.air outscalar=mylib.outscalars
      outarray=mylib.outarray;
  id date interval=month;
  var air;
  outscalars mu;
  outarrays iacf lags df iacfstd;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.IACF(air, 3, lags, df, mu, iacf, iacfstd, , , , );
  endsubmit;
run;
```

INTERMITTENCYTEST Method

rc = TSA.INTERMITTENCYTEST (*y*, <base>, <threshold>, <med>);

The INTERMITTENCYTEST function tests for intermittency of a univariate time series by computing the median of the length of contiguous constant periods (demand intervals).

Required Arguments

You must specify the following arguments, separated by commas:

y specifies the times series array to test. The test is applied to the last 100 values.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

base takes a numeric *Value* that specifies the base demand level. If you specify a missing value, the dependent series is analyzed to determine whether the base demand level can be set to one of the following values:

- 0
- the minimum value of the dependent series
- the maximum value of the dependent series
- the median value of the dependent series

First, the analysis determines whether the series is nearly constant by applying a threshold to the maximum (L_∞) norm of the nonmissing observations of the series. For nearly constant series, the base demand level is set to 0. Otherwise, analysis proceeds to compute the minimum, maximum, and median values of the series and assigns a score to each of those values. The

score of a value in this context is the number of observations that are sufficiently close to that value. If the number of missing values in the dependent series is higher than the scores for the minimum, maximum, and median, then the base demand level is set to 0. If scores for the minimum, maximum, and median are identical, then the base demand level is set to 0. Otherwise, the base demand level is set to the value that has the highest score among the minimum, maximum, and median.

threshold specifies the threshold value for intermittency. A value of 1 forces all series to be identified as intermittent, whereas a value of `.inf` (positive infinity) forces all series to be identified as nonintermittent. The default value is 2.

Returned Values

The INTERMITTENCYTEST function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Time series is not intermittent
1	Time series is intermittent
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

med returns the median length of the contiguous constant periods.

Example

This example uses the TSMODEL procedure to test the intermittency on the time series array *Air*:

```
proc tsmodel data=mylib.air outscalar=mylib.outscalars;
  id date interval=month;
  var air;
  outscalars intermittent;
  require tsa;
  submit;
  declare object TSA(tsa);
  intermittent=0;
  rc=TSA.INTERMITTENCYTEST(air, 0, 2, med);
  if rc>0 then intermittent= 1;
  endsubmit;
run;
```

MKTEST Method

```
rc = TSA.MKTEST (y, < seasonality > ,< alpha > , < sum > , < sigma > , < z > , < crit > , < pval > ) ;
```

The MKTEST function runs a Mann-Kendall test or a seasonal Mann-Kendall test.

Required Arguments

You must specify the following arguments, separated by commas:

y specifies the input time series array.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

seasonality specifies the seasonality of the series. You can specify an integer, or you can specify `_SEASONALITY_`. The predefined symbol `_SEASONALITY_` is the length of the seasonal cycle that is computed by using the seasonality or the time ID interval that you specify in a PROC TSMODEL step or a `runTimeCode` action call. For more information about predefined symbols, see the section “Predefined Symbols” in Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*). For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*). The default value is 1.

alpha specifies the significance level of the test. The default value is 0.05.

Returned Values

The MKTEST function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Test Result
1	Series has a monotonic trend
0	Series has no trend
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

sum returns the sum of the signs.

sigma returns the standard deviation of the sum of the signs.

z returns the Z-transformation of the sum of the signs.

crit returns the test critical value of the Z-transformation.
pval returns the p -value.

Details

When the specified seasonality is 1, the MKTEST function runs the Mann-Kendall test on a univariate time series and outputs the corresponding statistics. The Mann-Kendall test is a nonparametric test that is commonly used to detect monotonic trends in series. The null hypothesis, H_0 , is that the data come from a population that has independent realizations and the data are identically distributed. The alternative hypothesis, H_A , is that the data follow a monotonic trend. Given the series y_t , $t = 1, \dots, n$, the Mann-Kendall test statistic S is calculated as

$$S = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \text{sgn}(y_j - y_i)$$

where

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Under the null hypothesis, the mean of S is $E(S) = 0$, and the variance σ^2 is

$$\sigma^2 = (n(n-1)(2n+5) - \sum_{k=1}^p n_k(n_k-1)(2n_k+5))/18$$

where p is the number of tied groups in the data set and n_k is the number of data points in the k th tied group.

When the specified seasonality is greater than 1, the MKTEST function runs a seasonal Mann-Kendall test on a univariate time series. The null hypothesis, H_0 , is that, for each season, the data come from a population that has independent realizations and the data are identically distributed. The alternative hypothesis, H_A , is that the data follow a monotonic trend in one or more seasons. For the g th season,

$$S_g = \sum_{i=1}^{n_g-1} \sum_{j=i+1}^{n_g} \text{sgn}(y_{jg} - y_{ig})$$

and the corresponding variance is

$$\sigma_g^2 = (n_g(n_g-1)(2n_g+5) - \sum_{k=1}^{p_g} n_{kg}(n_{kg}-1)(2n_{kg}+5))/18$$

where p_g is the number of tied groups in the data set and n_{kg} is the number of data points in the k th tied group. The seasonal Mann-Kendall statistic, S , for the entire series of m seasons is calculated according to

$$S = \sum_{g=1}^m S_g$$

The corresponding seasonal variance σ^2 is

$$\sigma^2 = \sum_{g=1}^m \sigma_g^2$$

The following Z-transformation of the statistic S approximately follows a standard normal distribution:

$$Z = \begin{cases} \frac{S-1}{\sigma} & \text{if } S > 0 \\ 0 & \text{if } S = 0 \\ \frac{S+1}{\sigma} & \text{if } S < 0 \end{cases}$$

The test result is obtained by comparing the Z-transformation to the critical value.

The Mann-Kendall test requires at least three nonmissing observations. The constant series is regarded as no trend by the Mann-Kendall test. If any season has fewer than three nonmissing observations, the seasonal Mann-Kendall test ignores it. If no season has enough nonmissing observations, the seasonal Mann-Kendall test cannot be performed. If any season has a constant series, this season is also ignored. If all seasons have constant series, the whole series is regarded as having no trend by the seasonal Mann-Kendall test.

Examples

This example uses the TSMODEL procedure to run the Mann-Kendall test on the time series array Air:

```
proc tsmodel data=mylib.air outscalar = mylib.scalars;
  id date interval=month;
  var air;
  outscalar rc sum sig z crit pval;
  require tsa;
  submit;
  declare object tsa(tsa);
  rc = tsa.mktest(air,1,0.05,sum,sig,z,crit,pval);
  endsubmit;
run;
```

MOVINGSUMMARY Method

```
rc = TSA.MOVINGSUMMARY (y, 'method', k, <lead>, <w>, <'setmiss'>, <'abs'>, x, <p>, <nmiss>)
                        ;
```

The MOVINGSUMMARY function computes statistics for a set of values within a moving time window.

Required Arguments

You must specify the following arguments, separated by commas:

- y specifies the input time series array.
- 'method' specifies the statistic to calculate for each output array, x_t , based on the elements of the y input array in the t window.
You can specify the following methods within single quotation marks:

EWMA	calculates the exponentially weighted moving average.
GMEAN	calculates the moving geometric mean.
MAX	calculates the maximum value.
MEAN	calculates the moving average.
MED	calculates the median value.
MIN	calculates the minimum value.
PROD	calculates the moving product.
RANGE	calculates the maximum value minus the minimum value.
SUM	calculates the moving sum.
TVALUE	calculates the standard deviation divided by the mean.
VAR	calculates the variance of the sample defined by the window around t .

k specifies the window size, where k is a positive integer. When the *method* is EWMA, k is always set to 1, and any other value that you specify is ignored. Defaults are used for all other arguments except w . For any other *method*, k is a required argument, and its value should not be less than 1.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

lead specifies the number of leading terms, where *lead* is a nonnegative integer less than k . You can specify the following values:

0	specifies the backward moving summary.
$k/2$	specifies the centered moving summary.
$k-1$	specifies the forward moving summary.

The default value is 0. When the *method* is EWMA, *lead* is set to 0.

w specifies an array of weights that has k elements when $k > 1$, or specifies a scalar value when $k = 1$. When the *method* is EWMA, this argument must be a scalar between 0 and 1, inclusive. If you omit this value, then by default, EWMA sets w to 1, and the MEAN, PROD, TVALUE, and VAR *methods* do not use any weight. This argument is ignored for the GMEAN, MED, MIN, RANGE, and SUM *methods*.

'*setmiss*' specifies how missing values are interpreted.

You can specify the following values within single quotation marks:

IGNORE	specifies that missing values have no effect on the summary.
MEAN	specifies that missing values are replaced with the mean of the remaining nonmissing values in the window. This value is supported only for the SUM <i>method</i> .

MISSING specifies that if the input window contains a missing value, the output value is also missing.

The default value is IGNORE. When the *method* is EWMA, 'setmiss' is always set to IGNORE, and any other value that you specify is ignored.

'abs' specifies how the series is transformed into nonnegative values prior to performing the moving summary.

You can specify the following values within single quotation marks:

OFF specifies no modification. This value is not supported for the GMEAN *method*, because the geometric mean is undefined for negative values in the series.

ON transforms each member of the series into its absolute value.

SQUARE transforms each member of the series into its square.

The default value is ON when the *method* is GMEAN and is OFF for any other *method*. When the *method* is EWMA, 'abs' is always set to OFF, and any other value that you specify is ignored.

Returned Values

The MOVINGSUMMARY function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
4	Warping limits relaxed
3	Expansion limits relaxed
2	Compression limits relaxed
1	Warping limits imposed
0	Success
< 0	Computational failure

x returns the transformed series.

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

p returns an array in which element *t* is the number of products that contributed to element *t* of *x*. The *p* argument is supported only when *method* is PROD or GMEAN.

nmiss returns the number of missing values that are generated.

Examples

This example uses the TSMODEL procedure to compute the five-period moving average of the time series array Air:

```
proc tsmodel data=mylib.air outscalar=mylib.scalars outarray=mylib.arrays;
  id date interval=month;
  var air;
  outarrays x p;
  outscalars rc nmiss;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.MOVINGSUMMARY(air, 'MEAN' , 5, 0 , , , , x, p, nmiss);
  endsubmit;
run;
```

This example uses the TSMODEL procedure to compute the five-period centered weighted moving product of the time series array Air:

```
proc tsmodel data=mylib.air outscalar=mylib.scalars outarray=mylib.arrays;
  id date interval=month;
  var air;
  outarrays x;
  outscalars rc;
  require tsa;
  submit;
  declare object TSA(tsa);
  Array w[5]/nosymbols; w[1]=0.3; w[2]=0.2; w[3]=0.25; w[4]=0.1; w[5]=0.15;
  rc=TSA.MOVINGSUMMARY(air, 'PROD' , 5, 2.5 , w, , , x, , );
  endsubmit;
run;
```

PACF Method

```
rc = TSA.PACF (y, nlag, <lags>, <df>, <mu>, <pacf>, <pacfstd>, <pacf2std>, <pacfnorm>,
               <pacfprob>, <pacflprob>);
```

The PACF function computes the partial autocorrelation for a time series array.

Required Arguments

You must specify the following arguments, separated by a comma:

- y* specifies the times series array.
- nlag* specifies the number of the lag to use in the calculation.

Returned Values

The PACF function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

lags returns the number of the lag that was used in the calculation.

df returns the number of observations used to compute *pacf*.

mu returns the mean estimate.

pacf returns an array of partial autocorrelation estimates, with $nlag + 1$ entries.

pacfst returns an array of partial autocorrelation standard errors, with $nlag + 1$ entries.

pacf2std returns an array of twice standard errors, with $nlag + 1$ entries.

pacfnorm returns an array of normalized partial autocorrelation, with $nlag + 1$ entries.

pacfprob returns an array of partial autocorrelation probabilities, with $nlag + 1$ entries.

pacflprob returns an array of partial autocorrelation log probabilities, with $nlag + 1$ entries.

Example

This example uses the TSMODEL procedure to compute the autocorrelation of lag 3 of the time series Air:

```
proc tsmodel data=mylib.air outscalar=mylib.outscalars
              outarray=mylib.outarray;
  id date interval=month;
  var air;
  outscalars mu;
  outarrays pacf lags df pacfst;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.PACF(air, 3, lags, df, mu, pacf, pacfst, , , , );
  endsubmit;
run;
```

SCALE Method

$rc = \text{TSA.SCALE}(y, min, max, nomiss, x, < nmiss >);$

The SCALE function scales a time series between a specified minimum value and a specified maximum value.

Required Arguments

You must specify the following arguments, separated by commas:

- y* specifies the input time series array.
- min* specifies the minimum value in the output array.
- max* specifies the maximum value in the output array.
- nomiss* specifies how missing values are treated. You can specify the following values:
 - 0** allows missing values in the input array.
 - 1** does not allow missing values in the input array. If missing values exist, the output array x_t becomes missing for all values of t .

The default is 0.

Returned Values

The SCALE function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
5	The input series is nearly constant
4	Missing values were found when the value 1 was specified for <i>nomiss</i>
3	One or more arguments are ignored
2	One or more arguments are set to the default value
1	The input series is all missing
-1	One or more arguments are not supported
-2	The minimum value of the transformed series is greater than its maximum value
-4	Extreme slope
-99	Bad arguments

- x* returns the transformed series.

Optional Returned Values

You can also specify the following argument to request an additional returned value:

nmiss returns the number of missing values that are generated.

Example

This example uses the TSMODEL procedure to scale the time series array *Air* between a minimum value of 0 and a maximum value of 100:

```
proc tsmodel data=mylib.air outarray=mylib.scale_array;
  id date interval=month;
  var air;
  outarrays t1;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.SCALE(air, 0, 100, , t1, );
  endsubmit;
run;
```

SEASONALDECOMP Method

```
rc = TSA.SEASONALDECOMP (y, s, 'mode', <lambda>, <tcc>, <sic>, <sc>, <scstd>, <tcs>,
                        <ic>, <sa>, <pcsa>, <tc>, <cc>);
```

The SEASONALDECOMP function performs the seasonal decomposition or analysis of the time series arrays, and it outputs the values and results.

Required Arguments

You must specify the following arguments, separated by commas:

y specifies the times series array to decompose.

s specifies the length of the seasonal cycle, where *s* must be either an integer greater than 1 or `_SEASONALITY_`. The predefined symbol `_SEASONALITY_` is the length of the seasonal cycle that is computed by using the seasonality or the time ID interval that you specify in a PROC TSMODEL step or a `runTimeCode` action call. For more information about predefined symbols, see the section “Predefined Symbols” in Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*). For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*).

'mode' specifies the type of decomposition to be used to decompose the time series.

You can specify the following values within single quotation marks:

ADD | ADDITIVE specifies additive decomposition.

LOGADD | LOGADDITIVE specifies log-additive decomposition.

- MULT | MULTIPLICATIVE** specifies multiplicative decomposition.
- MULTORADD** specifies multiplicative or additive decomposition, depending on data.
- PSEUDOADD | PSEUDOADDITIVE** specifies pseudo-additive decomposition.

Optional Arguments

You can also specify the following argument, separated by a comma from arguments that precede it. If you want to use a default value for this argument, enter a space for it.

lambda specifies the Hodrick-Prescott filter parameter for trend-cycle decomposition. The default value is 1,600. Filtering applies when the trend component or the cycle component is requested. If filtering is not specified, this option is ignored.

Returned Values

The SEASONALDECOMP function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

- tcc* specifies the trend-cycle component.
- sic* specifies the seasonal-irregular component.
- sc* specifies the seasonal component.
- scstd* specifies the seasonal component standard errors.
- tcs* specifies the trend-cycle-seasonal component.
- ic* specifies the irregular component.
- sa* specifies the seasonally adjusted series.
- pcsa* specifies the percentage of change in seasonally adjusted series.
- tc* specifies the trend component.
- cc* specifies the cycle component.

Example

This example uses the TSMODEL procedure to compute the seasonal indices on the time series array Air:

```
proc tsmode1 data=mylib.air outarray=mylib.outarray;
  id date interval=month;
  var air;
  outarrays ADJUSTED;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.SEASONALDECOMP(air, _SEASONALITY_, 'ADD', , , , , , , ADJUSTED, , , );
  endsubmit;
run;
```

SEASONALINDICES Method

```
rc = TSA.SEASONALINDICES (y, s, <'mode'>, <'term'>, <indices>);
```

The SEASONALINDICES function computes the seasonal indices of a univariate time series by using regression seasonal dummies.

Required Arguments

You must specify the following arguments, separated by a comma:

- y* specifies the times series array.
- s* specifies the seasonality to test, where *s* must be either a positive integer or `_SEASONALITY_`. The predefined symbol `_SEASONALITY_` is the length of the seasonal cycle that is computed by using the seasonality or the time ID interval that you specify in a PROC TSMODEL step or a `runTimeCode` action call. For more information about predefined symbols, see the section “Predefined Symbols” in Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*). For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*).

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

- `'mode'` specifies the type of model to be used in the regression.
You can specify the following values within single quotation marks:

ADD ADDITIVE	uses an additive model.
MULT MULTIPLICATIVE	uses a multiplicative model.

The default method is ADD.

'term' specifies the type of terms to be used in the regression.

You can specify the following values within single quotation marks:

- S** uses only seasonal dummies terms.
- SC** uses only seasonal dummies and constant terms.
- ST** uses only seasonal dummies and trend terms.
- STC** uses seasonal dummies, trend, and constant terms.
- STQ** uses seasonal dummies, trend, and quadratic terms.
- STQC** uses seasonal dummies, trend, quadratic, and constant terms.

The default value is S. Quadratic values can be used only in the additive model.

Returned Values

The SEASONALINDICES function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

indices returns an array of seasonal indices.

Example

This example uses the TSMODEL procedure to compute the seasonal indices of the time series Air:

```
proc tsmode1 data=mylib.air outarray=mylib.outarray;
  id date interval=month;
  var air;
  outarrays sindices;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.SEASONALINDICES(air, _SEASONALITY_, 'ADD', 'STQC', sindices);
  endsubmit;
run;
```

SEASONTTEST Method

`rc = TSA.SEASONTTEST (y, s, < dif>, < p>, < alpha>, < aic>) ;`

The SEASONTTEST function tests whether a univariate time series is seasonal by comparing two time series models: one seasonal and one nonseasonal.

Required Arguments

You must specify the following arguments, separated by a comma:

- y* specifies the times series array to test.
- s* specifies the seasonality to test, where *s* must be either a positive integer or `_SEASONALITY_`. The predefined symbol `_SEASONALITY_` is the length of the seasonal cycle that is computed by using the seasonality or the time ID interval that you specify in a PROC TSMODEL step or a `runTimeCode` action call. For more information about predefined symbols, see the section “Predefined Symbols” in Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*). For more information about seasonality, see the section “Seasonality in Models” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*).

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

- dif* specifies an array of positive integers or a positive integer that is used for differencing. The default value is 0.
- p* specifies the autoregressive order (0 or 1). The default value is 0.
- alpha* specifies the significance level. The default value is 0.01.

Returned Values

The SEASONTTEST function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Time series is not seasonal
1	Time series is seasonal
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

aic returns an array of three values: Akaike's information criterion (AIC) for the nonseasonal model, AIC for seasonal model, and the p -value for the F test.

Details

The time series y is first predifferenced according to the specified *diff*. The predifferenced time series is fit by a nonseasonal autoregressive model of order p and a seasonal autoregressive model of order p . The AICs of both models and the F statistic of the seasonal model are calculated. If the seasonal model has a smaller AIC, and if the p -value of the F statistic is smaller than the significance level *alpha*, then the time series is identified as seasonal. Otherwise, the time series is identified as nonseasonal. The test is conducted only for series of at least two seasonal cycles. In general, seasonality is more difficult to identify in a series when the sample size is small because there are only a few seasonal cycles of data.

Example

The following example uses the TSMODEL procedure to test the seasonality of the time series array Air:

```
proc tsmodel data=mylib.air outscalar=mylib.outscalars
      outarray=mylib.outarray;
  id date interval=month;
  var air;
  outscalars seasonal;
  outarrays aic;
  require tsa;
  submit;
  declare object TSA(tsa);
  seasonal=0;
  rc=TSA.SEASONTTEST(air, _SEASONALITY_, 0, 1, , aic); /*- no detrending -*/
  if rc>0 then seasonal= 1;
  rc=TSA.SEASONTTEST(air, _SEASONALITY_, 1, 1, 0.05, ); /*- detrending -*/
  if rc>0 then seasonal= 1;
  endsubmit;
run;
```

SIMILARITY Method

`rc = TSA.SIMILARITY (x, y, <'type'>, <'scale'>, <expandpct>, <expandabs>, <compresspct>, <compressabs>, <measure>);`

The SIMILARITY function analyzes the similarity between two time series.

Required Arguments

You must specify the following arguments, separated by a comma:

- `x` specifies the input time series array to be compared to the target time series.
- `y` specifies the target time series array to be compared to the input time series.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

`'type'` specifies the similarity measure.

You can specify the following values within single quotation marks:

- ABSDEV** specifies the absolute deviation.
- MABSDEV** specifies the mean absolute deviation.
- MABSDEVINP** specifies the mean absolute deviation relative to the length of the input sequence.
- MABSDEVMAX** specifies the mean absolute deviation relative to the maximum valid path length.
- MABSDEVMIN** specifies the mean absolute deviation relative to the minimum valid path length.
- MABSDEVTAR** specifies the mean absolute deviation relative to the length of the target sequence.
- MSQRDEV** specifies the mean squared deviation.
- MSQRDEVINP** specifies the mean squared deviation relative to the length of the input sequence.
- MSQRDEVMAX** specifies the mean squared deviation relative to the maximum valid path length.
- MSQRDEVMIN** specifies the mean squared deviation relative to the minimum valid path length.
- MSQRDEVTAR** specifies the mean squared deviation relative to the length of the target sequence.
- SQRDEV** specifies the squared deviation.

The default value is `SQRDEV`.

'*scale*' specifies how the working input sequence is scaled with respect to the working target sequence. Scaling is performed after normalization.

You can specify the following values within single quotation marks:

ABS applies absolute scaling.
NONE applies no scaling.
STD applies standard scaling.

The default value is NONE.

expandpct specifies the warping expansion as a percentage of the length of the target sequence, where *expandpct* ranges from 0 to 100, 0 implies no compression, and 100 implies maximum allowable compression. The default value is 100.

expandabs specifies the absolute warping expansion, where *expandabs* is an integer that ranges from 0 to 10,000. The default is the maximum allowable absolute expansion.

compresspct specifies the warping compression as a percentage of the length of the target sequence, where *compresspct* ranges from 0 to 100, 0 implies no compression, and 100 implies maximum allowable compression. The default value is 100.

compressabs specifies the absolute warping compression, where *compressabs* is an integer that ranges from 0 to 10,000. The default is the maximum allowable absolute compression.

Returned Values

The SIMILARITY function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
4	Warping limits relaxed
3	Expansion limits relaxed
2	Compression limits relaxed
1	Warping limits imposed
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

measure returns the similarity measure.

Example

This example uses the TSMODEL procedure to compute the similarity of two time series arrays: x and y.

```
data test;
  input i x1 x2 x3 y1 y2 y3 r;
  datalines;
  1 3 2 4 2 3 2 1
  2 5 4 5 4 5 3 1
  3 3 3 4 6 4 5 1
  4 3 6 6 7 6 7 1
  5 3 5 5 3 5 7 1
  6 6 6 6 8 8 8 1
  7 3 8 5 9 9 8 1
  8 8 9 8 3 7 3 1
  9 6 7 6 8 4 9 1
  10 7 9 8 9 6 7 1
;
run;

data mylib.testsim;
  set test;
run;

proc tsmodel data=mylib.testsim
  outscalar=mylib.sim_scalar
  nthreads=1;
  require tsa;
  id i interval=day;
  var x1 y1;
  outscalars measure;
  submit;
  declare object TSA(tsa);
  rc = TSA.SIMILARITY(x1, y1, 'absdev', 'NONE', , , , measure);
  endsubmit;
run;
```

STATIONARITYTEST Method

```
rc = TSA.STATIONARITYTEST (y, <dif>, <d>, <p>, <'type'>, <pvalue>);
```

The STATIONARITYTEST function tests for stationarity of a univariate time series.

Required Arguments

You must specify the following argument:

y specifies the times series array to test.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

- dif* specifies an array of positive integers or a positive integer that is used for differencing. The default value is 0.
- d* specifies the order of unit root ($d = 1, \dots, 12$). If the *type* is SSM, then $d = 1$. The default value is 1.
- p* specifies the autoregressive order, where *p* must be a nonnegative integer. The default value is 5.
- '*type*' specifies the type of test statistic used.

You can specify the following values within single quotation marks:

- SSM** specifies the studentized test statistic for the single mean (intercept) case.
- STR** specifies the studentized test statistic for the deterministic time trend case.
- SZM** specifies the studentized test statistic for the zero mean (no intercept) case. This value is allowed only when *d* has a value of 1.

The default value of *type* is SZM.

Returned Values

The STATIONARITYTEST function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Time series is stationary with the default significance level of 0.05
1	Time series is not stationary with the default significance level of 0.05
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

- pvalue* returns the probability value associated with the test.

Example

This example uses the TSMODEL procedure to test the stationarity on the time series array Air:

```
proc tsmode1 data=mylib.air outscalar=mylib.outscalars;
  id date interval=month;
  var air;
  outscalars stationary1 stationary2;
  require tsa;
  submit;
  declare object TSA(tsa);
  stationary1=1; stationary2=1;
  rc = TSA.STATIONARITYTEST(air,,,,,pvalue);
  *test with the default significant level=0.05;
  if rc =1 then stationary1 = 0;
  *test with significant level = 0.1;
  if pvalue > 0.1 then stationary2 = 0;
  endsubmit;
run;
```

TRANSFORM Method

```
rc = TSA.TRANSFORM (y, <'type'>, <inverse>, <c>, <x>);
```

The TRANSFORM function transforms a time series to another form.

Required Arguments

You must specify the following arguments:

y specifies an input time series array.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

type specifies the type of transformation. You can specify the following values within single quotation marks:

LOG	specifies logarithmic transformation.
SQRT	specifies square root transformation.
LOGIT	specifies logit transformation.
BOXCOX	specifies Box-Cox transformation.
NONE	requests that no transformation be performed.

The default value is NONE.

inverse specifies whether to perform an inverse transformation. You can specify the following values:

- 0** does not perform an inverse transformation.
- 1** returns the inverse of the specified transformation method.

The default value is 0.

- c*** specifies a parameter to be used in the transformation. Its use depends on the transformation method as follows:
- For log transformation, ***c*** is bias: $x = \log(y + c)$. The default value is 0.
 - For square root transformation, ***c*** is bias: $x = \sqrt{y + c}$. The default value is 0.
 - For logit transformation, ***c*** is scaling: $x = \log(c \times y / (1 - (c \times y)))$. The default value is 1.
 - For the Box-Cox transformation, ***c*** is λ : $x = c^2 + (y^c - 1)/c$. If ***c*** is not specified, $x=y$.

Returned Values

The TRANSFORM function returns the following values:

- rc*** returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

- x*** returns the transformed series.

Example

This example uses the TSMODEL procedure to take the log transform of the time series array **Air**:

```
proc tsmode1 data=mylib.air outarray=mylib.trans_array;
  id date interval=month;
  var air;
  outarrays t1;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.TRANSFORM(air, 'LOG', 0, 0, t1);
  endsubmit;
run;
```

UNBIASEDNESS Method

***rc* = TSA.UNBIASEDNESS (*y*, *predict*, < *siglevel*>, *intercept*, *scale*, *fvalue*, *pvalue*) ;**

The UNBIASEDNESS function tests whether a univariate time series is unbiased.

Required Arguments

You must specify the following arguments, separated by a comma:

y specifies the input time series array.
predict specifies an input array of predicted time series.

Optional Arguments

You can also specify the following argument. If you want to use a default value for this argument, enter a space for it.

siglevel specifies the significance level.

Returned Values

The UNBIASEDNESS function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
1	Biased predictions
0	Unbiased predictions
-1	Degree of freedom error
-2	Singular system
-3	Extreme value

intercept returns the constant parameter.

scale returns the scale parameter.

fvalue returns the test statistic for the *F* test.

pvalue returns the *p*-value for the *F* test.

Example

This example uses the TSMODEL procedure to test whether the series Actual is unbiased:

```
proc hpf data=sashelp.air out=_null_ outfor=outfor;
  id date interval=month;
  forecast air;
run;

proc reg data=outfor;
  model actual=predict;
  test intercept=0, predict=1;
run;
quit;
data mylib.outfor;
  set outfor;
run;

proc tsmodel data=mylib.outfor outscalar=mylib.bias_scalar outarray=mylib.bias_array;
  id date interval=month;
  var ACTUAL PREDICT;
  outscalars intercept scale fvalue pvalue;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.UNBIASEDNESS(ACTUAL, PREDICT, 0.05, intercept, scale, fvalue, pvalue);
  endsubmit;
run;
```

WHITENOISE Method

`rc = TSA.WHITENOISE (y, nlag, <lags>, <df>, <wn>, <wnprob>, <wnlprob>);`

The WHITENOISE function tests for white noise in a time series array.

Required Arguments

You must specify the following arguments, separated by a comma:

- y* specifies the times series array to compute.
- nlag* specifies the number of the lag to use in the calculation.

Returned Values

The WHITENOISE function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas, to request additional returned values. If you do not want the value to be returned, enter a space for it.

<i>lags</i>	returns the number of the lag that was used in the calculation.
<i>df</i>	returns the number of observations that were used to test white noise.
<i>wn</i>	returns an array of Ljung-Box white noise tests, with <i>nlag</i> + 1 entries.
<i>wnprob</i>	returns white noise probabilities.
<i>wnlprob</i>	returns white noise log probabilities.

Example

This example uses the TSMODEL procedure to perform the white noise test of lag 3 of the time series Air:

```
proc tsmodel data=mylib.air outarray=mylib.outarray;
  id date interval=month;
  var air;
  outarrays lags df wn wnprob wnlprob;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.WHITENOISE(air, 3, lags, df, wn, wnprob, wnlprob);
  endsubmit;
run;
```

References

- Brockwell, P. J., and Davis, R. A. (1991). *Time Series: Theory and Methods*. 2nd ed. New York: Springer-Verlag.
- Cooley, J. W., and Tukey, J. W. (1965). "An Algorithm for the Machine Calculation of Complex Fourier Series." *Mathematics of Computation* 19:297–301.
- Hodrick, R. J., and Prescott, E. C. (1980). "Post-war U.S. Business Cycles: An Empirical Investigation." Discussion Paper 451, Carnegie Mellon University.
- Monro, D. M., and Branch, J. L. (1977). "Algorithm AS 117: The Chirp Discrete Fourier Transform of General Length." *Journal of the Royal Statistical Society, Series C* 26:351–361.
- Priestley, M. B. (1981). *Spectral Analysis and Time Series*. London: Academic Press.
- Singleton, R. C. (1969). "An Algorithm for Computing the Mixed Radix Fast Fourier Transform." *IEEE Transactions on Audio and Electroacoustics* 17:93–103.

Chapter 32

Time Series Model Package

Contents

Overview: TSM Package	2340
TSM Package Summary	2341
Using the TSM Package	2344
Return Codes	2346
TSM Object	2346
TSM Synopsis	2348
TSM Methods	2348
CFC Object	2355
CFC Synopsis	2358
CFC Methods	2358
ESMSPEC Object	2364
ESMSPEC Synopsis	2365
ESMSPEC Methods	2366
ARIMASPEC Object	2369
ARIMASPEC Synopsis	2372
ARIMASPEC Methods	2372
EXMSPEC Object	2380
EXMSPEC Synopsis	2381
EXMSPEC Methods	2382
IDMSPEC Object	2384
IDMSPEC Synopsis	2385
IDMSPEC Methods	2386
RNNSPEC Object	2390
RNNSPEC Synopsis	2391
RNNSPEC Methods	2392
UCMSPEC Object	2398
UCMSPEC Synopsis	2399
UCMSPEC Methods	2400
TSMPEST Object	2408
TSMPEST Synopsis	2409
TSMPEST Methods	2409
TSMSPEC Object	2410
TSMSPEC Synopsis	2411
TSMSPEC Methods	2411
TSMFOR Object	2411

TSMFOR Synopsis	2413
TSMFOR Methods	2414
TSMSTAT Object	2415
TSMSTAT Synopsis	2419
TSMSTAT Methods	2419
TSMINEST Object	2421
TSMINEST Synopsis	2421
TSMINEST Methods	2421
TSMINSPEC Object	2422
TSMINSPEC Synopsis	2422
TSMINSPEC Methods	2422
TSMOUTOPT Object	2423
TSMOUTOPT Synopsis	2424
TSMOUTOPT Methods	2424
TSMRNNSTAT Object	2424
TSMRNNSTAT Synopsis	2425
TSMRNNSTAT Methods	2425
Examples: TSM Package	2426
Using CAS Sessions and CAS Engine Librefs	2426
Example 32.1: Fitting and Forecasting with ARIMA and ESM Models	2427
Example 32.2: Fitting a Transfer Function Model	2430
Example 32.3: Replaying a Previously Fitted Model	2432
Example 32.4: Performing Time Series Imputation Using an ARIMA Model	2434
Example 32.5: Combining Forecasts	2438
Example 32.6: Recurrent Neural Network Forecasting	2440
References	2443

Overview: TSM Package

The time series model (TSM) package contains a set of time series modeling objects that provide a flexible way to model and forecast time series. For more information about the statistical methodology that underlies this package, see relevant chapters in *SAS/ETS User's Guide* and *SAS Forecast Server Procedures: User's Guide*.

TSM Package Summary

Table 32.1 lists the objects that are contained in the TSM package. These objects are designed to provide access to various univariate time series model families.

Table 32.1 Objects in the Time Series Model Package

Object	Description
Central Objects	
TSM	Time series model object (Note 1)
CFC	Combine Forecasts object (Note 2)
Time Series Model Specification Objects (Note 3)	
ARIMASPEC	Autoregressive integrated moving average model specification object
ESMSPEC	Exponential smoothing model specification object
EXMSPEC	External model specification object
IDMSPEC	Intermittent demand model specification object
RNNSPEC	Recurrent neural network (RNN) specification object
UCMSPEC	Unobserved component model (UCM) specification object
Time Series Model Collector Objects (Note 4)	
TSMFOR	Time series model forecast collector object
TSMOUTOPT	Time series RNN model optimization history collector object
TSMPEST	Time series model parameter estimates collector object
TSMRNNSTAT	Time series RNN model fit statistics collector object
TSMSPEC	Time series model specification collector object
TSMSTAT	Time series model fit statistics collector object
Time Series Model Repeater Objects (Note 5)	
TSMINEST	Time series model input parameter estimates repeater object
TSMINSPEC	Time series model input specification repeater object

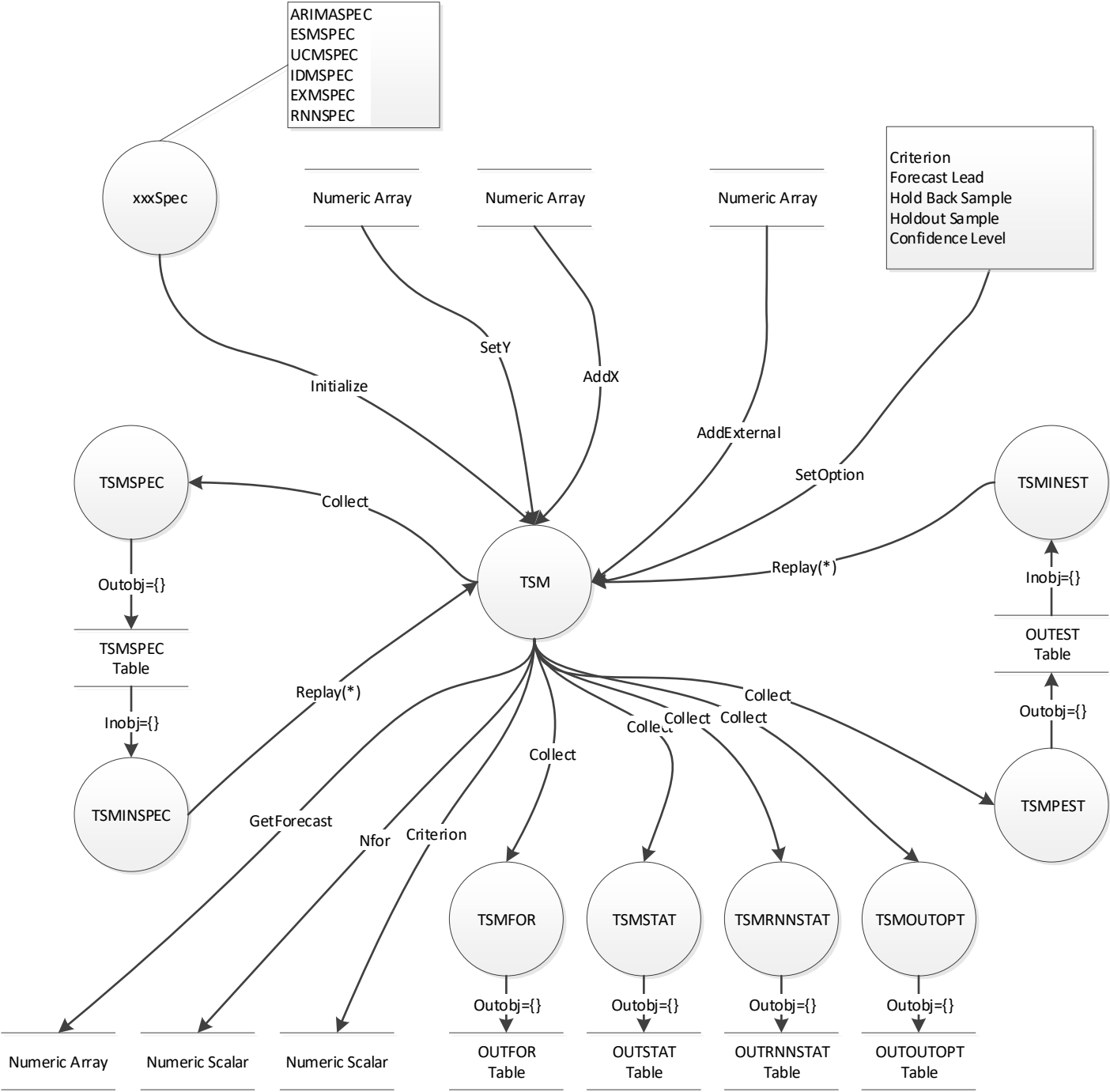
NOTE:

1. The TSM object is the central hub that interacts with all other objects. It executes and encapsulates the computational services for all univariate time series models.
2. The Combine Forecasts (CFC) object allows you to perform forecast combinations.
3. The time series model specification objects enable you to specify the characteristics of a time series model to be executed by a TSM object.

4. The time series model collector objects operate on TSM objects to collect and save various results from the time series model execution (fit statistics, forecasts, parameter estimates, model specifications, and so on).
5. The time series model repeater objects act as conduits to replay collected model specifications and model parameter estimates as input to other TSM objects.

Figure 32.1 diagrams the relationships among the objects in the TSM package.

Figure 32.1 TSM Object Data Flow



Using the TSM Package

The following steps provide a general outline of how to use each type of object in the TSM package. Subsequent sections describe each step in greater detail.

- 1 Configure a model specification object.
- 2 Use the TSM object with the model specification object to generate a forecast.
- 3 Use TSM collector objects to extract results and parameter estimates from the TSM object.
- 4 Use TSM repeater objects to replay collected parameters and specifications to another TSM object.

Step 1: Configure a Model Specification Object

Model specification objects define the time series model characteristics that you want to be applied to the TSM object, which then performs the model execution. Model specification objects use **Open** and **Close** methods to initialize and finalize model specifications. The basic execution pattern follows this sequence of operations:

- 1 **Declare:** Create the model specification object by using the object declaration statement. The object declaration assigns a default model specification to the specification objects.
- 2 **Open:** Initialize the specification object to a default state that is ready to accept configuration methods that shape the model and define its characteristics.
- 3 **Configure:** Use object-specific model specification methods to configure the model. For example, in an ESMSPEC, you might specify the smoothing method or model to be used, specify a functional transformation for the dependent variable to force the use of specific smoothing parameters, or specify bounds on the estimated parameters. For an ARIMASPEC, you might specify a set of autoregressive or moving average backshift operator polynomial factors, specify differencing operators, add simple and complex transfer functions, or specify a functional transformation for the dependent variable.
- 4 **Close:** Declare the model specification object to be complete and ready for use.
- 5 **Use:** Use the completed model specification object with a TSM object to directly perform the specified time series model fit and forecast computations. You can also use model specification objects with the automatic time series modeling (ATSM) package to include custom models into its automatic time series forecasting process. For more information about this use, see Chapter 2, “Automatic Time Series Modeling Package” (*SAS Visual Forecasting: Time Series Packages*).

Step 2: Use a TSM Object with a Model Specification Object

The TSM object executes the time series model. The TSM object is configured with a time series model from one of the model specification objects. Then the TSM object applies that configuration to the time series data to produce a forecast. The basic execution pattern follows this sequence of operations:

- 1 Declare:** Create the TSM object by using the object declaration statement.
- 2 Initialize:** Add a model specification object to the TSM object.
- 3 Specify variables:** Specify the dependent series (Y) and any independent series (X) variables.
- 4 Specify options:** Specify other options and properties as appropriate.
- 5 Run:** Execute the model in the TSM object to produce its forecast.
- 6 Extract:** Extract the results by using collector objects.

Various properties (attributes) of the executed TSM object can be queried directly and saved into declared variables, or the results can be collected by the TSM collector objects for presentation and storage.

Step 3: Use the TSM Collector Objects

Collector objects enable you to create a snapshot of results from TSM objects and store those results in tables. Each collector object defines a table schema that is determined by the collector object's design. The TSM collector objects follow a common pattern. The basic execution follows this sequence of operations:

- 1 Declare:** Create the collector object using the object declaration statement.
- 2 Collect:** Use the Collect method with a TSM object passed in as an argument to collect results from the TSM object. For example, the TSMPEST object collects parameter estimates from a TSM object, and the TSMSPEC object collects from a model specification's XML. Rows that are collected are automatically appended to the collector's associated table at the end of each BY group, and the collector object's saved row set is automatically reset. Rows that are added to the table are qualified by the values of the corresponding values of the BY variable. This enables repeater objects to locate the rows that are relevant to each BY group and correctly replay that information into a TSM object. The **nrows** attribute returns the current row count in the collector. A missing value is returned if nothing has been collected. The data can now be used for reports or used by a repeater object on another model.

Step 4: Use the TSM Repeater Objects

Repeater objects read rows from a table and convert their contents back into useful information that can be used by other TSM objects. This is the inverse function of the collector objects. Each repeater object defines a table schema that is determined by its counterpart collector object's design. Repeater objects must be associated with an existing table that has the table schema that is required by the repeater object. The basic execution follows this sequence of events:

- 1 Declare:** Create the repeater object by using the object declaration statement.

- 2 Replay:** Use the `TSM.Replay` method on a TSM object with the repeater object passed in as an argument to execute a new time series model. The replayed TSM object generates the forecast data that was produced by the new time series model. Then the data can be collected by using a collector object as described in Step 3.

Return Codes

Table 32.2 shows the return code (designated by *rc* in method statements) status values that are used in this package. These status code values are returned after a method that is associated with an object is called; they can help determine whether the method executed successfully.

Table 32.2 Return Codes

Status	Meaning
< 0	An unrecoverable error occurred. No result was produced.
= 0	Unconditional success. The requested action was completed, and a normal result was produced.
> 0	Conditional success or warning. A result was produced subject to conditions.

TSM Object

The TSM object generates forecasts of univariate time series.

Table 32.3 lists the time series model families that are supported.

Table 32.3 Model Families for the TSM Object

Family	Object	Description
ARIMA	ARIMASPEC	ARIMAX models
ESM	ESMSPEC	Exponential smoothing models
EXM	EXMSPEC	External model (external forecast)
IDM	IDMSPEC	Intermittent demand models (Croston's/average demand (ADEM))
RNN	RNNSPEC	Recurrent neural network models
UCM	UCMSPEC	Unobserved component models

Table 32.4 summarizes the methods that are associated with the TSM object.

Table 32.4 Methods of the TSM Object

Method	Description
AddX	Add an independent time series array (<i>XSeries</i>) for the TSM object
AddExternal	Add external forecast component series for the TSM object
criterion	Return the final fit statistic over a specified forecast region
GetForecast	Get the forecast series
Initialize	Initialize the TSM object
nfor	Return the forecast series length
Replay	Replay the restored model and parameter estimates
Run	Run the TSM object
SetOption	Specify the named option for the TSM object
SetY	Specify the dependent time series array (<i>YSeries</i>) for the TSM object

The basic execution pattern for using a TSM object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new TSM object.
- 2 Initialize:** The TSM.Initialize method takes a model specification object as its argument and initializes the TSM object for that specified time series model. If no model specification object is provided, the TSM object is initialized as an exponential smoothing method (ESM) that uses the best suited exponential smoothing model. This is equivalent to initializing the TSM object with an ESMSPEC object that has default option values.
- 3 SetY:** The TSM.SetY method defines the dependent time series for the TSM object.
- 4 AddX:** The TSM.AddX method defines any independent time series for the TSM object. Each call defines one predictor series. Repeat as needed for each predictor series.
- 5 SetOption:** The TSM.SetOption method specifies any options that affect the running of the model. Each call defines an option. Repeat as needed to specify all options that are required.
- 6 Run:** The TSM.Run method uses its currently configured X and Y time series data to execute the time series model that is defined by the TSM object's model specification. At completion, the model has estimated the parameters and produced a final forecast based on these parameters.

TSM Synopsis

DECLARE OBJECT *obj* (TSM) ;

Method syntax, in order of typical usage:

```
rc=obj.Initialize (< ModelSpec >) ;
rc=obj.SetY (YSeries) ;
rc=obj.AddX (XSeries < ,Required,NoDiff,ModelSymbol >) ;
rc=obj.AddExternal (Series < ,Role >) ;
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
rc=obj.Replay (TSMINSPECObj < ,TSMINESTObj >) ;
rc=obj.Run () ;
rc=obj.GetForecast (Which, Result) ;
nfor=obj.nfor () ;
criterion=obj.criterion (Region) ;
```

TSM Methods

TSM.AddExternal Method

```
rc=obj.AddExternal (Series < ,Role >) ;
```

Adds a time series array, *Series*, for use in external model computations when the TSM object is initialized from an external model specification (EXMSPEC) object. Calling this method when the TSM object is not configured with an EXMSPEC results in a nonzero return code and no further action. However, the specified parameters are saved so that you can configure the TSM object with an EXMSPEC by using the TSM.Replay method. In that case, the parameters that you specify in this TSM.AddExternal call are processed when you call the Run method. For more information about external model support, see the section “EXMSPEC Object” on page 2380.

No default role mapping is implied by the name of the *Series* variable that you specify in the method. Each call to the AddExternal method adds the specified series to the TSM object according to its role in the external model. That role association happens in one of the following ways:

- You specify the *Role* argument in the method call. This takes precedence over any role mapping that is defined in the EXMSPEC that was used in the TSM.Initialize method. If you specify an invalid *Role* value, an error occurs.
- You specify a *Series* variable that matches a role mapping in the EXMSPEC object that was used in the TSM.Initialize method. If you specify a *Series* variable that fails to match a role mapping in the EXMSPEC, an error occurs.

This method can be called as many times as needed to specify all of the external series that are required to run the external model. In all cases, if the series that you specify to add fails to resolve to a role in the EXMSPEC object, an error occurs without the series being included in the TSM object. Such failures do not cause subsequent TSM.AddExternal method calls to fail.

Input Arguments

You must specify the following input arguments:

<i>Series</i>	specifies a numeric array that contains an external forecast series for the TSM object.
<i>Role</i>	is a case-sensitive character string that specifies the role of the external forecast series in the external model. You can specify one of the following values:
ERROR	returns prediction errors.
LOWER	returns a lower confidence limit series.
STDERR	returns a prediction standard error series.
PREDICT	returns a prediction series.
UPPER	returns an upper confidence limit series.

TSM.AddX Method

```
rc=obj.AddX (XSeries < ,Required,NoDiff,ModelSymbol> ) ;
```

Adds an independent time series array (*XSeries*) for the TSM object. Each call of the TSM.AddX method adds the specified *XSeries* array variable to the TSM object. This method can be called as many times as needed to specify all the independent variables. By default, the name of the *XSeries* variable must match the name of an input symbol in the model specification that is used to configure the TSM object. You can specify a symbol name, *ModelSymbol*, to associate an *XSeries* array with an input symbol in the model specification. ARIMA, RNN, and UCM models support *XSeries* predictors, but RNN models ignore the optional arguments.

If you add multiple independent series, you must not use the same *XSeries* array variable for more than one series. For example, if you retrieve the values of independent variables by using the GetSeries method of the Time Series Data Frame (TSDF) object in the Automatic Time Series Modeling (ATSM) package, then you must use different arrays to store their values. The following programming statements illustrate this requirement by storing values of two independent variables, x1 and x2, in the arrays xarr1 and xarr2, respectively:

```
/* Create an instance of the TSDF object */
declare object df(tsdf);
rc = df.Initialize();
rc = df.AddY(sale);
rc = df.AddX(x1);
rc = df.AddX(x2);
rc = df.SetOption('lead',12);

/* Create an instance of the model object */
declare object ucm(ucmspec);
rc = ucm.Open();
rc = ucm.AddComponent('level',0,1);
rc = ucm.AddComponent('slope',0,1);
rc = ucm.AddComponent('irregular',.,0);
rc = ucm.AddInput('x1');
rc = ucm.AddInput('x2');
rc = ucm.Close();
```

```

/* Create an instance of the TSM object */
declare object tsm(tsm);
rc = tsm.Initialize(ucm);
rc = tsm.SetY(sale);

/* Store x1 and x2 values in separate arrays xarr1 and xarr2 */
array xarr1[1] / nosymbols;
call dynamic_array(xarr1, _LENGTH_);
rc = d.GetSeries('x1',xarr1,'adjust','yes');

array xarr2[1] / nosymbols;
call dynamic_array(xarr2, _LENGTH_);
rc = d.GetSeries('x2',xarr2,'adjust','yes');

/* Add x1 and x2 as independent variables to TSM instance */
rc = tsm.AddX(xarr1,0,0,'x1');
rc = tsm.AddX(xarr2,0,0,'x2');

/* Run the model */
rc = tsm.run();

```

Input Arguments

You must specify the following input argument:

XSeries specifies a numeric array that contains an independent series for the TSM object.

You can also specify the following input arguments:

<i>Required</i>	takes a Boolean value (0 or 1) that, when set to 1, specifies that the <i>XSeries</i> variable is required to be in the model. This might cause the model estimation to fail if the <i>XSeries</i> array variable is deemed inadmissible for inclusion in the underlying model. The default value is 0.
<i>NoDiff</i>	takes a Boolean value (0 or 1) that, when set to 1, specifies that the <i>XSeries</i> variable does not automatically follow the <i>XSeries</i> variable differencing. The default value is 0.
<i>ModelSymbol</i>	takes a character variable that specifies the name of an input symbol in the model specification to be associated with the <i>XSeries</i> variable when the model is run. By default, the TSM.AddX method adds <i>XSeries</i> variables to a model specification if they are not already referenced as an input symbol. Specification of <i>ModelSymbol</i> defines a way to include the <i>XSeries</i> variable if the model specification object includes a symbol that matches the <i>ModelSymbol</i> argument.

TSM.criterion Method

*criterion=*obj.criterion (*Region*) ;

Returns the fit statistic value over the specified forecast *Region* for the TSM object. The criterion is set via the 'CRITERION' argument in the “TSM.SetOption Method” on page 2353. A missing value indicates that the TSM object instance has not produced a successful forecast.

Input Arguments

You must specify the following input argument:

<i>Region</i>	is a case-insensitive character string that specifies the forecast region over which the fit statistic is computed. You can specify one of the following values:
BACK	returns the fit statistic over the time region that is subsequent to the FIT region and that did not contribute any data for estimating model parameters (that is, the model forecast region).
FIT	returns the fit statistic over the time region that supplied observations for estimating model parameters (that is, the model fit region).

TSM.GetForecast Method

*rc=*obj.GetForecast (*Which*, *Result*) ;

Gets the specified forecast series (*Which*) from the TSM object and stores it in the specified numeric array (*Result*).

Input Arguments

You must specify the following input argument:

<i>Which</i>	is a case-insensitive character string that specifies the type of forecast series to return. You can specify one of the following values:
ERROR	returns prediction errors.
LOWER	returns a lower confidence limit series.
STDERR	returns a prediction standard error series.
PREDICT	returns a prediction series.
UPPER	returns an upper confidence limit series.

Output Arguments

You must specify the following output argument:

<i>Result</i>	specifies a numeric array to receive the forecast series.
---------------	---

TSM.Initialize Method

```
rc=obj.Initialize (< ModelSpec>) ;
```

Initializes a TSM object to use the specified *ModelSpec*. This method must be called before the time series arrays (*XSeries* and *YSeries*) and other attributes for the TSM object are specified. If no *ModelSpec* object is specified, the TSM object is initialized to use the default ESM specification. This is equivalent to initializing the TSM object with an ESMSPEC object that has default option values.

Input Arguments

You can specify the following input argument:

ModelSpec specifies an optional name for a TSM model specification object that is used to configure the TSM object.

TSM.nfor Method

```
nfor=obj.nfor () ;
```

Returns the length (observation count) of the forecast series for the TSM object. A missing value indicates that the TSM object has not produced a successful forecast.

Arguments

There are no arguments associated with this method.

TSM.Replay Method

```
rc=obj.Replay (TSMINSPECObj< ,TSMINESTObj>) ;
```

Uses a previously saved model specification from a *TSMINSPECObj* as input to another TSM object. Optionally, restored parameter estimates from *TSMINESTObj* are applied to the model specification.

Input Arguments

You must specify the following input argument:

TSMINSPECObj specifies the TSMINSPEC object to supply the model specification.

You can also specify the following input argument:

TSMINESTObj specifies the TSMINEST object to supply the model's parameter estimates.

TSM.Run Method

```
rc=obj.Run () ;
```

Runs the TSM object to estimate and forecast the time series model by using the specified dependent (*YSeries*) and independent (*XSeries*) series. Upon successful completion, various results can be extracted from the TSM object. The Run method also attempts to process any TSM.AddExternal calls that were deferred because of the TSM object not being configured with an EXMSPEC. An error occurs at this point if the TSM object is still not configured with an EXMSPEC but you had specified TSM.AddExternal method calls.

Arguments

There are no arguments associated with this method.

TSM.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... > ) ;
```

Specifies the named options for the TSM object. When you invoke this method, the previous forecast produced by the [TSM.Run](#) method is discarded. You must then rerun the [TSM.Run](#) method in order to produce a new forecast using the updated configuration options.

Input Arguments

You must specify at least one of the following *'Names'* and its associated *Value*:

'ALPHA'	takes a numeric <i>Value</i> between 0 and 1, exclusive, that specifies the significance level for forecast confidence bands. The default value is 0.05.
'BACK'	takes a nonnegative integer <i>Value</i> that specifies the back region for model performance. If <i>'BACK'=n</i> and the number of observations is <i>T</i> , then the first $T - n$ observations are used to diagnose a series. The default is value 0.
'CRITERION'	takes a string <i>Value</i> that specifies the model selection criterion (statistic of fit) to be used to select from several candidate models. You can specify one of the following <i>Values</i> :
AADJRSQ	specifies Amemiya's adjusted R-square.
ADJRSQ	specifies the adjusted R-square.
AIC	specifies Akaike's information criterion.
AICC	specifies the finite-sample corrected AIC.
APC	specifies Amemiya's prediction criterion.
GMAPE	specifies the geometric mean absolute percentage error.
GMAPES	specifies the geometric mean absolute error as a percentage of standard deviation.
GMAPPE	specifies the geometric mean absolute predictive percentage error.
GMASPE	specifies the geometric mean absolute symmetric percentage error.
GMRAE	specifies the geometric mean relative absolute error.
IMASE	specifies the in-sample mean absolute scaled error.
MAE	specifies the mean absolute error.
MAPE	specifies the mean absolute percentage error.
MAPES	specifies the mean absolute error as a percentage of standard deviation.
MAPPE	specifies the symmetric mean absolute predictive percentage error.
MASE	specifies the mean absolute scaled error.
MAXERR	specifies the maximum error.
MAXPE	specifies the maximum percentage error.
MAXPPE	specifies the maximum predictive percentage error.

MAXRE	specifies the maximum relative error.
MAXSPE	specifies the maximum symmetric percentage error.
MDAPE	specifies the median absolute percentage error.
MDAPES	specifies the median absolute error as a percentage of standard deviation.
MDAPPE	specifies the median absolute predictive percentage error.
MDRAE	specifies the median relative absolute error.
MDASPE	specifies median absolute symmetric percentage error.
ME	specifies the mean error.
MINERR	specifies the minimum error.
MINPE	specifies the minimum percentage error.
MINPPE	specifies the minimum predictive percentage error.
MINRE	specifies the minimum relative error.
MINSPE	specifies the minimum symmetric percentage error.
MPE	specifies the mean percentage error.
MPPE	specifies the mean predictive percentage error.
MRAE	specifies the mean relative absolute error.
MRE	specifies the mean error.
MSE	specifies the mean square error.
MSPE	specifies the mean symmetric percentage error.
RMSE	specifies the root mean square error.
RMSSE	specifies the root mean square scaled error.
RSQUARE	specifies the R-square.
RWRSQ	specifies the random walk R-square.
SMAPE	specifies the symmetric mean absolute percentage error.
SBC	specifies the Schwarz Bayesian information criterion.
SSE	specifies the sum of square error.
UMSE	specifies the unbiased mean square error.
URMSE	specifies the unbiased root mean square error.

The default value is RMSE.

'HOLDOUT' takes a nonnegative integer *Value* that specifies the holdout region for model selection. The holdout sample is a subset of dependent series (which you specify by using the `TSM.SetY` method) that ends at the last nonmissing observation. This option is relevant only when a TSM instance has been initialized via the [Initialize](#) method by using a model specification that requires model selection. Currently, this applies only to a model specification that is created by an [ESMSPEC](#) object and whose 'METHOD' option is set to either 'BEST', 'BESTN', or 'BESTS' via the [ESMSPEC.SetOption](#) method. The default value is 0.

'HORIZON'	takes a numeric <i>Value</i> that specifies the forecast horizon reference time. When set to a missing value, the forecast horizon reference time is automatically set as the first time period that follows the last nonmissing observation of the dependent series. The default value is a missing value. Currently, this option does not apply to a model specification that is created by an RNNSPEC object.
'LEAD'	takes a nonnegative integer <i>Value</i> that specifies the forecast lead. The default value is the value specified in the PROC TSMODEL statement (LEAD= option). If LEAD= is not specified in the PROC TSMODEL statement, the default value of LEAD=0.
'VERBOSE'	takes a Boolean <i>Value</i> (0 or 1) that, when set to 1, enables the TSM.Run method to generate additional log messages when issues occur during the process of fitting a specified model to the input data. These additional messages can help you identify the cause of model fitting failures, such as the specification of invalid weight bounds for model parameters. The default is 0 (no additional log messages are generated).

TSM.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the dependent time series array (*YSeries*) for the TSM object.

Input Arguments

You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the TSM object.

CFC Object

The CFC object creates optimized combinations of time series model forecasts. The performance of combined model forecasts tends to improve over that of individual models. You can specify the model forecasts to be combined either as arbitrary time series arrays or as instances of the TSM object that have been previously run. [Table 32.5](#) summarizes the methods that are associated with the CFC object.

Table 32.5 Methods of the CFC Object

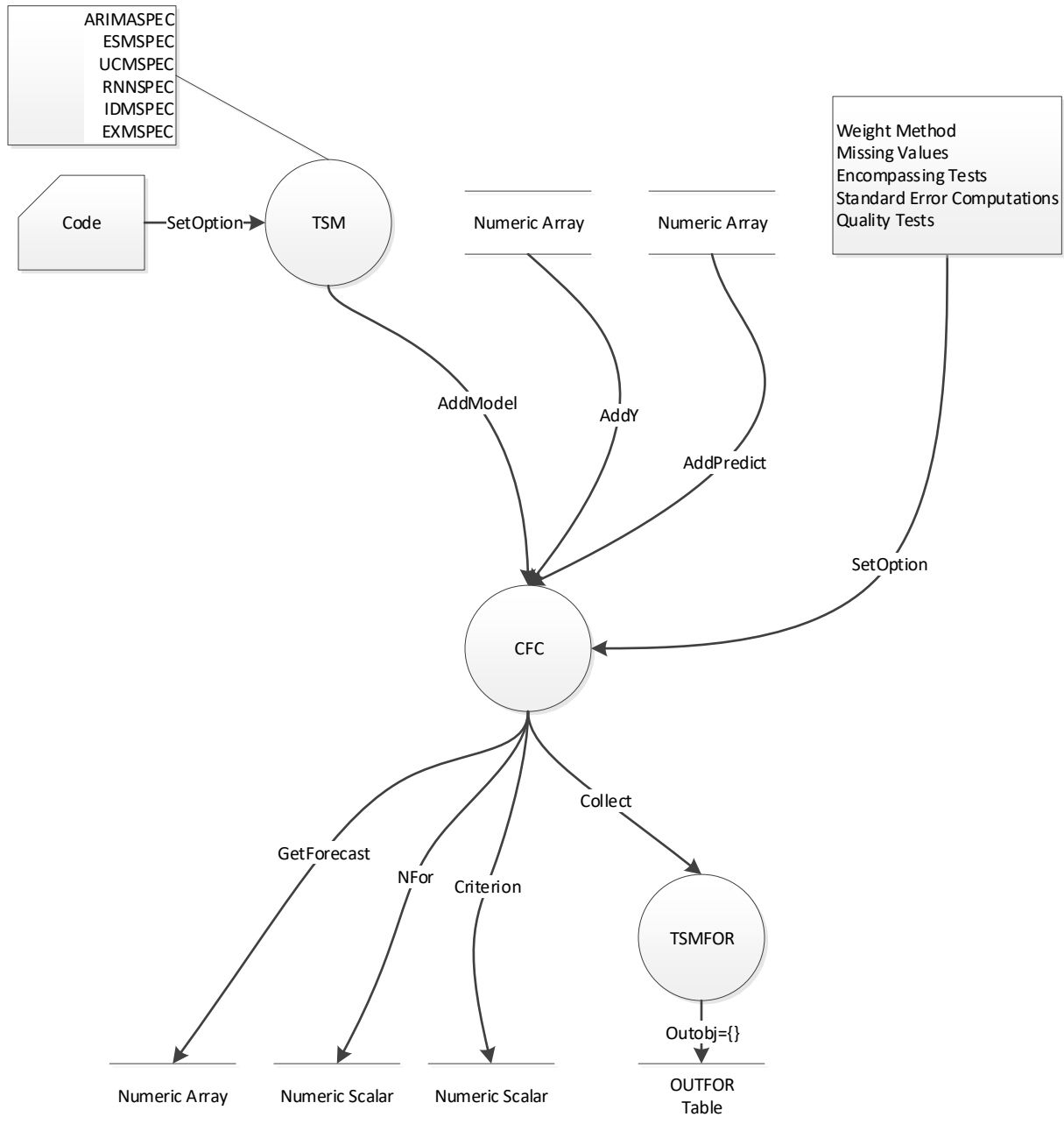
Method	Description
AddModel	Add a forecast series from an instance of the TSM object
AddPredict	Add a time series array as a forecast series
criterion	Return the final fit statistic over a specified forecast region
GetForecast	Get the forecast series
Initialize	Initialize the CFC object
nfor	Return the forecast series length
Run	Run the CFC object to generate a combined forecast
SetOption	Specify the named option for the CFC object
SetY	Specify the dependent time series array (<i>YSeries</i>) for the CFC object

The basic execution pattern for using a CFC object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new instance of the CFC object.
- 2 Initialize:** The [CFC.Initialize](#) method resets the object in preparation for a new run.
- 3 SetY:** The [CFC.SetY](#) method defines the dependent time series for the CFC object.
- 4 AddModel:** The [CFC.AddModel](#) method adds the forecast series that was produced by an instance of the TSM object. Each call adds one model forecast series to the combination. Repeat as needed for each model forecast series to be combined. Also, if you invoke this method prior to invoking the [CFC.SetY](#) method, then this method will use the dependent series of the TSM object as the dependent series of the CFC object.
- 5 AddPredict:** The [CFC.AddPredict](#) method adds a forecast series to the combination that is specified as a numeric array. Each call adds one forecast series to the combination. Repeat as needed for each forecast series to be combined.
- 6 SetOption:** The [CFC.SetOption](#) method specifies any options that affect the generation of the combined model forecast. Each call defines an option. Repeat as needed to specify all options that are required.
- 7 Run:** The [CFC.Run](#) method uses its currently configured dependent and forecast series data to generate a combined forecast.

Figure 32.2 illustrates the data flow through the CFC object and its relationship with other components of the TSM package.

Figure 32.2 CFC Object Data Flow



CFC Synopsis

DECLARE OBJECT *obj* (CFC) ;

Method syntax, in order of typical usage:

```
rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.AddModel (ModelObj < , ModelName >) ;
rc=obj.AddPredict (Predict, PredictSE < ,ModelName, Nparms > ) ;
rc=obj.SetOption ('Name', Value < , 'Name', Value,... >) ;
rc=obj.Run () ;
rc=obj.GetForecast (Which, Result) ;
nfor=obj.nfor () ;
criterion=obj.criterion (Region) ;
```

CFC Methods

CFC.AddModel Method

```
rc=obj.AddModel (ModelObj < , ModelName >) ;
```

Adds the forecast model series that were produced by an instance of the TSM object to the forecast combination process. This includes both the predicted and standard error series that were produced by the TSM object during the forecasting process. By default, the CFC object uses the name of the TSM object instance to identify the forecast component associated with its forecast series. However, you can optionally specify the forecast component via the *ModelName* argument.

Input Arguments

You must specify the following input argument:

ModelObj specifies the TSM object to use as a source of forecast model series.

You can also specify the following input argument:

ModelName is a character string that specifies the name of the forecast component that is used to identify the model forecast series in the combination. The default is the name of the TSM object instance that is used as the source of forecast model series.

CFC.AddPredict Method

```
rc=obj.AddPredict (Predict, PredictSE < , ModelName, Nparms > ) ;
```

Adds a forecast series to be included in the forecast combination for the CFC object. This includes both the *Predict* predicted series and the *PredictSE* standard errors series as numeric arrays. You can optionally specify the name of the forecast component in the combination via the *ModelName* argument. Each call to the AddPredict method adds the specified *Predict* and *PredictSE* series to the CFC object to be included in the combination process. The arrays that are used to specify the *Predict* and *PredictSE* series do not need to be unique, but the forecast component names that are specified in the *ModelName* argument must be unique if you specify them. You can call the AddPredict method repeatedly to specify multiple forecast series.

Input Arguments

You must specify the following input arguments:

- Predict* specifies a numeric array that contains the predicted series.
- PredictSE* specifies a numeric array that contains the standard errors of the predicted series.

You can also specify the following input arguments:

- ModelName* is a character string that specifies the name of the forecast component that is used to identify the model forecast series in the combination. The default is a unique name that is autogenerated by the CFC object.
- Nparms* takes a numeric value that specifies the number of parameters used by the model that generated the forecast series. Valid values are integers greater than or equal to 0. The default value is 0.

CFC.criterion Method

criterion=obj.criterion (Region) ;

Returns the fit statistic value over the specified forecast *Region* for the CFC object. The criterion is set via the 'CRITERION' argument in the CFC.SetOption method. A missing value indicates that the CFC object instance has not produced a successful forecast.

Input Arguments

You must specify the following input argument:

- Region* is a case-insensitive character string that specifies the forecast region over which the fit statistic is computed. You can specify one of the following values:
- | | |
|-------------|--|
| BACK | returns the fit statistic over the time region that is subsequent to the FIT region and that did not contribute any data to the process of estimating combination weights for the specified model forecast series. |
| FIT | returns the fit statistic over the time region that supplied observations for the process of estimating combination weights for the specified model forecast series. |

CFC.GetForecast Method

rc=obj.GetForecast (Which, Result) ;

Gets the specified forecast series (*Which*) from the CFC object and stores it in the specified numeric array (*Result*).

Input Arguments

You must specify the following input argument:

- Which* is a case-insensitive character string that specifies the type of forecast series to return. You can specify one of the following values:

ERROR	returns prediction errors.
LOWER	returns a lower confidence limit series.
STDERR	returns a prediction standard error series.
PREDICT	returns a prediction series.
UPPER	returns an upper confidence limit series.

Output Arguments

You must specify the following output argument:

Result specifies a numeric array to receive the forecast series.

CFC.Initialize Method

```
rc=obj.Initialize ();
```

Initializes a CFC object by discarding a previous forecast and resetting the object's configuration to the default values. This method must be called before any other attributes for the CFC object are specified.

Arguments

There are no arguments associated with this method.

CFC.nfor Method

```
nfor=obj.nfor ();
```

Returns the length (observation count) of the forecast series for the CFC object. A missing value indicates that the CFC object has not produced a successful forecast.

Arguments

There are no arguments associated with this method.

CFC.Run Method

```
rc=obj.Run ();
```

Runs the CFC object to perform the forecast combination process by using the dependent (*YSeries*) and candidate forecast series that have been specified for it. Upon successful completion, various results can be extracted from the CFC object.

Arguments

There are no arguments associated with this method.

CFC.SetOption Method

```
rc=obj.SetOption ('Name', Value <, 'Name', Value, ... >) ;
```

Specifies the named options for the CFC object. When you invoke this method, the previous forecast combination produced by the [CFC.Run](#) method is discarded. You must then rerun the [CFC.Run](#) method in order to produce a new forecast combination using the updated configuration options.

Input Arguments

You must specify at least one of the following 'Names' and its associated *Value*:

'BACK'	takes a nonnegative integer <i>Value</i> that specifies the back region for model performance. If 'BACK'= n and the number of observations is T , then the first $T - n$ observations are used to diagnose a series. The default value is 0.						
'CRITERION'	takes a string <i>Value</i> that specifies the fit statistic to be computed for the final forecast. The default value is RMSE. For a list of valid values, see the 'CRITERION' argument in the "TSM.SetOption Method" on page 2353.						
'ENCALPHA'	takes a numeric <i>Value</i> between 0 and 1, inclusive, that specifies the significance level of the encompassing test. The default value is 0.05.						
'ENCTEST'	takes a string <i>Value</i> that specifies the encompassing test type. The encompassing test attempts to eliminate from consideration any forecasts that fail to add significant information to the final forecast. For a detailed description of the encompassing test, see the chapter "Forecast Combination Computational Details" in <i>SAS Forecast Server Procedures: User's Guide</i> . You can specify the following <i>Values</i> : <table data-bbox="406 1081 1443 1585"> <tr> <td>HLN</td><td>uses the Harvey-Leybourne-Newbold (HLN) test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.</td></tr> <tr> <td>NONE</td><td>performs no encompassing tests.</td></tr> <tr> <td>OLS</td><td>uses an OLS-based regression test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.</td></tr> </table>	HLN	uses the Harvey-Leybourne-Newbold (HLN) test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.	NONE	performs no encompassing tests.	OLS	uses an OLS-based regression test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.
HLN	uses the Harvey-Leybourne-Newbold (HLN) test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.						
NONE	performs no encompassing tests.						
OLS	uses an OLS-based regression test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.						
	The default is NONE.						
'HMISSPCT'	takes a numeric <i>Value</i> between 0 and 100 that specifies a threshold for the percentage of missing forecast values in the combination horizon. This threshold is used to exclude a candidate forecast from consideration in the final combination. By default, no horizon missing percentage test is performed on candidate forecasts. The forecast horizon is the region of time in which multistep forecasts are generated.						
'LEAD'	takes a nonnegative integer <i>Value</i> that specifies the forecast lead. The default is the value that is specified in the PROC TSMODEL statement (LEAD= option). If the LEAD=						

option is not specified in the PROC TSMODEL statement, the default value of LEAD=0 is used.

'MISSMODE' takes a string *Value* that specifies a method for treating missing values in the forecast combination. In a particular time slice across the combination ensemble, one or more combination contributors can have a missing value. This setting determines the treatment of those contributors in the final combination for such time indices. You can specify the following *Values*:

MISSING generates a missing combined forecast at each time index with one or more missing contributors.

RESCALE rescales the combination weights for the nonmissing contributors at each time index to sum to 1. You cannot specify RESCALE when the value of the 'WEIGHT' argument is OLS or NRLS.

The default value depends on the combination weight method that is specified by the 'WEIGHT' argument. RESCALE is the default for simple average, user-specified weights, ranked user weights, ranked weights, and RMSE weights. MISSING is the default for all other methods.

'MISSPCT' takes a numeric *Value* between 0 and 100 that specifies a threshold for the percentage of missing forecast values in the combination estimation region. This threshold is used to exclude a candidate forecast from consideration in the final combination. By default, no missing percentage test is performed on candidate forecasts.

'RANKING' takes a string *Value* that specifies the fit statistic to be used for ranking forecast candidates. This option is used in conjunction with the 'ENCTEST' and 'WEIGHT' (when *Value* is set to RANKWGT) arguments. For a list of valid values, see the 'CRITERION' argument in the “[TSM.SetOption Method](#)” on page 2353. The default value is RMSE.

'SEMODE' takes a string *Value* that specifies the method for computing the prediction error variance series. This series is used to compute the prediction standard error, which in turn is used to compute confidence bands on the combined forecast. You can specify the following *Values*:

DIAG computes the prediction error variance by assuming that the forecast errors at time t are uncorrelated so that the simple diagonal form of Σ_t is used.

ESTCORR computes the prediction error variance by using estimates of $\rho_{i,j,t}$, the sample cross-correlation between $e_{i,t}$ and $e_{j,t}$ over the time span $t = 1, \dots, T$, where t denotes the last time index of the actual series y_t . This *Value* implies that the error series $e_{i,t}$ and $e_{j,t}$ are assumed to be jointly stationary.

The default is DIAG.

'USERWEIGHTS' takes a numeric array *Value* that specifies the combination weights to be used when the value of the 'WEIGHT' argument is RANKWGT or USERDEF. If the specified weights do not sum to 1, then the weights are automatically normalized to sum to 1 and a warning message is issued. For more information, see the 'WEIGHT' argument.

'WEIGHT' takes a string *Value* that specifies the method for determining the combination weights that are used in the weighted average of the candidate model forecasts. You can specify the following *Values*:

AICC	computes the combination weights on the basis of corrected AIC weights. By default, all AICC scored candidate forecasts are combined.
AVERAGE	computes the simple average of the forecasts that are selected for combination.
ERLS	computes the combination weights on the basis of a constrained least squares problem to minimize the ℓ_2 norm of the combined forecast residuals subject to the constraint that the weights sum to 1.
LAD	computes the weights on the basis of a least absolute deviations (LAD) measure of fit for the combined forecast. A linear program is formulated in which an objective function to be minimized is expressed in terms of the absolute values of a loss series subject to constraints that the weights sum to 1 and be nonnegative.
NERLS	computes the combination weights on the basis of a constrained least squares problem to minimize the ℓ_2 norm of the combined forecast residuals subject to the constraints that the weights sum to 1 and be nonnegative.
NRLS	is equivalent to NERLS, except that the resulting combination weights are not constrained to summing to 1.
OLS	computes the combination weights that result from the ordinary least squares problem to minimize the ℓ_2 norm of the combined forecast residuals.
RANKWGT	assigns weights by using the rank of the candidate forecasts at the time the combination is performed. You can optionally specify the weights in the 'USERWEIGHTS' argument, where the number of specified values must match the number of model forecasts that are specified in the current instance of the CFC object. The weights are assigned by ranking the candidate forecasts from best to worst on the basis of the fit statistic specified in the 'RANKING' argument. The best candidate uses the first weight, W1, and so on. If you do not specify the weights in the 'USERWEIGHTS' argument, then the weight of each candidate forecast is set to the inverse value of its rank. The set of weights that is used is normalized to account for candidate forecasts that are omitted from the final combination.
RMSEWGT	computes the combination weights on the basis of the RMSE statistic of fit for the forecast contributors. The weights are normalized to sum to 1.
USERDEF	assigns weights by using the list of user-specified values. You must specify the weights in the 'USERWEIGHTS' argument, where the number of specified values must match the number of model forecasts that are specified in the current instance of the CFC object. The weights correspond to the order of specification of the model forecasts. The set of weights that is used is normalized to account for candidates that are omitted from the final combination.

The default is AVERAGE.

CFC.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the dependent time series array (*YSeries*) for the CFC object.

Input Arguments

You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the CFC object.

ESMSPEC Object

The ESMSPEC object defines an exponential smoothing model specification for use with the TSM object. The basic execution pattern for defining an ESMSPEC follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new ESMSPEC object. By default, it is an ESMSPEC with METHOD=BEST.
- 2 Open:** The ESMSPEC.Open method initializes the default ESMSPEC object for a new configuration.
- 3 Configure:** The various ESMSPEC.Set methods configure the ESMSPEC object.
- 4 Close:** The ESMSPEC.Close method finalizes the ESMSPEC object.
- 5 Apply:** Add the ESMSPEC to a TSM object using the TSM.Initialize method.

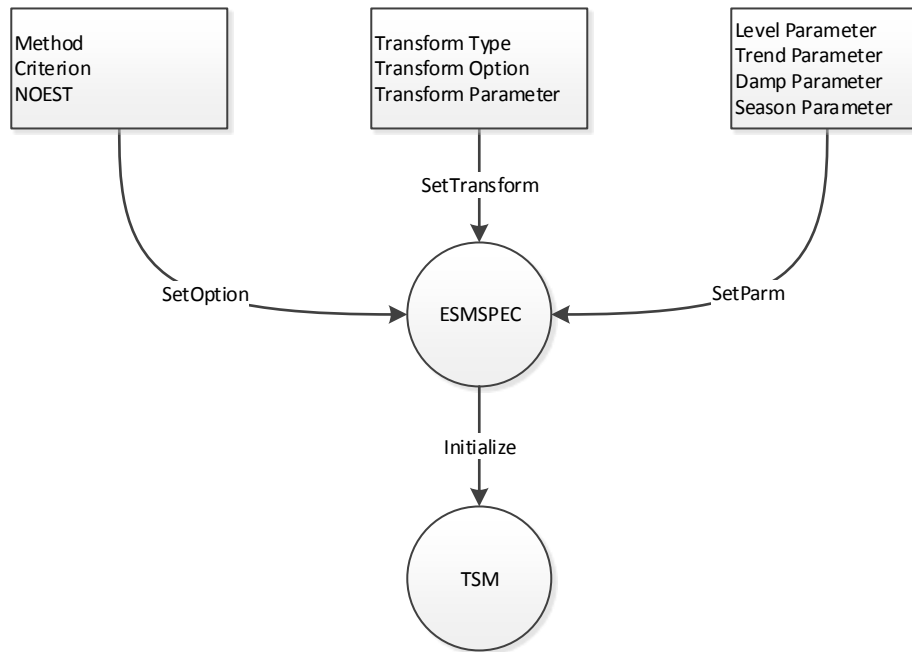
Table 32.6 summarizes the methods that are associated with the ESMSPEC object.

Table 32.6 Methods of the ESMSPEC Object

Method	Description
Close	Close ESM model specification
GetLabel	Retrieve label of specified ESM model
Open	Open ESM model specification
SetOption	Set option for ESM model
SetParm	Set parameters for ESM model
SetTransform	Set transform for ESM model

You can also store the XML representation of the ESMSPEC object in an output table. For more information, see the section “[TSM SPEC Object](#)” on page 2410.

Figure 32.3 illustrates the data flow through the ESMSPEC object.

Figure 32.3 ESMSPEC Object Data Flow

ESMSPEC Synopsis

DECLARE OBJECT *obj* (ESMSPEC) ;

Method syntax, in order of typical usage:

```

rc=obj.Open () ;
rc=obj.SetTransform ('Type' <,'Option',Parm>) ;
rc=obj.SetParm ('CompName' < , Parm, LRest, URest>) ;
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.GetLabel () ;
rc=obj.Close () ;
  
```

ESMSPEC Methods

ESMSPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the ESMSPEC object to prepare the ESM model to be used in a TSM object or to be imported to a TMSPEC object for printing or for storage in a model repository catalog.

Arguments

There are no arguments associated with this method.

ESMSPEC.GetLabel Method

```
rc=obj.GetLabel () ;
```

Retrieves the label of the specified ESM model and stores it in the return variable, *rc*.

Arguments

There are no arguments associated with this method.

ESMSPEC.Open Method

```
rc=obj.Open () ;
```

Opens the ESMSPEC object.

Arguments

There are no arguments associated with this method.

ESMSPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value <, 'Name', Value, ... >) ;
```

Specifies ESM options.

Input Arguments

You must specify at least one of the following *'Names'* and its associated *Value*:

'CRITERION' takes a string *Value* that specifies the model selection criterion (statistic of fit) to be used to select from several candidate models. For a list of valid values, see the **'CRITERION'** argument in the “[TSM.SetOption Method](#)” on page 2353. The default value is RMSE.

'METHOD' takes a string *Value* that specifies the ESM method. You can specify the following values:

ADDWINTERS requests the Winters additive method.

BEST uses seasonal properties of the data to choose between the BESTN and BESTS methods. If the data consist of a single seasonal cycle, the BESTN method is used. Otherwise, a seasonality test is performed on both the original data and the original data after one

nonseasonal difference is taken. If seasonality is present in either the original or differenced series, the series is considered to be seasonal. If the data are seasonal, the BESTS method is used. If neither seasonality test indicates seasonality, the BESTN method is used. The results of the seasonality test are summarized in [Table 32.7](#). The significance level for the seasonality test is 0.05. For more information about the seasonality test, see the SEASONTTEST function of the time series analysis (TSA) package.

BESTN	requests the best candidate nonseasonal smoothing model among the SIMPLE, LINEAR, and DAMPTREND methods.
BESTS	requests the best candidate seasonal smoothing model among the SEASONAL, ADDWINTERS, and WINTERS methods.
DAMPTREND	requests damped trend exponential smoothing.
DOUBLE	requests second order exponential smoothing.
LINEAR	requests linear (Holt) exponential smoothing.
MULTSEASONAL	requests multiplicative seasonal exponential smoothing.
SEASONAL	requests additive seasonal exponential smoothing.
SIMPLE	requests simple (single) exponential smoothing.
WINTERS	requests the Winters multiplicative method.

By default, METHOD=BEST.

'NOSTABLE'	takes a Boolean <i>Value</i> (0 or 1) that, when set to 1, requests that the smoothing model parameter estimates not be restricted to the additive invertible region of the parameter space. The default is 0 (parameter estimates are restricted).
'NOEST'	takes a Boolean <i>Value</i> that indicates whether the smoothing model parameters are fixed values. By default, the smoothing model parameters are optimized. 'NOEST' requires all of the exponential smoothing model parameters to be explicitly specified via calls to the ESMSPEC.SetParm method. This argument is ignored if any of the model parameters is not specified.

Table 32.7 Results of BEST Method of the ESMSPEC.SetOption Method

Number of Seasonal Cycles	Result of Seasonality Test	Method Used
< 2	Test is not performed	BESTN
≥ 2	Test indicates seasonality	BESTS
≥ 2	Test does not indicate seasonality	BESTN

ESMSPEC.SetParm Method

```
rc=obj.SetParm ('CompName' < , Parm, LRest, URest>);
```

Specifies parameter values and restrictions for the specified ESM component. Optional bounds, *LRest* and *URest*, can be specified to restrict the weight value for the specified ESM component during parameter optimization. Parameter values and restrictions must be in the range from -1 to 2 .

Input Arguments

You must specify the following input argument:

<i>'CompName'</i>	specifies a character variable that indicates the ESM component context. You can specify the following values:
DAMP	specifies that <i>Parm</i> is the initial value of a damping weight parameter.
LEVEL	specifies that <i>Parm</i> is the initial value of a level weight parameter.
SEASON	specifies that <i>Parm</i> is the initial value of a season weight parameter.
TREND	specifies that <i>Parm</i> is the initial value of a trend weight parameter.

You can also specify the following input arguments:

<i>Parm</i>	takes a numeric value that specifies the initial value of the smoothing component weight. The default value is a missing value, which causes the optimizer to use a grid search to find an appropriate initial value.
<i>LRest</i>	is a numeric variable that specifies a lower bound on the smoothing weight. The default value of the level, season, and trend weight parameters is 0.001 . The default value of the damping weight parameter is 0.8 .
<i>URest</i>	is a numeric variable that specifies an upper bound on the smoothing weight. The default value is 0.999 for all smoothing weights.

ESMSPEC.SetTransform Method

```
rc=obj.SetTransform ('Type' < , 'Option', Parm>);
```

Specifies the functional transform, *'Type'*, to be used by the ESM model. The optional arguments *Option* and *Parm* offer greater control over the transform.

Input Arguments

You must specify the following input argument:

<i>'Type'</i>	takes one of the following string values:
AUTO	automatically chooses between NONE and LOG based on model selection criteria.
BOXCOX	requests Box-Cox transformation. You can specify the control parameter for the transformation by using the optional <i>Parm</i> argument, whose value must be between -5 and 5 . The default value of <i>Parm</i> is 1 .

LOG	requests logarithmic transformation.
LOGIT LOGISTIC	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	requests square-root transformation.

The default is NONE.

You can also specify the following input arguments:

<i>'Option'</i>	takes one of the following string values that specifies prediction semantics for the inverse transform: <table><tr><td>MEAN</td><td>requests that the inverse transform produce mean forecasts.</td></tr><tr><td>MEDIAN</td><td>requests that the inverse transform produce median forecasts.</td></tr></table> The default is MEAN.	MEAN	requests that the inverse transform produce mean forecasts.	MEDIAN	requests that the inverse transform produce median forecasts.
MEAN	requests that the inverse transform produce mean forecasts.				
MEDIAN	requests that the inverse transform produce median forecasts.				
<i>Parm</i>	takes a numeric value between −5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transformations. The default is 1.				

ARIMASPEC Object

The ARIMASPEC object generates autoregressive integrated moving average (ARIMA) model specifications for use with a TSM object.

Table 32.8 summarizes the methods that are associated with the TSM.ARIMASPEC method.

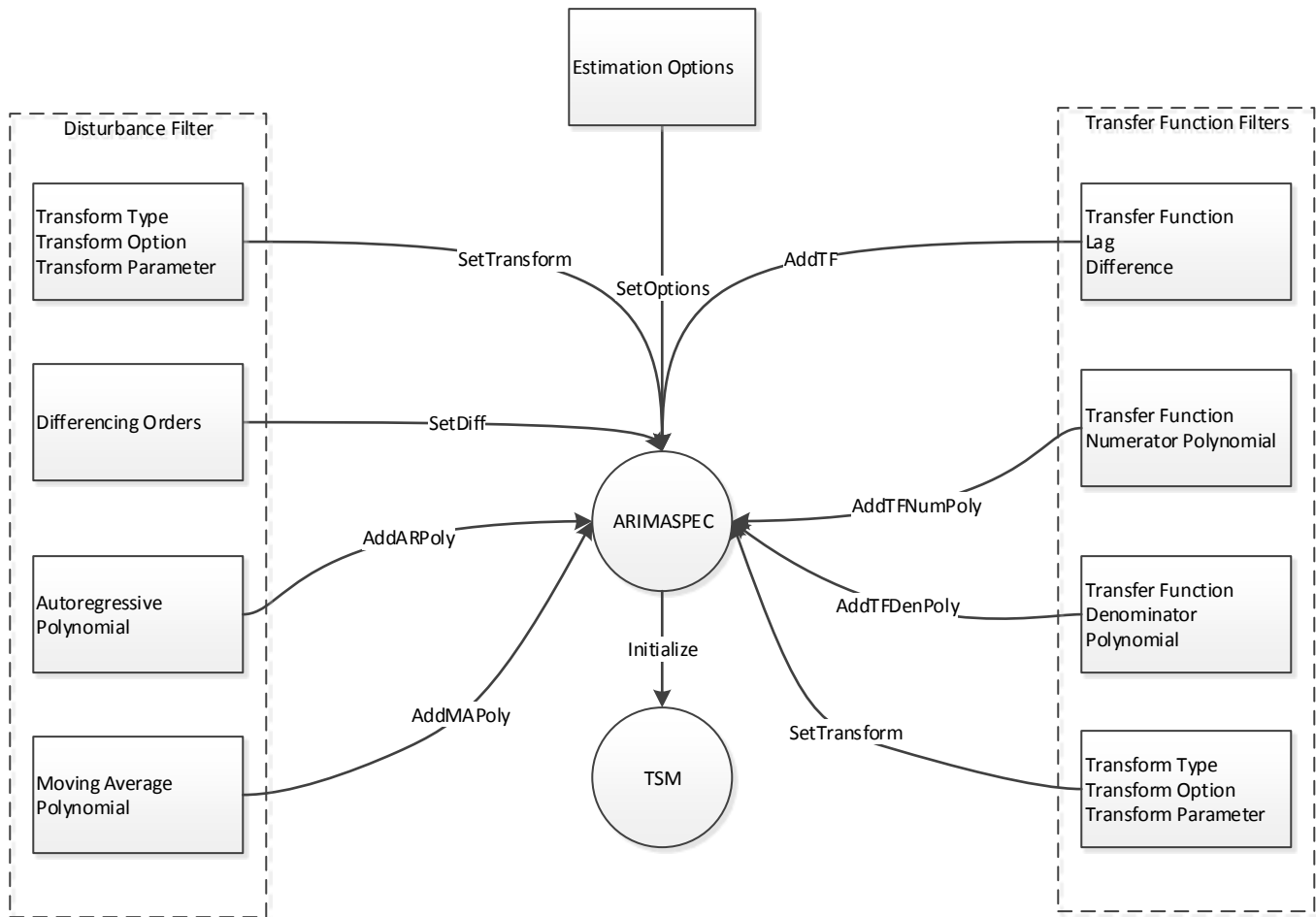
Table 32.8 Methods of the ARIMASPEC Object

Method	Description
AddARPoly	Add an autoregressive polynomial factor to the ARIMA model
AddMAPoly	Add a moving average polynomial factor to the ARIMA model
AddTF	Add a transfer function to the ARIMA model
AddTFDenPoly	Add a transfer function denominator factors to the ARIMA model
AddTFNumPoly	Add a transfer function numerator factors to the ARIMA model
Close	Close the ARIMA model specification
GetLabel	Retrieve the label of the specified ARIMA model
Open	Open the ARIMA model specification
SetDiff	Add differencing to the ARIMA model
SetOption	Specify the ARIMA option
SetTFTransform	Specify the transfer function transform
SetTransform	Specify transform

The basic execution pattern for using an ARIMASPEC object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new ARIMASPEC object. At creation, its default state is ARIMA(0,1,0) with intercept (random walk with drift).
- 2 Open:** The ARIMASPEC.Open method readies the default ARIMASPEC object for new configuration.
- 3 Configure:** The various ARIMASPEC.Set methods configure the ARIMASPEC object.
- 4 Close:** The ARIMASPEC.Close method finalizes the ARIMASPEC object.
- 5 Apply:** The TSM.Initialize method adds the ARIMASPEC to a TSM object.

You can also store the XML representation of the ARIMA model in a table. For more information, see the section “[TSMSPC Object](#)” on page 2410. [Figure 32.4](#) illustrates the data flow through the ARIMASPEC object.

Figure 32.4 ARIMASPEC Object Data Flow

You can include series transforms such as log or Box-Cox in the specification. Deferred seasonality wildcards for seasonal ARIMA polynomial factors and seasonal differencing lags are also supported. The following list summarizes where these features can be used. For more information, see the method descriptions.

- `ARIMASPEC.SetDiff` supports a seasonal wildcard lag in the `DiffArray`.
- `ARIMASPEC.AddARPoly` and `ARIMASPEC.AddMAPoly` support addition of seasonal ARIMA polynomial factors.
- `ARIMASPEC.AddTF` supports a seasonal wildcard lag in the `DiffArray`.
- `ARIMASPEC.AddTFNumPoly` and `ARIMASPEC.AddTFDenPoly` support addition of seasonal ARIMA polynomial factors for the transfer function numerator and denominator, respectively.
- `ARIMASPEC.SetTransform` applies transforms such as log, square root, logistic, or Box-Cox to the model.

When you specify ARIMA model coefficient values for any model components, you must specify them for all model components without regard to the value of the NOEST argument in the [ARIMASPEC.SetOption](#) method. Failing to specify a complete set of ARIMA model parameters results in an error when you call [ARIMASPEC.Close](#), and the accumulated ARIMA model specification is reset.

ARIMASPEC Synopsis

DECLARE OBJECT *obj* (**ARIMASPEC**) ;

Method syntax, in order of typical usage:

```
rc=obj.Open () ;
rc=obj.SetTransform ('Type' <,'Option', Parm>) ;
rc=obj.SetDiff (DiffArray <,NDiff>) ;
rc=obj.AddARPoly (OrderArray <,NOrder,Seasonal,CoeffArray>) ;
rc=obj.AddMAPoly (OrderArray <,NOrder,Seasonal,CoeffArray>) ;
rc=obj.AddTF (XName, < Delay,DifArray,NDiff,ConstCoeff>) ;
rc=obj.AddTFNumPoly (XName, NumArray <,NNum,Seasonal,CoeffArray>) ;
rc=obj.AddTFDenPoly (XName, DenArray <,NDen,Seasonal,CoeffArray>) ;
rc=obj.SetTFTransform (XName, 'Type', < Parm>) ;
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.GetLabel () ;
rc=obj.Close () ;
```

ARIMASPEC Methods

ARIMASPEC.AddARPoly Method

rc=obj.AddARPoly (OrderArray <, NOrder, Seasonal, CoeffArray>) ;

Adds an autoregressive (AR) polynomial factor to ARIMA model. Additional AR polynomial factors can be added to the ARIMA model with subsequent calls to this method.

Input Arguments

You must specify the following input argument:

OrderArray is a numeric array that specifies AR polynomial lags. Valid values are integers greater than or equal to 1.

You can also specify the following input arguments:

<i>NOrder</i>	takes a numeric variable that specifies the number of <i>OrderArray</i> values to use. If you do not specify <i>NOrder</i> , or if you specify no value or a missing value for it, it is set to the length of <i>OrderArray</i> —that is, all values in <i>OrderArray</i> are used.
<i>Seasonal</i>	takes a Boolean value (0 or 1) that, when set to 1, specifies that the AR polynomial is seasonal. By default, the AR polynomial is not seasonal and the lags of the AR polynomial are simple.

CoeffArray is a numeric array that specifies the initial values of AR polynomial coefficients. If you set the 'NOEST' option to 1 by using the [ARIMASPEC.SetOption](#) method, then these initial values are also the final values of the coefficients. If you specify this argument, then you must also complete all the following tasks:

- Ensure that the cardinality of *CoeffArray* is at least *NOrder*.
- If you call [ARIMASPEC.AddTF](#) method, then specify the *ConstCoeff* argument in that call.
- If you call any of the following methods, then specify the *CoeffArray* argument in that method call: [ARIMASPEC.AddMAPoly](#), [ARIMASPEC.AddTFDenPoly](#), or [ARIMASPEC.AddTFNumPoly](#).
- If you do not set 'NOINT' to 1 by using the [ARIMASPEC.SetOption](#) method, then specify 'MU' by using the [ARIMASPEC.SetOption](#) method.

ARIMASPEC.AddMAPoly Method

```
rc=obj.AddMAPoly (OrderArray < , NOrder, Seasonal, CoeffArray> ) ;
```

Adds a moving average (MA) polynomial factor to ARIMA model. More MA polynomial factors can be added to the ARIMA model by subsequent calls to the this method.

Input Arguments

You must specify the following input argument:

OrderArray is a numeric array that specifies MA polynomial lags. Valid values are integers greater than or equal to 1.

You can also specify the following input arguments:

NOrder takes a numeric variable that specifies the number of *OrderArray* values to use. If you do not specify *NOrder*, or if you specify no value or a missing value for it, it is set to the length of *OrderArray*—that is, all values in *OrderArray* are used.

Seasonal takes a Boolean value (0 or 1) that, when set to 1, specifies that the MA polynomial is seasonal. By default, the MA polynomial is not seasonal and the lags of the MA polynomial are simple.

CoeffArray is a numeric array that specifies the initial values of MA polynomial coefficients. If you set the 'NOEST' option to 1 by using the [ARIMASPEC.SetOption](#) method, then these initial values are also the final values of the coefficients. If you specify this argument, then you must also complete all the following tasks:

- Ensure that the cardinality of *CoeffArray* is at least *NOrder*.
- If you call [ARIMASPEC.AddTF](#) method, then specify the *ConstCoeff* argument in that call.
- If you call any of the following methods, then specify the *CoeffArray* argument in that method call: [ARIMASPEC.AddARPoly](#), [ARIMASPEC.AddTFDenPoly](#), or [ARIMASPEC.AddTFNumPoly](#).
- If you do not set 'NOINT' to 1 by using the [ARIMASPEC.SetOption](#) method, then specify 'MU' by using the [ARIMASPEC.SetOption](#) method.

ARIMASPEC.AddTF Method

```
rc=obj.AddTF (XName < , Delay, DifArray, NDiff, ConstCoeff>) ;
```

Adds a transfer function to the ARIMA model for the specified *XName* variable. This method adds the variable as a simple scale effect subject to any specified lag and differencing that might be applied.

Input Arguments

You must specify the following input argument:

XName is a character string that specifies the name of the X variable.

You can also specify the following input arguments:

Delay takes a numeric value that specifies the simple delay for the predictor. The default value is 0.

DifArray is a numeric array that specifies differencing orders for the X variable. Valid values are integers greater than or equal to 1. If you do not specify *DifArray*, or if you specify no value for it, then no differencing is applied. Values in *DifArray* are interpreted as follows:

- Negative values are not allowed and result in an error condition.
- Nonnegative values represent differencing orders.
- An .S missing value is interpreted to include seasonal difference order.
- Any other missing value is ignored.

NDiff takes a numeric value that specifies the number of *DifArray* values to use. If you do not specify *NDiff*, or if you specify no value or a missing value for it, then all the values in *DifArray* are used.

ConstCoeff takes a numeric value that specifies the initial value of the scale parameter—that is, the zero-degree coefficient of the numerator. If you set the 'NOEST' option to 1 by using the [ARIMASPEC.SetOption](#) method, then this initial value is also the final value of the coefficient. If you specify this argument, then you must also complete the following tasks:

- If you call any of the following methods, then you must also specify the *CoeffArray* argument in that method call: [ARIMASPEC.AddARPoly](#), [ARIMASPEC.AddMAPoly](#), [ARIMASPEC.AddTFDenPoly](#), or [ARIMASPEC.AddTFNumPoly](#).
- If you do not set 'NOINT' to 1 by using the [ARIMASPEC.SetOption](#) method, then specify 'MU' by using the [ARIMASPEC.SetOption](#) method.

ARIMASPEC.AddTFDenPoly Method

```
rc=obj.AddTFDenPoly (XName, DenArray < , NDen, Seasonal, CoeffArray>) ;
```

Adds a transfer function denominator polynomial factor for the specified *XName* variable. More polynomials can be added by subsequent calls to the [ARIMASPEC.AddTFNumPoly](#) method.

Input Arguments

You must specify the following input argument:

<i>XName</i>	is a character string that specifies the name of the X variable.
<i>DenArray</i>	is a numeric array that specifies denominator polynomial lags for the X variable. Valid values are integers greater than 0.

You can also specify the following input arguments:

<i>NDen</i>	takes a numeric value that specifies the number of <i>DenArray</i> values to use. Valid values are integers greater than 0. If you do not specify <i>NDen</i> , or if you specify no value or a missing value for it, it is set to the length of <i>DenArray</i> —that is, all values in <i>DenArray</i> are used.
<i>Seasonal</i>	takes a Boolean value (0 or 1) that, when set to 1, specifies that the denominator polynomial is seasonal. By default, the denominator polynomial is not seasonal and the lags of the denominator polynomial are simple.
<i>CoeffArray</i>	is a numeric array that specifies the initial values of denominator polynomial coefficients. If you set the 'NOEST' option to 1 by using the ARIMASPEC.SetOption method, then these initial values are also the final values of the coefficients. If you specify this argument, then you must also complete all the following tasks: • Ensure that the cardinality of <i>CoeffArray</i> is at least <i>NDen</i>. • Specify the <i>ConstCoeff</i> argument in the ARIMASPEC.AddTF method. • If you call any of the following methods, then specify the <i>CoeffArray</i> argument in that method call: ARIMASPEC.AddARPoly, ARIMASPEC.AddMAPoly, or ARIMASPEC.AddTFNumPoly. • If you do not set 'NOINT' to 1 by using the ARIMASPEC.SetOption method, then specify 'MU' by using the ARIMASPEC.SetOption method.

ARIMASPEC.AddTFNumPoly Method

```
rc=obj.AddTFNumPoly (XName, NumArray < , NNum, Seasonal, CoeffArray> ) ;
```

Adds a transfer function numerator polynomial factor for the specified *XName* variable. More polynomials can be added by subsequent calls to the [ARIMASPEC.AddTFNumPoly](#) method.

Input Arguments

You must specify the following input argument:

<i>XName</i>	is a character string that specifies the name of the X variable.
<i>NumArray</i>	is a numeric array that specifies numerator polynomial lags for the X variable. Valid values are integers greater than 0.

You can also specify the following input arguments:

<i>NNum</i>	takes a numeric value that specifies the number of <i>NumArray</i> values to use. If you do not specify <i>NNum</i> , or if you specify no value or a missing value for it, it is set to the length of <i>NumArray</i> —that is, all values in <i>NumArray</i> are used.
<i>Seasonal</i>	takes a Boolean value (0 or 1) that, when set to 1, specifies that the numerator polynomial is seasonal. By default, the numerator polynomial is not seasonal and the lags of the numerator polynomial are simple.

CoeffArray is a numeric array that specifies the initial values of numerator polynomial coefficients. If you set the 'NOEST' option to 1 by using the [ARIMASPEC.SetOption](#) method, then these initial values are also the final values of the coefficients. If you specify this argument, then you must also complete all the following tasks:

- Ensure that the cardinality of *CoeffArray* is at least *NNum*.
- Specify the *ConstCoeff* argument in the [ARIMASPEC.AddTF](#) method.
- If you call any of the following methods, then specify the *CoeffArray* argument in that method call: [ARIMASPEC.AddARPoly](#), [ARIMASPEC.AddMAPoly](#), or [ARIMASPEC.AddTFDenPoly](#).
- If you do not set 'NOINT' to 1 by using the [ARIMASPEC.SetOption](#) method, then specify 'MU' by using the [ARIMASPEC.SetOption](#) method.

ARIMASPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the ARIMASPEC object to prepare the ARIMA model to be used in a TSM object or to be imported to a TSMSPEC object for printing or for storage in a model repository.

Arguments

There are no arguments associated with this method.

ARIMASPEC.GetLabel Method

```
rc=obj.GetLabel () ;
```

Retrieves the label of the specified ARIMA model and stores it in the return variable, *rc*.

Arguments

There are no arguments associated with this method.

ARIMASPEC.Open Method

```
rc=obj.Open () ;
```

Initializes an empty ARIMASPEC object for configuration.

Arguments

There are no arguments associated with this method.

ARIMASPEC.SetDiff Method

```
rc=obj.SetDiff (DiffArray < , NDiff>) ;
```

Adds differencing to the ARIMA model.

Input Arguments

You must specify the following input argument:

DiffArray is a numeric array that specifies differencing orders, where each order must be an integer greater than or equal to 1. Values in *DiffArray* are interpreted as follows:

- Negative values are not allowed and result in an error condition.
- Nonnegative integer values represent differencing orders.
- An .S missing value is interpreted to include seasonal difference order.
- Any other missing value is ignored.

You can also specify the following input argument:

NDiff takes a numeric value that specifies the number of *DiffArray* values to use. The default is to use all elements of *DiffArray*.

ARIMASPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value <, 'Name', Value, ... >) ;
```

Specifies ARIMA model options. Options are ('Name',Value) pairs where 'Name' is a case-insensitive character string and Value depends on the 'Name'.

Input Arguments

You must specify at least one of the following 'Names' and its associated Value:

'CONVERGE' takes a numeric Value between 0 and 1, exclusive, that specifies the convergence criterion. Convergence is assumed when the largest change in the estimate for any parameter is less than the specified Value. If the absolute value of the parameter estimate is greater than 0.01, the relative change is used; otherwise, the absolute change in the estimate is used. The default is 0.001.

'DELTA' takes a numeric Value between 0 and 1, exclusive, that specifies the perturbation value for computing numerical derivatives. The default is 0.001.

'MAXITER' takes a positive integer Value that specifies the maximum number of iterations allowed. The default is 50.

'METHOD' takes a string Value that specifies the estimation method to use. You can specify one of the following Values:

CLS specifies the conditional least squares method.

ML specifies the maximum likelihood method.

ULS specifies the unconditional least squares method.

The default is CLS.

'MU' takes a numeric Value that specifies a constant term for the ARIMA model. The default is 0.

'NOEST'	takes a Boolean <i>Value</i> (0 or 1) that, when set to 1, specifies that no estimation is performed. The default is 0 (estimation is performed).
'NOINT'	takes a Boolean <i>Value</i> (0 or 1) that, when set to 1, specifies that the model should not have an intercept. This suppresses the fitting of a constant (or intercept) parameter in the model, and the 'MU' <i>Value</i> that you specify is ignored. The default is 0 (model has an intercept).
'NOSTABLE'	takes a Boolean <i>Value</i> (0 or 1) that, when set to 1, requests that the autoregressive and moving average parameter estimates for the noise part of the model not be restricted to the stationary and invertible regions, respectively. The default is 0 (parameter estimates for the noise part of the model are restricted).
'SINGULAR'	takes a numeric <i>Value</i> between 0 and 1, exclusive, that specifies the criterion for checking singularity. If a pivot of a sweep operation is less than <i>Value</i> , the matrix is deemed singular. Sweep operations are performed on the Jacobian matrix during final estimation and on the covariance matrix when preliminary estimates are obtained. The default is 1E-7.

ARIMASPEC.SetTFTTransform Method

rc=obj.SetTFTTransform (XName, 'Type', < Parm >) ;

Specifies a functional transform for specified *XName* variable.

Input Arguments

You must specify the following input arguments:

<i>XName</i>	is a character string that specifies the name of the X variable.
<i>'Type'</i>	takes a string value that specifies the functional transform to use. You can specify the following values:
BOXCOX	requests Box-Cox transformation. You can specify the control parameter for the transformation by using the optional <i>Parm</i> argument, whose value must be between -5 and 5. The default value of <i>Parm</i> is 1.
LOG	requests logarithmic transformation.
LOGIT LOGISTIC	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	specifies square-root transformation.

The default is NONE.

You can also specify the following input argument:

<i>Parm</i>	takes a numeric value between -5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transformations. The default is 1.
-------------	--

ARIMASPEC.SetTransform Method

```
rc=obj.SetTransform ('Type' < ','Option',Parm>);
```

Specifies the functional transform '*Type*' to be used by the ARIMA model. Optional arguments *Option* and *Parm* offer greater control over the transform.

Input Arguments

You must specify the following input argument:

'Type'	takes a string value that specifies the functional transform to use. You can specify the following values:
BOXCOX	requests Box-Cox transformation. You can specify the control parameter for the transformation by using the optional <i>Parm</i> argument, whose value must be between -5 and 5 . The default value of <i>Parm</i> is 1 .
LOG	requests logarithmic transformation.
LOGIT LOGISTIC	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	specifies square-root transformation.

The default is NONE.

You can also specify the following input arguments:

'Option'	takes a string value that specifies prediction semantics for the inverse transform. You can specify the following values:
MEAN	requests that the inverse transform produce mean forecasts.
MEDIAN	requests that the inverse transform produce median forecasts.
	The default is MEAN.
Parm	takes a numeric value between -5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transformations. The default is 1 .

EXMSPEC Object

The EXMSPEC object generates an external model (EXM) specification for use with the TSM object, or with the automatic time series modeling (ATSM) package for automatic forecasting services.

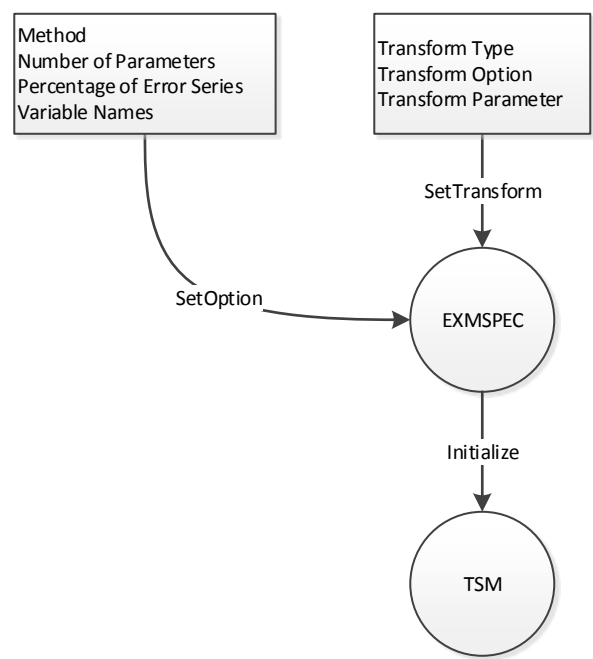
Table 32.9 summarizes the methods that are associated with the EXMSPEC object.

Table 32.9 Methods of the EXMSPEC Object

Method	Description
Close	Close EXM model specification
GetLabel	Retrieve label of specified EXM model
Open	Open EXM model specification
SetOption	Set options for EXM model specification
SetTransform	Set transform for EXM model specification

Figure 32.5 illustrates the data flow through the EXMSPEC object.

Figure 32.5 EXMSPEC Object Data Flow



The basic execution pattern for using an EXMSPEC object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new EXMSPEC object.
- 2 Open:** The EXMSPEC.Open method readies the default EXMSPEC object for new configuration.
- 3 Configure:** The various EXMSPEC.Set methods configure the EXMSPEC object.
- 4 Close:** The EXMSPEC.Close method finalizes the EXMSPEC object.
- 5 Apply:** The EXMSPEC object is applied to a TSM object using the TSM.Initialize method or is used with the ATSM package. For more information about using the EXMSPEC object, see the section “[TSM Object](#)” on page 2346 or Chapter 2, “Automatic Time Series Modeling Package” (*SAS Visual Forecasting: Time Series Packages*).

NOTE:

1. You can also store the XML representation of the EXMSPEC object in an output table. For more information, see the section “[TSM Object](#)” on page 2410.
2. To run the EXMSPEC object in a TSM object, time series variables for the forecast series roles must be added to the TSM object via the TSM.AddExternal method. For more information, see the section “[TSM.AddExternal Method](#)” on page 2348.
3. To run the EXMSPEC object in an ATSM:FORENG object, time series variables for the forecast series roles must be added to the ATSM:TSDf object that is used to specify the time series data for the ATSM:FORENG object. When the EXMSPEC object is used with the ATSM package, more sophisticated persistence mechanisms are available. For more information, see Chapter 2, “Automatic Time Series Modeling Package” (*SAS Visual Forecasting: Time Series Packages*).

EXMSPEC Synopsis

DECLARE OBJECT *obj* (EXMSPEC) ;

Method syntax, in order of typical usage:

```
rc=obj.Open () ;
rc=obj.SetTransform ('Type' < ,Option,Parm>) ;
rc=obj.SetOption ('Name', Value < ,'Name',Value,... >) ;
rc=obj.GetLabel () ;
rc=obj.Close () ;
```

EXMSPEC Methods

EXMSPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the EXM model in the EXMSPEC object. This prepares the EXM model for use in a TSM object or to be imported to a TSMSPEC object for printing or storage in a model repository catalog.

Arguments

There are no arguments associated with this method.

EXMSPEC.GetLabel Method

```
rc=obj.GetLabel () ;
```

Retrieves the label of the specified EXM model and stores it in the return variable, *rc*.

Arguments

There are no arguments associated with this method.

EXMSPEC.Open Method

```
rc=obj.Open () ;
```

Initializes an EXMSPEC object for configuration.

Arguments

There are no arguments associated with this method.

EXMSPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
```

Specifies the options for the EXMSPEC object.

Input Arguments

You must specify at least one of the following '*Names*' and its associated *Value*:

'ERROR'	takes a string <i>Value</i> that specifies the variable name for ERROR series.
'LOWER'	takes a string <i>Value</i> that specifies the variable name for LOWER series.
'METHOD'	takes a string <i>Value</i> that specifies the method to approximate prediction STDERR series. You can specify one of the following values:
ACF	Autocorrelation is used.
ERRORACF	Prediction error autocorrelation is used.
NONE	No prediction error autocorrelation is used.
PERFECT	Perfect autocorrelation is assumed.

	WN	Prediction error autocorrelation is white noise.
		The default is PERFECT.
'NLAGPCT'		takes a numeric <i>Value</i> between 0 and 100 that specifies the percentage of error series count for ACF computations. The default is 25 (25%).
'NPARMS'		takes a nonnegative integer <i>Value</i> that specifies the number of parameters for the external forecast. The default is 0.
'PREDICT'		takes a string <i>Value</i> that specifies the variable name for PREDICT series.
'SIGMA'		takes a numeric <i>Value</i> that specifies the prediction standard error for the external model. If left unspecified, then the prediction mean square error is computed from the prediction errors by using the 'NPARMS' argument. The default is missing value.
'STDERR'		takes a string <i>Value</i> that specifies the variable name for STDERR series.
'UPPER'		takes a string <i>Value</i> that specifies the variable name for UPPER series.

EXMSPEC.SetTransform Method

rc=obj.SetTransform ('Type' < ,Option,Parm>) ;

Specifies the functional transform to be used by the EXM model.

Input Arguments

You must specify the following input argument:

'Type'	takes one of the following string values:
BOXCOX	requests Box-Cox transformation. You can specify the control parameter for the transformation by using the optional <i>Parm</i> argument, whose value must be between −5 and 5. The default value of <i>Parm</i> is 1.
LOG	requests logarithmic transformation.
LOGIT LOGISTIC	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	requests square-root transformation.

The default is NONE.

You can also specify the following input arguments:

'Option'	takes one of the following string values that specifies prediction semantics for the inverse transform:
MEAN	requests that the inverse transform produce mean forecasts.
MEDIAN	requests that the inverse transform produce median forecasts.
	The default is MEAN.
<i>Parm</i>	takes a numeric value between −5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transformations. The default is 1.

IDMSPEC Object

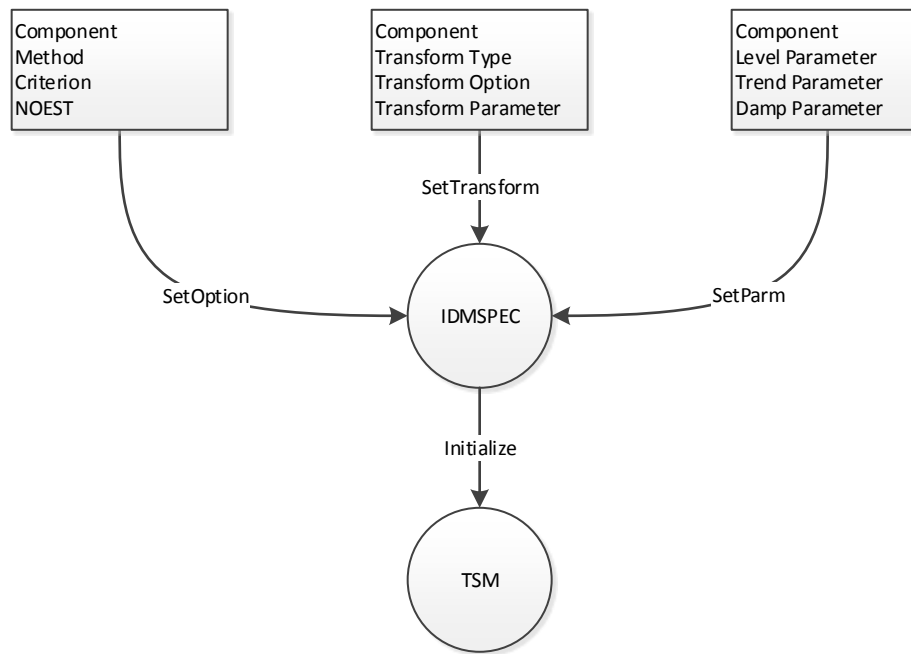
The IDMSPEC object generates intermittent demand models for use with the TSM object. You open the IDMSPEC object to begin defining a new IDM model. You call the IDMSPEC methods to configure the object with the desired settings to define the model characteristics of interest, and then you close the IDMSPEC object to ready it for use in a TSM object. For more information, see the section “[TSM.Initialize Method](#)” on page 2352. You can also store the XML representation of the IDM in an output table. For more information, see the section “[TSMSPEC Object](#)” on page 2410.

[Table 32.10](#) the methods that are associated with the IDMSPEC object.

Table 32.10 Methods of the IDMSPEC Object

Method	Description
Close	Close IDM model specification
GetLabel	Retrieve label of specified IDM model
Open	Open IDM model specification
SetMethod	Set component smoothing method
SetOption	Set options for IDM model specification
SetParm	Set component parameters
SetTransform	Set component transform

[Figure 32.6](#) illustrates the data flow through the IDMSPEC object.

Figure 32.6 IDMSPEC Object Data Flow

IDMSPEC Synopsis

DECLARE OBJECT *obj* (IDMSPEC) ;

Method syntax, in order of typical usage:

```

rc=obj.Open () ;
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.SetTransform ('IDMComp', 'Type' <,'Option',Parm>) ;
rc=obj.SetMethod ('IDMComp', 'Method') ;
rc=obj.SetParm ('IDMComp', 'ESMComp' < , Parm, Noest, LRest, URest>) ;
rc=obj.GetLabel () ;
rc=obj.Close () ;
  
```

IDMSPEC Methods

IDMSPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the IDM model in the IDMSPEC object and prepares the IDM model for use in a TSM object or to be imported to a TSMSPEC object for printing or storage in a model repository catalog.

Arguments

There are no arguments associated with this method.

IDMSPEC.GetLabel Method

```
rc=obj.GetLabel () ;
```

Retrieves the label of the specified IDM model and stores it in the return variable, *rc*.

Arguments

There are no arguments associated with this method.

IDMSPEC.Open Method

```
rc=obj.Open () ;
```

Opens the IDMSPEC object for configuration and initializes it with default options. The default options specify that the best IDM model from among the average demand and Croston's model be selected based on the RMSE criterion (see the value BEST for the '**MODEL**' argument in the SetOption method). The smoothing model for the individual components of both models is selected from among the best candidate nonseasonal smoothing method (see value BESTN for the '*Method*' argument in the SetMethod method). In addition, by default, all component transformations are automatically chosen between NONE and LOG based on the model selection criterion (see the value AUTO for the '*Type*' argument in the SetTransform method).

Arguments

There are no arguments associated with this method.

IDMSPEC.SetMethod Method

```
rc=obj.SetMethod ('IDMComp', 'Method') ;
```

Sets a smoothing method for IDM model component.

Input Arguments

You must specify the following input arguments:

'IDMComp' is a character string that specifies the IDM component model. You can specify one of the following values:

AVERAGE specifies the average demand model.

INTERVAL specifies demand interval model.

SIZE specifies demand size model.

'Method' takes a string value that specifies the name of the smoothing method. You can specify one of the following values:

BESTN requests the best candidate nonseasonal smoothing model among the SIMPLE, LINEAR, or DAMPTREND methods.

DAMPTREND requests damped trend exponential smoothing.

LINEAR requests linear (Holt) exponential smoothing.

NONE disables the use of a smoothing model for the specified component.

SIMPLE requests simple (single) exponential smoothing.

The default is BESTN.

IDMSPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... > ) ;
```

Specifies the options for the IDMSPEC object.

Input Arguments

You must specify at least one of the following *'Names'* and its associated *Value*:

'BASE' takes a numeric *Value* that specifies the base demand level. If you specify a missing value, the dependent series is analyzed to determine whether the base demand level can be set to one of the following values:

- 0
- the minimum value of the dependent series
- the maximum value of the dependent series
- the median value of the dependent series

First, the analysis determines whether the series is nearly constant by applying a threshold to the maximum (L_∞) norm of the nonmissing observations of the series. For nearly constant series, the base demand level is set to 0. Otherwise, analysis proceeds to compute the minimum, maximum, and median values of the series and assigns a score to each of those values. The score of a value in this context is the number of observations that are sufficiently close to that value. If the number of missing values in the dependent series is higher than the scores for the minimum, maximum, and median, then the base demand level is set to 0. If scores for the minimum, maximum, and median are identical, then the base demand level is set to 0. Otherwise, the base demand level is set to the value that has the highest score among the minimum, maximum, and median.

'CRITERION' takes a string *Value* that specifies the model selection criterion (statistic of fit) to be used to select from several candidate models. For a list of valid values, see the 'CRITERION' argument in the “[TSM.SetOption Method](#)” on page 2353. The default value is RMSE.

'MODEL' is a character string that specifies a mnemonic for the IDM model to use. You can specify the following *Values*:

- AVERAGE** uses the single smoothing model to fit the average demand component.
- BEST** requests the best IDM model from among the AVERAGE and CROSTON models.
- CROSTON** uses the two smoothing models to fit the demand interval component and the demand size component.

The default is BEST.

IDMSPEC.SetParm Method

```
rc=obj.SetParm ('IDMComp', 'ESMComp' < , Parm, Noest, LRest, URest>);
```

Sets parameter value and restrictions for the ESM component in the IDM component model. The ESM method-specific bounds are employed to limit or filter the smoothing method weight values.

Input Arguments

You must specify the following input arguments:

'IDMComp' is a character string that specifies the IDM component model. You can specify one of the following values:

- AVERAGE** specifies the average demand model.
- INTERVAL** specifies demand interval model.
- SIZE** specifies demand size model.

'ESMComp' is a character string that specifies the ESM component of the IDM component. You can specify one of the following values:

- DAMP** specifies that *Parm* is the initial value of a damping weight parameter.
- LEVEL** specifies that *Parm* is the initial value of a level weight parameter.
- TREND** specifies that *Parm* is the initial value of a trend weight parameter.

You can also specify the following input arguments:

- Parm* takes a numeric value that specifies the smoothing component weight.
- Noest* takes a Boolean value (0 or 1) that, when set to 1, specifies that the *Parm* argument is fixed. The default value is 0 (initial).
- LRest* takes a numeric value between –1 and 2 that specifies a lower bound restriction on the ESM weight. The default value of the level and trend weight parameters is 0.001. The default value of the damping weight parameter is 0.8.
- URest* takes a numeric value between –1 and 2 that specifies an upper bound restriction on the ESM weight. The default value is 0.999 for all smoothing weights.

IDMSPEC.SetTransform Method

```
rc=obj.SetTransform ('IDMComp', 'Type' <','Option',Parm>) ;
```

Sets the functional transform to be used by the IDM model component.

Input Arguments

You must specify the following input arguments:

'IDMComp' is a character string that specifies the IDM component model. You can specify one of the following values:

AVERAGE specifies the average demand model.

INTERVAL specifies demand interval model.

SIZE specifies demand size model.

'Type' takes a string value that specifies the transform to use. You can specify one of the following values:

AUTO automatically chooses between NONE and LOG based on model selection criteria.

BOXCOX requests Box-Cox transformation. You can specify the control parameter for the transformation by using the optional *Parm* argument, whose value must be between –5 and 5. The default value of *Parm* is 1.

LOG requests logarithmic transformation.

LOGIT | LOGISTIC requests logistic transformation.

NONE does not apply a transformation.

SQRT requests square-root transformation.

The default is AUTO.

You can also specify the following input arguments:

'Option' takes a string value that specifies prediction semantics for the inverse transform. You can specify the following values:

MEAN requests that the inverse transform produce mean forecasts.

MEDIAN requests that the inverse transform produce median forecasts.

The default is MEAN.

Parm takes a numeric value between –5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transformations. The default is 1.

RNNSPEC Object

The RNNSPEC object generates time series recurrent neural network (RNN) model specifications for use with a TSM object.

Table 32.11 summarizes the methods that are associated with the TSM.RNNSPEC method.

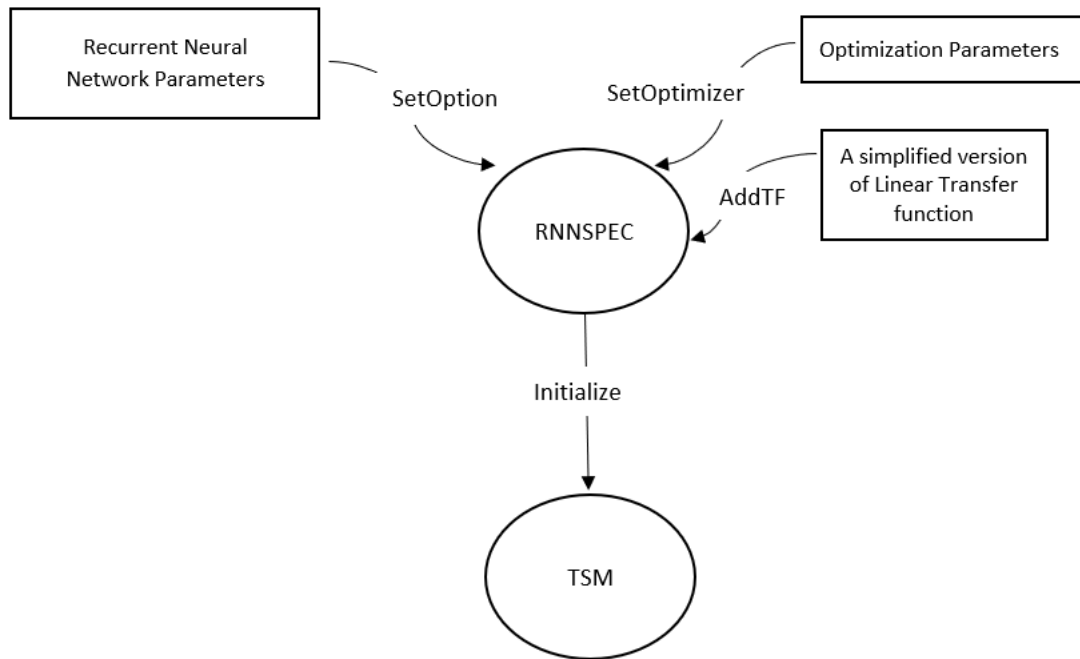
Table 32.11 Methods of the RNNSPEC Object

Method	Description
AddTF	Add a transfer function to the RNN model
Close	Close the RNN model specification
GetLabel	Retrieve the label of the specified RNN model
Open	Open the RNN model specification
SetOptimizer	Specify optimizer options
SetOption	Specify the RNN model options

The basic execution pattern for using an RNNSPEC object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new RNNSPEC object.
- 2 Open:** The RNNSPEC.Open method prepares the default RNNSPEC object for a new configuration.
- 3 Configure:** The various RNNSPEC.Set methods configure the RNNSPEC object.
- 4 Close:** The RNNSPEC.Close method finalizes the RNNSPEC object.
- 5 Apply:** The TSM.Initialize method adds the RNNSPEC to a TSM object.

You can also store the XML representation of the RNN model in an output table. For more information, see the section “TSMSPEC Object” on page 2410. Figure 32.7 illustrates the data flow through the RNNSPEC object.

Figure 32.7 RNNSPEC Object Data Flow

RNNSPEC Synopsis

DECLARE OBJECT *obj* (RNNSPEC) ;

Method syntax, in order of typical usage:

```

rc=obj.Open () ;
rc=obj.AddTF (XName) ;
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.SetOptimizer ('Name', Value <,'Name',Value,... >) ;
rc=obj.GetLabel () ;
rc=obj.Close () ;
  
```

RNNSPEC Methods

RNNSPEC.AddTF Method

```
rc = obj.AddTF (XName) ;
```

This option adds a transfer function to the RNN model for the specified *XName* variable. Unlike the ARIMASPEC object, all other parameters except the variable name in AddTF are disabled. The RNNSPEC AddTF method provides a basic transfer function, which is described in the TNF package.

Input Arguments

You must specify the following input argument:

XName is a character string that specifies the name of the X variable.

RNNSPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the RNNSPEC object to prepare the RNN model to be used in a TSM object or to be imported to a TSMSPEC object for printing or for storage in a model repository.

Arguments

There are no arguments associated with this method.

RNNSPEC.GetLabel Method

```
rc=obj.GetLabel () ;
```

Retrieves the label of the specified RNN model and stores it in the return variable, *rc*.

Arguments

There are no arguments associated with this method.

RNNSPEC.Open Method

```
rc=obj.Open () ;
```

Initializes an empty RNNSPEC object for configuration.

Arguments

There are no arguments associated with this method.

RNNSPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... > ) ;
```

Specifies RNN model options. Options are ('Name', Value) pairs, where 'Name' is a case-insensitive character string and Value depends on the 'Name'.

Input Arguments

You must specify at least one of the following 'Names' and its associated Value:

'BATCHSELECTION'	takes a string Value that specifies how to make a minibatch data set for training networks. You can specify one of the following values: RANDOM RAN selects the minibatch data set randomly. SEQUENCE SEQ selects the minibatch data set sequentially according to the order of observations in the training data. SHUFFLE SHUF selects the minibatch data set sequentially after shuffling the training data. The default is SHUFFLE.
'FCSTLOWERBOUND'	takes a numeric Value that specifies a lower bound for the forecast. Forecast values that are below the specified value are truncated to the Value. A missing value indicates that no lower-bound truncation should occur. The default is a missing value.
'FCSTUPPERBOUND'	takes a numeric Value that specifies an upper bound for the forecast. Forecast values that are above the specified value are truncated to the Value. A missing value indicates that no upper-bound truncation should occur. The default is a missing value.
'NINPUT'	takes a positive integer Value that specifies the length of input series for the network. It is the length of the sampled subsequence for the input layer. The default is 6.
'NLAYER'	takes a positive integer Value that specifies the number of neural network layers. The first layer is the input layer, and the last layer is the output layer. The default is 3, which means that the neural network has one hidden layer.
'NNEURONH'	takes a positive integer Value that specifies the number of neurons for each hidden node. The default is 10.
'NORMALIZE'	takes a string Value that specifies a normalization method for the input sequence and the target sequence. You can specify one of the following values: MIDRANGE uses the midrange normalization, which scales variables so that the midrange is 0 and the half-range is 1. The result is that variables have a minimum value of -1 and a maximum value of 1. NONE means that variables are not modified.

STD uses the normal standardization, which scales variables so that the mean is 0 and the standard deviation is 1.

The default is STD.

'NVALIDATION' takes a nonnegative integer *Value* that specifies the number of validation samples. The validation sample is used to validate models. The default is 0. When the value is set to 0, the training error is used for model validation.

'OUTPUTINITBIAS' takes a numeric *Value* that specifies the initial bias for the output layer. The default is 0.

'POSTTRAIN' takes a string *Value* that specifies whether the optimization will run on the validation samples after the training process ends. You can specify one of the following values:

NO means that the post-training process is not performed.

YES means that the post-training process is performed.

The default is YES if the size of the validation sample is greater than 0. Otherwise, the default is NO.

'RECURRENTINITBIAS' takes a numeric *Value* that specifies the initial bias for the recurrent layers. The default is 0.

'RNNTYPE' takes a string *Value* that specifies a recurrent network type for the hidden layers. You can specify one of the following values:

GRU uses the gated recurrent unit.

LSTM uses the long short-term memory unit.

RNN uses the original recurrent neural network.

The default is LSTM.

'SEED' takes a nonnegative integer *Value* that specifies the seed to generate random numbers for the optimization process and data shuffling of mini-batch selection. When the value is 0, the seed is generated by using the time of day from the computer's clock. The default is 12345.

'USEERRORACF' takes a string *Value* that specifies whether to use the autocorrelation (ACF) to calculate forecast standard errors for forecast confidence intervals.

NO means that the standard error of the h -step forecast is calculated by the residual standard deviation multiplied by the square root of h .

YES means that the standard error of the h -step forecast is calculated by the residual standard deviation multiplied by the square root of the cumulative ACF up to the h lag.

The default is YES.

'WEIGHTINIT'

takes a string *Value* that specifies the weight initialization function for the neurons in each hidden layer. You can specify one of the following values:

CAUCHY	initializes weights so that the median is 0 and the scale is 1.
NORMAL	initializes weights so that the mean is 0 and the standard deviation is 1.
UNIFORM	initializes weights so that the mean is 0 and the half-range is 1.
XAVIER	initializes weights according to the method of Glorot and Bengio (2010) for a normal distribution.

The default is XAVIER.

RNNSPEC.SetOptimizer Method

```
rc=obj.SetOptimizer ('Name', Value <,'Name',Value,... > ) ;
```

Specifies optimizer options for the RNNSPEC object.

Input Arguments

You must specify at least one of the following '*Names*' and its associated *Value*:

'ALGORITHM'

takes a string *Value* that specifies one of the following stochastic gradient descent (SGD) optimization algorithms:

ADAM	uses adaptive moment estimation, which computes adaptive learning rates for each parameter.
MOMENTUM	uses an exponential smoothing model to accelerate SGD in the relevant direction and to dampen oscillation.
VANILLA	uses the plain stochastic gradient descent method.

The default is ADAM.

'BETA1'

takes a positive numeric *Value* that specifies the exponential decay rate for the first moment in the Adam learning algorithm. The value should be between 0 (inclusive) and 1 (exclusive). The default is 0.9.

'BETA2'

takes a positive numeric *Value* that specifies the exponential decay rate for the second moment in the Adam learning algorithm. The value should be between 0 (inclusive) and 1 (exclusive). The default is 0.999.

'CLIPGRADMAX'

takes a positive numeric *Value* that specifies the maximum gradient value. All gradient values that exceed the specified maximum value are set to the specified maximum value. The default is 1,000. When you specify the missing value, it means that there is no restriction on the gradient values.

'CLIPGRADMIN'	takes a negative numeric <i>Value</i> that specifies the minimum gradient value. All gradient values that are less than the specified minimum value are set to the specified minimum value. The default is -1,000. When you specify the missing value, it means that there is no restriction on the gradient values.
'DROPOUT'	takes a numeric <i>Value</i> that specifies the probability that the output of a neuron in a fully connected layer will be set to 0 during training. The specified probability is recalculated each time an observation is processed. The value should be between 0 (inclusive) and 1 (exclusive). The default is 0.
'DROPOUTINPUT'	takes a numeric <i>Value</i> that specifies the probability that an input variable will be set to 0 during training. The specified probability is recalculated each time an observation is processed. The value should be between 0 (inclusive) and 1 (exclusive). The default is 0.
'DROPOUTTYPE'	takes a string <i>Value</i> that specifies one of the following dropout types: INVERTED uses the inverted dropout. STANDARD uses the standard dropout. The default is STANDARD.
'EPSILON'	takes a positive numeric <i>Value</i> that specifies a very small number to prevent any division by zero in the Adam learning algorithm. The default is 1E-08.
'GAMMA'	takes a positive numeric <i>Value</i> that specifies the gamma parameter for the learning rate policy. The value should be between 0 and 1, inclusive. The default is 0.1.
'LEARNINGPOLICY'	takes a string <i>Value</i> that specifies one of the following learning rate policies: FIXED specifies a fixed learning rate. INV sets the learning rate parameter value after each epoch, according to the initial learning rate, the value of the gamma parameter, and the value of the power parameter. The rate is calculated as $\text{LEARNINGRATE} * (1 + \text{GAMMA} * \text{currentEpoch})^{-\text{POWER}}$ POLY sets the learning rate parameter value after each epoch, according to the initial learning rate, the maximum number of epochs, and the value of the power parameter. The rate is calculated as $\text{LEARNINGRATE} * \left(1 - \frac{\text{currentEpoch}}{\text{MAXEPOCHS}}\right)^{-\text{POWER}}$ STEP sets the learning rate parameter by multiplying the current learning rate value by the gamma parameter value. The

number of steps is specified in the `stepSize` parameter. The learning rate is recalculated for each group of epochs according to the step size. The rate is calculated as

$$\text{LEARNINGRATE} * \text{GAMMA}$$

The default is `FIXED`.

'LEARNINGRATE'	takes a positive numeric <i>Value</i> that specifies the learning rate for stochastic gradient descent. The default is 0.001.
'MAXEPOCHS'	takes a positive integer <i>Value</i> that specifies the maximum number of epochs. The default is 100.
'MINIBATCHSIZE'	takes a positive integer <i>Value</i> that specifies the number of observations per thread in a minibatch. The default is 1.
'MOMENTUM'	takes a numeric <i>Value</i> that specifies the momentum parameter value for stochastic gradient descent. The value should be between 0 and 1, inclusive. The default is 0.9.
'NMINIBATCH'	takes a positive integer <i>Value</i> that specifies the maximum number of minibatch sets. The default is the maximum number of 64-bit signed integers.
'POWER'	takes a nonnegative numeric <i>Value</i> that specifies the power for the learning rate policy. The default is 0.75.
'REGL1'	takes a nonnegative numeric <i>Value</i> that specifies the weight for the L1 regularization term. The default is 0, which means that the L1 regularization is not performed. It is suggested that the weight parameter begin with a small value, such as 1E-6. L1 regularization can be combined with L2 regularization.
'REGL2'	takes a nonnegative numeric <i>Value</i> that specifies the weight for the L2 regularization term. The default is 0, which means that the L2 regularization is not performed. It is suggested that the weight parameter begin with a small value, such as 1E-3. L2 regularization can be combined with L1 regularization.
'STAGNATION'	takes a nonnegative integer <i>Value</i> that specifies the number of successive training errors when a validation set is not given or the number of successive validation errors without improvement that will cause early termination. The default is 20. A value of 0 means that no stagnation early stopping is applied.
'STEPSIZE'	takes a positive integer <i>Value</i> that specifies the step size when the learning rate policy is set to <code>STEP</code> . The default is 10.
'THRESHOLD'	takes a nonnegative numeric value that specifies the threshold that is used to determine whether the validation score is improving or stagnating. When

$$\text{abs}(\text{currentScore} - \text{previousScore}) \leq \text{abs}(\text{currentScore}) * \text{THRESHOLD}$$

the current iteration does not improve the optimization and the stagnation counter is incremented. Otherwise, the stagnation counter is set to 0. When a validation (holdout) data set is not available, the training score is used. The default value is 1E-08, and the minimum value is 0.

'WARMUPEPOCHS'

takes a nonnegative integer *Value* that specifies the number of epochs for the learning rate warmup. The number should not exceed the maximum number of epochs. The default is 0, which means that there is no warmup.

UCMSPEC Object

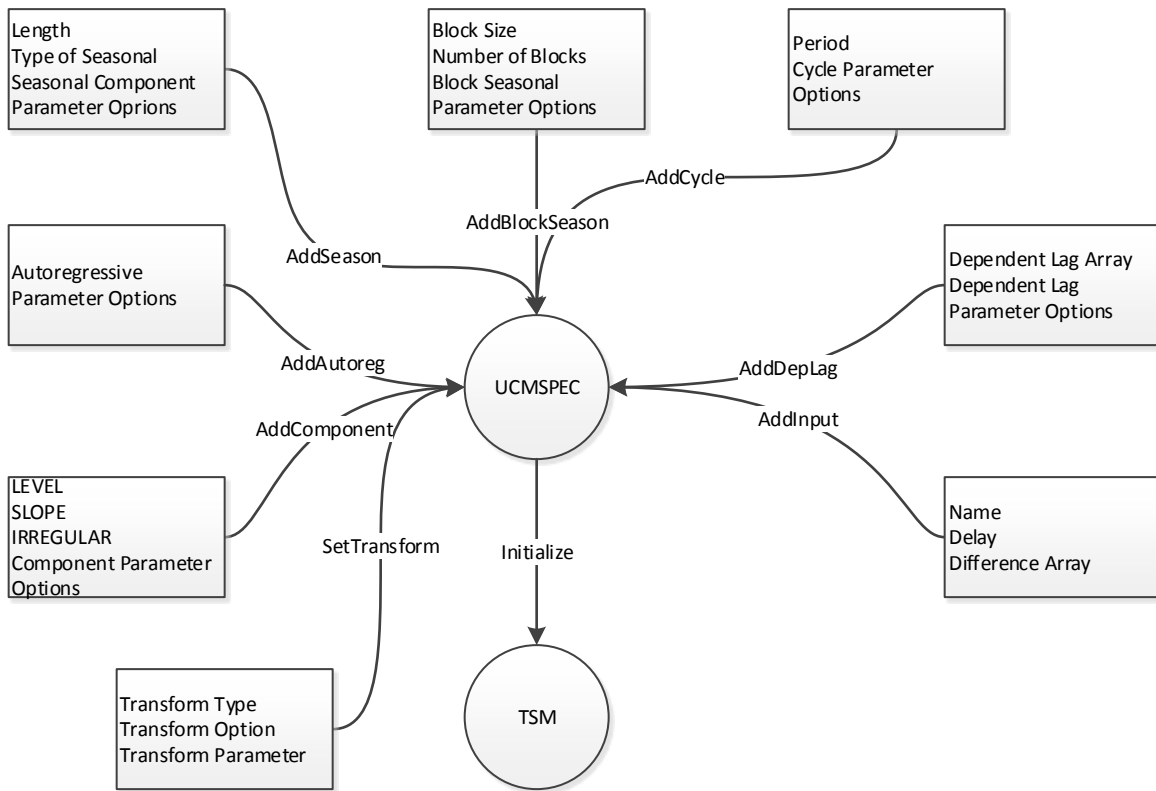
The unobserved component model (UCM) specification object generates UCM models for use with the TSM object. You open the UCMSPEC object to begin defining a new UCM model. You call the UCMSPEC methods to define the model characteristics, and then you close the UCMSPEC object to ready it for use in a TSM object. For more information, see the section “[TSM.Initialize Method](#)” on page 2352. You can also store the XML representation of the UCM in an output table. For more information, see the section “[TSMSPEC Object](#)” on page 2410. Note that the UCMSPEC object is equivalent to the HPFUCMSPEC procedure in *SAS Forecast Server Procedures: User’s Guide*.

Table 32.12 lists the methods that are associated with the UCMSPEC object.

Table 32.12 Methods of the UCMSPEC Object

Method	Description
AddAutoreg	Add an autoregressive component to a UCM model
AddBlockSeason	Add a block-season component to a UCM model
AddComponent	Add a fundamental component to a UCM model
AddCycle	Add a cycle component to a UCM model
AddDeplag	Add a deplag component to a UCM model
AddInput	Add an input component to a UCM model
AddSeason	Add a seasonal component to a UCM model
Close	Close the UCM model specification
GetLabel	Retrieve the label of the specified UCM model
Open	Open the UCM model specification
SetTransform	Specify the transform

Figure 32.8 illustrates the data flow through the UCMSPEC object.

Figure 32.8 UCMSPEC Object Data Flow

UCMSPEC Synopsis

DECLARE OBJECT *obj* (UCMSPEC) ;

Method syntax, in order of typical usage:

```

rc=obj.Open () ;
rc=obj.AddAutoreg (< Rho, NoestRho, Variance, Noest>) ;
rc=obj.AddBlockSeason (BlockSize, NBlocks, <'Type', Offset, Variance, Noest>) ;
rc=obj.AddComponent (Comp, < Variance, Noest>) ;
rc=obj.AddCycle (< Period, NoestPeriod, Rho, NoestRho, Variance, Noest>) ;
rc=obj.AddDeplag (LagArray, < NLag, CoeffArray, Noest>) ;
rc=obj.AddInput (XName, < Delay, DiffArray, NDiff, Transform, TransParm>) ;
rc=obj.AddSeason (Length, <'Type', Variance, Noest>) ;
rc=obj.SetTransform (XName, < Delay, DiffArray, NDiff, 'Type', TransParm>) ;
rc=obj.GetLabel () ;
rc=obj.Close () ;

```

UCMSPEC Methods

UCMSPEC.AddAutoreg Method

`rc=obj.AddAutoreg (< Rho, NoestRho, Variance, Noest>) ;`

Adds an autoregressive component to a UCM model specification. You can add at most one autoregressive component to a UCM model specification.

Input Arguments

You can specify the following input arguments:

<i>Rho</i>	takes a numeric value that specifies the starting value for the AR(1) coefficient during the parameter estimation process. The value of <i>Rho</i> must be in the interval $[-1, 1)$, which includes -1 but excludes 1 . The default is a missing value.
<i>NoestRho</i>	takes a Boolean value that specifies whether the AR(1) coefficient is to be estimated (0) or fixed (1) at the specified starting value. If the value of <i>NoestRho</i> is 1, you must specify a valid nonmissing <i>Rho</i> value. The default value is 0.
<i>Variance</i>	specifies an initial value for the disturbance variance during the parameter estimation process. If you specify this argument, then any nonnegative value, including 0, is acceptable. If you omit this argument or specify a missing value, then the estimation process computes a reasonable initial value.
<i>Noest</i>	takes a Boolean value that specifies whether the variance of the AR(1) noise process is to be estimated (0) or fixed (1). If the value of <i>Noest</i> is 1, you must specify a valid nonmissing <i>Variance</i> value. The default value is 0.

UCMSPEC.AddBlockSeason Method

`rc=obj.AddBlockSeason (BlockSize, NBlocks, <'Type', Offset, Variance, Noest>) ;`

Adds a block-season component to a UCM model specification. You can use this method repeatedly to add more block-season components to a UCM model specification.

Input Arguments

You must specify the following input arguments:

<i>BlockSize</i>	takes a numeric value that specifies the block size, where the value of <i>BlockSize</i> can be any integer greater than or equal to 2. Typical examples of block sizes are 24 (which corresponds to the hours of the day when a day is used as a block in hourly data) or 60 (which corresponds to the minutes in an hour when an hour is used as a block in data that are recorded by minutes).
<i>NBlocks</i>	takes a numeric value that specifies the number of blocks, where the value of <i>NBlocks</i> can be any integer greater than or equal to 2.

You can also specify the following input arguments:

'Type'	is a character string that specifies the type of the seasonal component. You can specify one of the following values: DUMMY specifies a dummy-type seasonal component. TRIG specifies a trigonometric seasonal component. The default is DUMMY.
Offset	takes a numeric value that specifies the offset (the position of the first measurement within the block, if the first measurement is not at the start of a block). The <i>Offset</i> value must be an integer between 1 and <i>BlockSize</i> . The default value is 1.
Variance	specifies an initial value for the disturbance variance during the parameter estimation process. If you specify this argument, then any nonnegative value, including 0, is acceptable. If you omit this argument or specify a missing value, then the estimation process computes a reasonable initial value.
Noest	takes a Boolean value that specifies whether the disturbance variance is to be estimated (0) or fixed (1). If the value of <i>Noest</i> is 1, you must specify a valid nonmissing <i>Variance</i> value. The default value is 0.

UCMSPEC.AddComponent Method

rc=obj.AddComponent (Comp, < Variance, Noest>);

Adds a fundamental component (such as LEVEL, SLOPE, or IRREGULAR) to a UCM model specification. This method must be called separately to add each of these fundamental components. A UCM model specification cannot contain more than one instance of these fundamental components.

Input Arguments

You must specify the following input argument:

Comp	is a character string that specifies the UCM component model. You can specify one of the following values: IRREGULAR includes an irregular component in the model that corresponds to the overall random error in the model. LEVEL includes a level component in the model. The level component, either by itself or together with a slope component, forms the trend component. If the slope component is absent, the resulting trend is a random walk (RW). If the slope component is present, then a locally linear trend (LLT) is obtained. SLOPE includes a slope component in the model. If you specify this value, you must call the method again with the value LEVEL for the <i>Comp</i> argument.
-------------	---

You can also specify the following input arguments:

Variance	specifies an initial value for the disturbance variance during the parameter estimation process. If you specify this argument, then any nonnegative value, including 0, is acceptable. If you omit this argument or specify a missing value, then the estimation process computes a reasonable initial value.
-----------------	---

Noest takes a Boolean value that specifies whether the disturbance variance is to be estimated (0) or fixed (1). If the value of *Noest* is 1, you must specify a valid nonmissing *Variance* value. The default value is 0.

UCMSPEC.AddCycle Method

rc=obj.AddCycle (< *Period*, *NoestPeriod*, *Rho*, *NoestRho*, *Variance*, *Noest*>) ;

Adds a cycle component to a UCM model specification. You can add multiple cycle components to a UCM model specification.

Input Arguments

You can specify the following input arguments:

Period specifies a numeric value as a starting value for the cycle period during the parameter estimation process, where the value of *Period* can be any number greater than 2. The default is a missing value.

NoestPeriod takes a Boolean value that specifies whether the cycle period is to be estimated (0) or fixed (1) at the specified starting value. If the value of *NoestPeriod* is 1, you must specify a valid nonmissing *Period* value. The default value is 0.

Rho takes a numeric value that specifies the starting value for the AR(1) coefficient during the parameter estimation process. The value of *Rho* must be in the interval (0, 1], which includes 1 but excludes 0. The default is a missing value.

NoestRho takes a Boolean value that specifies whether the AR(1) coefficient is to be estimated (0) or fixed (1) at the specified starting value. If the value of *NoestRho* is 1, you must specify a valid nonmissing *Rho* value. The default value is 0.

Variance specifies an initial value for the disturbance variance during the parameter estimation process. If you specify this argument, then any nonnegative value, including 0, is acceptable. If you omit this argument or specify a missing value, then the estimation process computes a reasonable initial value.

Noest takes a Boolean value that specifies whether the disturbance variance is to be estimated (0) or fixed (1). If the value of *Noest* is 1, you must specify a valid nonmissing *Variance* value. The default value is 0.

UCMSPEC.AddDeplag Method

rc=obj.AddDeplag (*LagArray*, < *NLag*, *CoeffArray*, *Noest*>) ;

Adds the lags of the dependent variable as predictors in the UCM specification.

Input Arguments

You must specify the following input argument:

LagArray specifies an integer array that defines a factor of lags of the dependent variable to be included in the deplag component of the UCM specification. The array must contain lags in ascending order.

You can also specify the following input arguments:

<i>NLag</i>	takes an integer that specifies the number of <i>LagArray</i> values to use. The default is all <i>LagArray</i> values.
<i>CoeffArray</i>	is a numeric array that specifies initial coefficients for the lagged dependent variables. This array must include initial values for all lag coefficients. The range of values of the lag coefficients are not constrained to lie in any region. If you specify this argument in the first invocation of the UCMSPEC.AddDeplag method, then you must also specify it in all subsequent invocations of the method. That is, you must either specify initial values for all lag coefficients across all factors of lags or specify initial values for none. By default, the estimation process computes a reasonable initial value for all coefficients.
<i>Noest</i>	takes a Boolean value that specifies whether lag coefficients are to be estimated (0) or fixed (1). The value of <i>Noest</i> must match across all subsequent invocations of the UCMSPEC.AddDeplag method. That is, lag coefficients across all factors of lags are either fixed or estimated from the data. If the value of <i>Noest</i> is 1, then the lag coefficients are fixed to the values that you specify in <i>CoeffArray</i> . Therefore, in this case, you must also specify the <i>CoeffArray</i> argument in all subsequent invocations of the UCMSPEC.AddDeplag method. The default value of <i>Noest</i> is 0.

Details

You can specify the lags of the dependent variable in the UCM specification as a set of polynomial factors. One call to the UCMSPEC.AddDeplag method adds one such polynomial factor. For each polynomial, you can specify the lags to include in the *LagArray* list. Each lag corresponds to the exponent of a backward shift operator that is included as an additive term in the corresponding polynomial factor. When the corresponding UCM is fitted, the polynomial factors of all UCMSPEC.AddDeplag calls are expanded, and the terms of the resulting polynomial correspond to lags of the dependent variable that are added as predictors to the fitted UCM.

This is best illustrated by an example. The following statements add to the UCM specification the simplest polynomial factor that includes only the first lag of the dependent variable as a predictor in the fitted model:

```
/* Create an instance of the UCM specification object */
DECLARE OBJECT obj(UCMSPEC);

/* Initialize an empty UCM specification */
rc = obj.open();

/* A UCM specification must have at least one stochastic component */
rc = obj.AddComponent('IRREGULAR');

/* Specify a polynomial factor that includes a single lag of */
/* the dependent variable as a predictor in the fitted UCM */
ARRAY lags1[1] / NOSYMBOLS;
lags1[1] = 1;

/* Specify the lag coefficient */
ARRAY coeff1[1] / NOSYMBOLS;
coeff1[1] = 0.1;

/* Add a new polynomial factor to the UCM specification */
rc = obj.AddDeplag(lags1, 1, coeff1, 1);
```

In particular, the preceding statements specify that the polynomial $(1 - \phi_{1,1}B)$ be included in the UCM specification and that the coefficient of the first lag in this polynomial (that is, $\phi_{1,1}$) be fixed at 0.1. Also, because the value of the *Noest* argument is 1, all subsequent calls to the UCMSPEC.AddDeplag method must also specify the value of *Noest* as 1 and supply lag coefficients in the *CoeffArray* argument. You can submit the following code to add more factors to the polynomial:

```
/* Specify a polynomial factor that includes three lags of */
/* the dependent variable as predictors in the fitted UCM */
ARRAY lags2[3] / NOSYMBOLS;
lags2[1] = 2;
lags2[2] = 12;
lags2[3] = 18;

/* Specify the coefficients for lags 2, 12, and 18 */
ARRAY coeff2[3] / NOSYMBOLS;
coeff2[1] = 0.3;
coeff2[2] = -1.2;
coeff2[3] = 0.7;

/* Specify a polynomial factor that includes two lags of */
/* the dependent variable as predictors in the fitted UCM */
ARRAY lags3[2] / NOSYMBOLS;
lags3[1] = 4;
lags3[2] = 24;

/* Specify the coefficients for lags 4 and 24 */
ARRAY coeff3[2] / NOSYMBOLS;
coeff3[1] = 0.2;
coeff3[2] = 0.5;

/* Add two more polynomial factors to the UCM specification */
rc = obj.AddDeplag(lags2, 3, coeff2, 1);
rc = obj.AddDeplag(lags3, 2, coeff3, 1);

/* Finalize the UCM specification (prepare it to be fitted) */
rc = obj.close();
```

The preceding statements, when you combine them with the first UCMSPEC.AddDeplag method call, specify that the polynomial $(1 - \phi_{1,1}B)(1 - \phi_{2,1}B^2 - \phi_{2,2}B^{12} - \phi_{2,3}B^{18})(1 - \phi_{3,1}B^4 - \phi_{3,2}B^{24})$ be included in the UCM specification. In addition, the statements specify that the coefficients of the 2nd, 4th, 12th, 18th, and 24th lags in this polynomial be fixed at 0.3, 0.2, -1.2, 0.7, and 0.5, respectively.

You can replace the last call to the UCMSPEC.AddDeplag method with the following statement in order to include only the first element of the lags3 array in the polynomial factor (that is, the 4th lag) and to specify that its coefficient be fixed at 0.2 and not be estimated:

```
rc = obj.AddDeplag(lags3, 1, coeff3, 1);
```

If you want to estimate the lag coefficient values from the data rather than specifying fixed ones, you can just specify the *LagArray* argument and replace all previous UCMSPEC.AddDeplag calls with the following:

```
rc = obj.AddDeplag(lags1);
rc = obj.AddDeplag(lags2);
rc = obj.AddDeplag(lags3);
```

For more information about the structure of the deplag component of a UCM specification, see the DEPLAG statement of PROC HPFUCMSPEC in *SAS Forecast Server Procedures: User's Guide*.

UCMSPEC.AddInput Method

```
rc=obj.AddInput (XName, < Delay, DiffArray, NDiff, Transform, TransParm>);
```

Adds an input (predictor variable) to a UCM model specification. You can add multiple inputs to a UCM model specification.

If you use the UCM specification to initialize a Time Series Model (TSM) object, then for each input that you add by using this AddInput method, you must also add that input to the TSM object by using the TSM.AddX method. Otherwise, the TSM object's model preparation fails when you run the TSM object.

Input Arguments

You must specify the following input argument:

XName is a character variable that specifies the name of the X-variable symbol.

You can also specify the following input arguments:

Delay takes a nonnegative integer value that specifies the lag of the X-variable in the model. If not specified, the *Delay* is 0.

DiffArray is numeric array of nonnegative integers that specifies differencing orders for the X-variable. If not specified, no differencing is applied to the predictor series.

NDiff takes a numeric variable that specifies the number of *DiffArray* values to use. If not specified, the cardinality of *DiffArray* is used.

Transform is character variable that specifies the name of a functional transform. You can specify one of the following values:

BOXCOX requests Box-Cox transformation. You can specify the control parameter for the transformation by using the optional *TransParm* argument, whose value must be between –5 and 5. The default value of *TransParm* is 1.

LOG requests logarithmic transformation.

LOGIT | LOGISTIC requests logistic transformation.

NONE does not apply a transformation.

SQRT requests square-root transformation.

The default is NONE.

TransParm takes a numeric value between –5 and 5 that specifies a transform parameter. This parameter is allowed only for Box-Cox transformations. The default is 1.

UCMSPEC.AddSeason Method

```
rc=obj.AddSeason (Length, <'Type', Variance, Noest>) ;
```

Adds a seasonal component to a UCM model specification. You can add multiple seasonal component to a UCM model specification.

Input Arguments

You must specify the following input argument:

Length is a numeric variable that specifies the length of a seasonal cycle.

You can also specify the following input arguments:

'Type' is a character string that specifies the type of the seasonal component. You can specify one of the following values:

DUMMY specifies a dummy type seasonal component.

TRIG specifies a trigonometric seasonal component.

The default is DUMMY.

Variance specifies an initial value for the disturbance variance during the parameter estimation process. If you specify this argument, then any nonnegative value, including 0, is acceptable. If you omit this argument or specify a missing value, then the estimation process computes a reasonable initial value.

Noest takes a Boolean value that specifies whether the disturbance variance is to be estimated (0) or fixed (1). If the value of *Noest* is 1, you must specify a valid nonmissing *Variance* value. The default value is 0.

UCMSPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the UCM model in the UCMSPEC object. This method prepares the UCM model for use in a TSM object or to be imported to a TSMSPEC object for printing or storage in a model repository catalog.

Arguments

There are no arguments associated with this method.

UCMSPEC.GetLabel Method

```
rc=obj.GetLabel () ;
```

Retrieves the label of the specified UCM model and stores it in the return variable, *rc*.

Arguments

There are no arguments associated with this method.

UCMSPEC.Open Method

```
rc=obj.Open () ;
```

Opens the UCMSPEC object for configuration. This method initializes an empty UCM model.

Arguments

There are no arguments associated with this method.

UCMSPEC.SetTransform Method

```
rc=obj.SetTransform ('Type' <','Option',Parm>) ;
```

Specifies the functional transform to be used by the UCM model.

Input Arguments

You must specify the following input argument:

'Type'	takes a string value that specifies the transform to use. You can specify one of the following values:
BOXCOX	requests Box-Cox transformation. You can specify the control parameter for the transformation by using the optional <i>Parm</i> argument, whose value must be between -5 and 5 . The default value of <i>Parm</i> is 1 .
LOG	requests logarithmic transformation.
LOGIT LOGISTIC	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	requests square-root transformation.
The default is NONE.	
'Option'	takes a string value that specifies prediction semantics for the inverse transform. You can specify the following values:
MEAN	requests that the inverse transform produce mean forecasts.
MEDIAN	requests that the inverse transform produce median forecasts.
The default is MEAN.	
<i>Parm</i>	takes a numeric value between -5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transformations. The default is 1 .

TSMPEST Object

The time series model parameter estimates (TSMPEST) collector object collects parameter estimates from a TSM object and stores them in an output table for printing or archiving or for use by a repeater object on another TSM object.

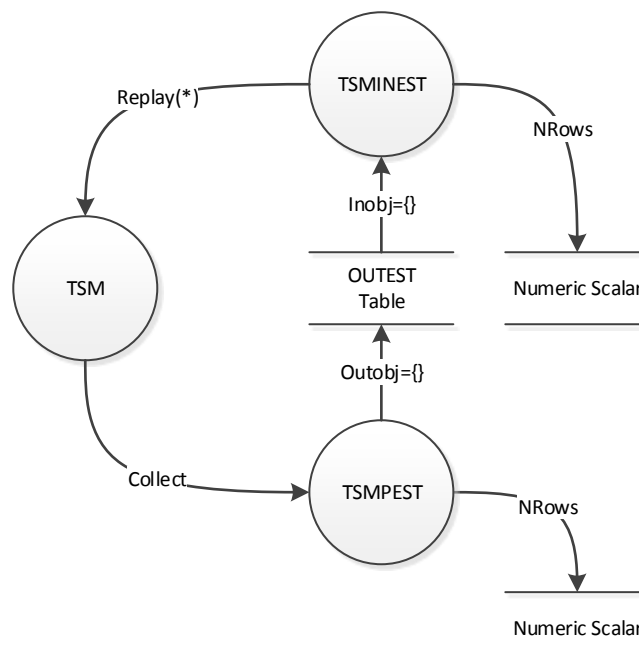
Table 32.13 summarizes the methods that are associated with the TSMPEST object.

Table 32.13 Methods of the TSMPEST Object

Method	Description
Collect	Collect parameter estimates from TSM objects
nrows	Return the number of rows that are collected
SetOption	Set options for the TSMPEST collector object

Figure 32.9 illustrates the data flow through the TSMPEST object.

Figure 32.9 TSMPEST Object Data Flow



TSMPEST Synopsis

DECLARE OBJECT *obj* (TSMPEST) ;

Method syntax, in order of typical usage:

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
rc=obj.Collect (TSMObj) ;
nrows=obj.nrows () ;
```

TSMPEST Methods

TSMPEST.Collect Method

```
rc=obj.Collect (TSMObj) ;
```

Collects time series model parameter estimates from a TSM object, *TSMObj*, and stores them in an output table.

Input Arguments

You must specify the following input argument:

TSMObj specifies the TSM object to use as the source of time series model parameter estimates.

TSMPEST.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the table.

Arguments

There are no arguments associated with this method.

TSMPEST.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
```

Specifies the options for the TSMPEST object.

Input Arguments

You must specify at least one of the following '*Names*' and its associated '*Value*':

- | | |
|-------------------------|---|
| 'MODELNAME' | takes a string <i>Value</i> that specifies the name of the model to be stored in the output table along with the parameter estimates of the model. By default, the model name is the name of the TSM object instance that is collected by the TSMPEST.Collect method. |
| 'FORCEMODELNAME' | takes a Boolean <i>Value</i> that specifies whether the model name should be stored in the output table even when the model failed to run. You can specify one of the following values: |

DISABLE|NO|N|FALSE|F|OFF does not store the model name in the output table if the model failed to run.

ENABLE|YES|Y|TRUE|T|ON always stores the model in the table.

The default is DISABLE.

TSMSPEC Object

The time series model specification (TSMSPEC) collector object collects the model specification from a TSM object, *TSMObj*, and stores it in an output table for printing or archiving or for use by a repeater object on another TSM object.

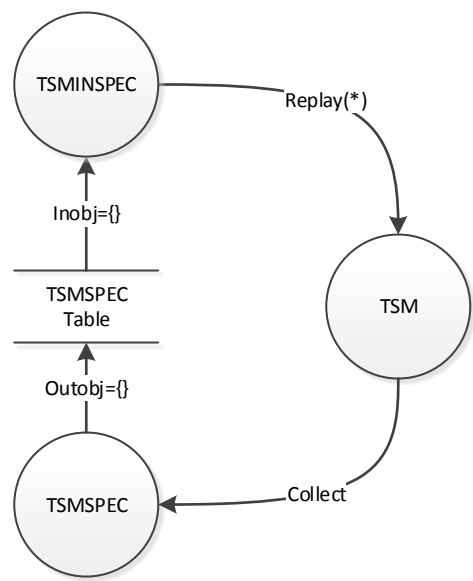
Table 32.14 summarizes the methods that are associated with the TSMSPEC object.

Table 32.14 Methods of the TSMSPEC Object

Method	Description
Collect	Collect model specification from TSM objects
nrows	Return the number of rows that are collected

Figure 32.10 illustrates the data flow through the TSMSPEC object.

Figure 32.10 TSMSPEC Object Data Flow



TSMSPEC Synopsis

DECLARE OBJECT *obj* (TSMSPEC) ;

Method syntax, in order of typical usage:

```
rc=obj.Collect (TSMObj) ;
nrows=obj.nrows () ;
```

TSMSPEC Methods

TSMSPEC.Collect Method

```
rc=obj.Collect (TSMObj) ;
```

Collects time series model specifications from a TSM object, *TSMObj*, and stores them in an output table.

Input Arguments

You must specify the following input argument:

TSMObj specifies the TSM object to use as the source of time series model parameter estimates.

TSMSPEC.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the table.

Arguments

There are no arguments associated with this method.

TSMFOR Object

The TSMFOR object collects forecast series from a TSM object, *TSMObj*, and stores them in an output table. The table schema that is used for storing the set of forecast series variables is compatible with the schema that the HPFENGINE procedure uses for the data set that is specified in the TSMFOR= option. Alternatively, the TSMFOR object can be configured to collect from a TSM object the smoothed (imputed) values for the dependent series and their associated standard errors and confidence limits.

Table 32.15 shows the contents of the TSMFOR object.

Table 32.15 Contents of the TSMFOR Object

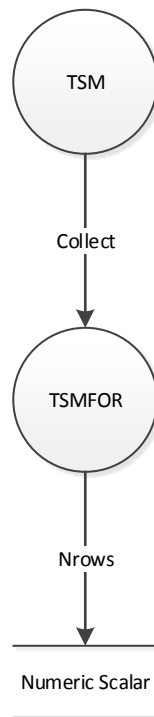
Column	Type	Description
NAME	String	Name of the dependent variable
MODEL	String	(Optional) Model name
TIMEID	Numeric	Uniform time ID values for series
ACTUAL	Numeric	Accumulated values of dependent variable
ERROR	Numeric	Residuals
LOWER	Numeric	Lower confidence limit
PREDICT	Numeric	Forecasts of the dependent variable
STD	Numeric	Prediction standard error
UPPER	Numeric	Upper confidence limit

Table 32.16 summarizes the methods that are associated with the TSMFOR object.

Table 32.16 Methods of the TSMFOR Object

Method	Description
Collect	Collect forecasts estimates from the TSM object
nrows	Return the number of rows collected
SetOption	Specifies options for the TSMFOR instance

Figure 32.11 illustrates the data flow through the TSMFOR object.

Figure 32.11 TSMFOR Object Data Flow

TSMFOR Synopsis

DECLARE OBJECT *obj* (TSMFOR <('Name', Value)>) ;

Method syntax, in order of typical usage:

```

rc=obj.SetOption (Name, Value) ;
rc=obj.Collect (ModelObj <,Region>) ;
rc=obj.nrows () ;

```

Input Arguments

You can optionally specify one or more of the following '*Names*' and its associated *Value*:

'MODELNAME' takes a string *Value* that specifies whether to add a model name column to the output table. The default model name is the TSM instance name. You can override the default model name via the 'MODELNAME' option of the [SetOption](#) method. You can specify one of the following *Values*:

YES	includes a model name column in the output table.
NO	does not include a model name column in the output table.

The default is NO.

'SMOOTH'	takes a string <i>Value</i> that specifies whether to output the smoothed (imputed) dependent series values and their associated standard errors and confidence limits. If you set this option to 'YES', the PREDICT column holds the values of the smoothed dependent series instead of the forecast series. You can specify one of the following <i>Values</i> :
YES	stores the smoothed dependent series and its associated statistics from a TSM instance in an output table.
NO	stores the forecast series and its associated statistics from a TSM instance in an output table.

The default is NO.

TSMFOR Methods

TSMFOR.Collect Method

```
rc=obj.Collect (ModelObj< ,Region> ) ;
```

Collects the forecast series from a TSM object, *ModelObj*. An optional *Region* argument can be specified to indicate the forecast region to be collected. If you specify the 'SMOOTH' option in the declaration of the TSMFOR instance, then the smoothed (imputed) dependent series is collected instead.

Input Arguments

You must specify the following input argument:

<i>ModelObj</i>	is a character string that specifies the name of the TSM object to use as the source of time series model forecasts.
-----------------	--

You can also specify the following input argument:

<i>Region</i>	specifies the time region over which to collect the forecast series. You can specify the following values for <i>Region</i> :
<i>string</i>	specifies the collection region. You can specify the following <i>strings</i> :
ALL	collects over the entire time span of the available data.
FIT	collects over the time region that supplied observations to estimate model parameters (that is, the model fit region).
FORECAST	collects over the time region that is subsequent to the FIT region and that did not contribute any data to the model parameter estimation process (that is, the model forecast region).

The default is ALL.

numeric is a two-valued numeric array, in which the first value specifies the starting time ID and the second value specifies the ending time ID of the time region over which the forecast series are to be collected. Either the starting time ID or the ending time ID can be a missing value. If both are missing values, then the default value ALL is used.

TSMFOR.nrows Method

```
rc=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the table.

Arguments

There are no arguments associated with this method.

TSMFOR.SetOption Method

```
rc=obj.SetOption ('Name', Value) ;
```

Specifies the named options for the TSMFOR instance.

Input Arguments

You must specify the following *'Name'* and its associated *Value*:

'MODELNAME' takes a string *Value* that is copied to the `_MODEL_` column of the output table. This option is relevant only if you set the 'MODELNAME' option to 'YES' in the declaration of the TSMFOR instance. The default value is the TSM instance name.

TSMSTAT Object

The TSMSTAT object collects model statistics of fit from a TSM object, *TSMObj*, and stores them in an output table. This information is useful for evaluating how well the model fits the dependent series. For more information about forecast fit statistics, see the section “Statistics of Fit” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*).

Table 32.17 shows the contents of the TSMSTAT object.

Table 32.17 Contents of the TSMSTAT Object

Column	Type	Description
NAME	String	Name of the dependent variable
REGION	String	Region in which the fit statistics are calculated. You can specify the following values: FIT indicates that fit statistics were calculated over the fit region. FORECAST indicates that fit statistics were calculated over the forecast (back) region. HOLDOUT indicates that fit statistics were calculated over the holdout region.
MODEL	String	Model name
AADJRSQ	Numeric	Amemiya's adjusted R-square
ADJRSQ	Numeric	Adjusted R-square
AIC	Numeric	Akaike's information criterion
AICC	Numeric	Finite sample corrected AIC
APC	Numeric	Amemiya's prediction criterion
DFE	Numeric	Degrees of freedom error
GMAPE	Numeric	Geometric mean absolute percentage error
GMAPES	Numeric	Geometric mean absolute error as a percentage of standard deviation
GMAPPE	Numeric	Geometric mean absolute predictive percentage error
GMRAE	Numeric	Geometric mean relative absolute error
GMASPE	Numeric	Geometric mean absolute symmetric percentage error
IMASE	Numeric	In-sample mean absolute scaled error
MAE	Numeric	Mean absolute error
MAPE	Numeric	Mean absolute percentage error
MAPES	Numeric	Mean absolute error as a percentage of standard deviation
MAPPE	Numeric	Mean absolute predictive percentage error
MASE	Numeric	Mean absolute scaled error
MAXAPES	Numeric	Maximum absolute error as a percentage of standard deviation
MAXERR	Numeric	Maximum error

Table 32.17 *continued*

Column	Type	Description
MAXPE	Numeric	Maximum percentage error
MAXPPE	Numeric	Maximum predictive percentage error
MAXRE	Numeric	Maximum relative error
MAXSPE	Numeric	Maximum symmetric percentage error
MDAPE	Numeric	Median absolute percentage error
MDAPES	Numeric	Median absolute error as a percentage of standard deviation
MDAPPE	Numeric	Median absolute predictive percentage error
MDASPE	Numeric	Median absolute symmetric percentage error
MDRAE	Numeric	Median relative absolute error
ME	Numeric	Mean error
MINAPES	Numeric	Minimum absolute error as a percentage of standard deviation
MINERR	Numeric	Minimum error
MINPE	Numeric	Minimum percentage error
MINPPE	Numeric	Minimum predictive percentage error
MINRE	Numeric	Minimum relative error
MINSPE	Numeric	Minimum symmetric percentage error
MPE	Numeric	Mean percentage error
MPPE	Numeric	Mean predictive percentage error
MRAE	Numeric	Mean relative absolute error
MRE	Numeric	Mean relative error
MSE	Numeric	Mean square error
MSPE	Numeric	Mean symmetric percentage error
N	Numeric	Number of observations that were used. This is the number of observations in the block of observations from the first to the last nonmissing error, which includes any embedded missing errors.
NMISSA	Numeric	Number of missing actual values
NMISSP	Numeric	Number of missing predicted values
NOBS	Numeric	Number of observations
NPARMS	Numeric	Number of parameters

Table 32.17 *continued*

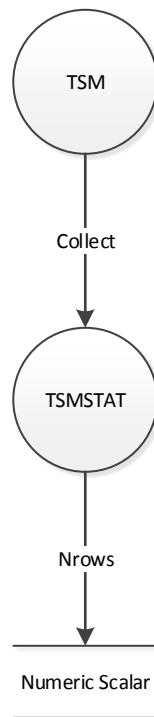
Column	Type	Description
RMSE	Numeric	Root mean square error
RMSSE	Numeric	Root mean square scaled error
RSQUARE	Numeric	R-square
RWRSQ	Numeric	Random walk R-square
SBC	Numeric	Schwarz Bayesian information criterion
SMAPE	Numeric	Symmetric mean absolute percentage error
SSE	Numeric	Sum of square error
SST	Numeric	Corrected total sum of squares
TSS	Numeric	Total sum of squares
UMSE	Numeric	Unbiased mean square error
URMSE	Numeric	Unbiased root mean square error

Table 32.18 summarizes the methods that are associated with the TSMSTAT object.

Table 32.18 Methods of the TSMSTAT Object

Method	Description
Collect	Collect model statistics of fit from the TSM object
nrows	Return the number of rows collected
SetOption	Set options for the TSMSTAT collector object

Figure 32.12 illustrates the data flow through the TSMSTAT object.

Figure 32.12 TSMSTAT Object Data Flow

TSMSTAT Synopsis

DECLARE OBJECT *obj* (TSMSTAT) ;

Method syntax, in order of typical usage:

```

rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.Collect (TSMObj) ;
nrows=obj.nrows () ;

```

TSMSTAT Methods

TSMSTAT.Collect Method

```
rc=obj.Collect (TSMObj) ;
```

Collects model statistics of fit from a TSM object, *TSMObj*, and stores them in an output table.

Input Arguments

You must specify the following input argument:

TSMObj specifies the TSM object to use as the source of model fit statistics.

TSMSTAT.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the table.

Arguments

There are no arguments associated with this method.

TSMSTAT.SetOption Method

```
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
```

Specifies the options for the TSMSTAT object.

Input Arguments

You must specify at least one of the following '*Names*' and its associated *Value*:

'MODELNAME' takes a string *Value* that specifies the name of the model to be stored to the output table along with the parameter estimates of the model. By default, the model name is the name of the TSM object instance that is collected by the [TSMSTAT.Collect](#) method.

'FORCEMODELNAME' takes a Boolean *Value* that specifies whether the model name should be stored in the output table even when the model failed to run. You can specify one of the following values:

DISABLE|NO|N|FALSE|F|OFF does not store the model name in the output table if the model failed to run.

ENABLE|YES|Y|TRUE|T|ON always stores the model in the output table.

The default is DISABLE.

TSMINEST Object

The time series model input estimates (TSMINEST) repeater object imports parameter estimates from a table for use in a TSM object. The TSMINEST table schema that is required for the parameter estimates is compatible with that used by the TSMPEST collector object. The TSMINEST object must be used in conjunction with a TSMINSPEC specification object and a TSM model object to replay a model by using its saved parameters (for an example, see [Example 32.3](#)).

Table 32.19 summarizes the methods of the TSMINEST object.

Table 32.19 Methods of the TSMINEST Object

Method	Description
<code>nrows</code>	Return number of rows in the TSMINEST object

TSMINEST Synopsis

```
DECLARE OBJECT obj (TSMINEST) ;
```

Method syntax:

```
rc=obj.nrows () ;
```

TSMINEST Methods

TSMINEST.nrows Method

```
rc=obj.nrows () ;
```

Returns the number of rows in the TSMINEST object, *obj*. A returned missing value indicates that the object has not been successfully configured.

Arguments

There are no arguments associated with this method.

TSMINSPEC Object

The time series model input specification (TSMINSPEC) repeater object imports model specifications from a table for use in a TSM object. The TSMINSPEC table schema that is required for the model specification is compatible with that used by the TSMSPEC collector object.

Table 32.20 summarizes the methods of the TSMINSPEC object.

Table 32.20 Methods of the TSMINSPEC Object

Method	Description
<code>nrows</code>	Return number of rows in TSMINSPEC object

TSMINSPEC Synopsis

DECLARE OBJECT *obj* (TSMINSPEC) ;

Method syntax:

rc obj.nrows () ;

TSMINSPEC Methods

TSMINSPEC.nrows Method

rc=obj.nrows () ;

Returns the number of rows that have been collected and stored in the table. A returned missing value indicates that the TSMINSPEC object has not been successfully configured.

Arguments

There are no arguments associated with this method.

TSMOUTOPT Object

The TSMOUTOPT object collects the optimization history, which contains training and validation errors, from a TSM object, *TSMObj*, and stores them in an output table. This information is useful for evaluating how well the model fits the dependent series. This object is available only for the RNNSPEC object.

Table 32.21 shows the contents of the TSMOUTOPT object.

Table 32.21 Contents of the TSMOUTOPT Object

Column	Type	Description
Epoch	Numeric	Epoch number
Train Error	Numeric	Training error
Valid Error	Numeric	Validation error

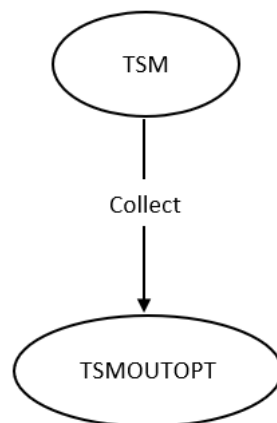
Table 32.22 summarizes the methods that are associated with the TSMOUTOPT object.

Table 32.22 Methods of the TSMOUTOPT Object

Method	Description
Collect	Collects the optimization history from the TSM object

Figure 32.13 illustrates the data flow through the TSMOUTOPT object.

Figure 32.13 TSMOUTOPT Object Data Flow



TSMOUTOPT Synopsis

DECLARE OBJECT *obj* (TSMOUTOPT) ;

Method syntax, in order of typical usage:

rc=obj.Collect (TSMObj) ;

TSMOUTOPT Methods

TSMOUTOPT.Collect Method

rc=obj.Collect (TSMObj) ;

Collects the optimization history from the TSM object, *TSMObj*, and stores it in an output table.

Input Arguments

You must specify the following input argument:

TSMObj specifies the TSM object to use as the source of the optimization history.

TSMRNNSTAT Object

The TSMRNNSTAT object collects RNN model statistics of fit from a TSM object, *TSMObj*, and stores them in an output table. This information is useful for evaluating how well the model fits the dependent series. This object is available only for the RNNSPEC object.

Table 32.23 shows the contents of the TSMRNNSTAT object.

Table 32.23 Contents of the TSMRNNSTAT Object

Column	Type	Description
NAME	String	Name of the dependent variable
OPTEPOCH	Numeric	Epoch number of the best model selected
PTVLDERROR	Numeric	Post-training validation error
TRNERROR	Numeric	Training error
VLDERROR	Numeric	Validation error

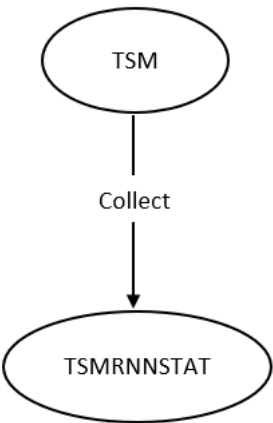
Table 32.24 summarizes the methods that are associated with the TSMRNNSTAT object.

Table 32.24 Methods of the TSMRNNSTAT Object

Method	Description
Collect	Collect model statistics of fit from the TSM object

Figure 32.14 illustrates the data flow through the TSMRNNSTAT object.

Figure 32.14 TSMRNNSTAT Object Data Flow



TSMRNNSTAT Synopsis

DECLARE OBJECT *obj* (TSMRNNSTAT) ;

Method syntax, in order of typical usage:

*rc=***obj.Collect** (TSMObj) ;

Methods of the RNNSPEC Object

TSMRNNSTAT Methods

TSMRNNSTAT.Collect Method

*rc=***obj.Collect** (TSMObj) ;

Collects RNN model statistics of fit from a TSM object, *TSMObj*, and stores them in an output table.

Input Arguments

You must specify the following input argument:

TSMObj specifies the TSM object to use as the source of model fit statistics.

Examples: TSM Package

Throughout this section, it is assumed that you have already started a CAS session and the data tables that are used in this section are stored in mylib, a CAS library that you have the necessary permissions to work with. This section assumes that you are familiar with the general workings of the TSMODEL procedure; for more information, see Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*).

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mylib CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Example 32.1: Fitting and Forecasting with ARIMA and ESM Models

The airline passenger data, given as Series G in Box and Jenkins (1976), have been used in time series analysis literature as an example of a nonstationary seasonal time series; for more information about ARIMA modeling of Series G, see “Example 7.2 Seasonal Model for the Airline Series” in the ARIMA chapter of *SAS/ETS User’s Guide*.

This example shows how you can use the objects in the TSM package to fit the airline model, $ARIMA(0,1,1) \times (0,1,1)_{12}$ NOINT and to fit the Winters exponential smoothing model to the airline series. In particular, it shows how you can do the following:

1. Create ARIMA and ESM specifications by using the ARIMASPEC and ESMSPEC specification objects.
2. Use these specification objects to initialize the TSM model objects.
3. Use these TSM model objects to fit the airline model and Winters model to the airline series, and to forecast the series according to these models.
4. Postprocess the forecast results to compute an ad hoc statistic.
5. Output the computed results to tables.

The broad outline of the code in this example is as follows:

1. The PROC TSMODEL statement specifies the input data set (mylib.air), a variety of output tables (mylib.airFor, mylib.airEst, and so on), and the forecast lead (12).
2. The ID statement specifies date as the time index variable, and the INTERVAL= option indicates that the data are monthly.
3. The VAR statement specifies the input data set variable, air, which contains the airline series.
4. The OUTARRAYS and OUTSCALARS statements declare some output arrays and scalars that are used to store the analysis results, which are subsequently saved as tables.
5. The REQUIRE statement specifies the TSM package, which is needed for the analysis.
6. The statements between the SUBMIT and ENDSUBMIT statements use the TSM package objects to perform the actual analysis in your CAS session.
7. These statements are grouped in three sections:
 - The first section does the specification, fitting, and forecasting according to the airline model. The airSpec object contains the airline specification, and the airModel object is a TSM model object that is initialized by using airSpec. After the airModel is run, the parameter estimates are collected in airEst and the forecasts are collected in airFor. The model residuals are stored in the airErr array for later processing.

- The second section does the specification, fitting, and forecasting according to the Winters exponential smoothing model. The `esmSpec` object contains the Winters specification, and the `esmModel` object is a TSM model object that is initialized by using `esmSpec`. The `esmModel` is run, and the model residuals are stored in the `esmErr` array for later processing.
- The third section computes an ad hoc statistic called `nbetter`, which counts the number of times the airline model residuals are smaller (in absolute size) than the Winters model. This section is included to illustrate how you can write your own custom postprocessing code to analyze the results that are produced by the TSM objects.

```
proc tsmodel data=mylib.air
    outobj=(airFor=mylib.airFor airEst=mylib.airEst)
    outscalar=mylib.nbetter outarray=mylib.out lead=12;
id date interval=month;
var air;
outarrays esmErr airErr;    **Store residuals of ESM and ARIMA models;
outscalars nbetter nfor;
require tsm;
submit;

    **Temporary work arrays used in ARIMA spec;
    array diff[2]/nosymbols;
    array ma[1]/nosymbols;

    *** Analysis based on the airline model ***;
    declare object airModel(tsm);
    declare object airSpec(arimaspec);

    **Set up the airline model spec:**;
    ** Model: log(air) ~ (0,1,1)(0,1,1)12 noint *;
    rc = airSpec.Open( );
    *** Specify differencing orders ***;
    diff[1] = 1;
    diff[2] = 12;
    rc = airSpec.SetDiff(diff,2);
    *** Specify moving average orders: q = (1)(12) ***;
    *** Use AddMAPoly twice for the two factors ***;
    ma[1] = 1;
    rc = airSpec.AddMAPoly(ma);
    ma[1] = 12;
    rc = airSpec.AddMAPoly(ma);
    *** Specify NOINT ***;
    rc = airSpec.SetOption('noint',1);
    *** Specify the log transform ***;
    rc = airSpec.SetTransform('log');
    *** Done setting up the ARIMA model ***;
    rc = airSpec.Close( );

    *** Set up and run the airModel TSM object ***;
    rc = airModel.Initialize(airSpec);
    rc = airModel.SetY(Air);
    rc = airModel.SetOption('lead',12);
    rc = airModel.Run( );
```

```

*** Output the airline model forecasts and estimates ***;
declare object airFor(tsmfor);
declare object airEst(tsmpest);
rc = airFor.Collect(airModel);
rc = airEst.Collect(airModel);

***Put the airline model residuals in airErr array;
rc = airModel.getForecast('error',airErr);

*** Analysis based on ESM model ***;
declare object esmModel(tsm);
declare object esmSpec(esmspec);
rc = esmSpec.open( );
rc = esmSpec.SetOption('method', 'winters');
rc = esmSpec.close( );

***Set up and run the TSM object ***;
rc = esmModel.Initialize(esmspec);
rc = esmModel.SetY(Air);
rc = esmModel.SetOption('lead',12);
rc = esmModel.Run( );

**Put the ESM model residuals in esmErr array;
rc = esmModel.getForecast('error',esmErr);

***Compute an ad hoc statistic based on airErr and esmErr arrays;
nbetter = 0;
nfor = esmModel.nfor( );
do t=1 to nfor;
    if airErr[t] ^= . & esmErr[t] ^= . then do;
        if abs(airErr[t]) < abs(esmErr[t])
            then nbetter = nbetter + 1;
    end;
end;
endsubmit;
quit;

```

Output 32.1.1 shows the predictions, and [Output 32.1.2](#) shows the parameter estimates for the airline model.

Output 32.1.1 Airline Model Predictions (Partial Output)**Airline Model Predictions**

DATE	PREDICT	STD	UPPER	LOWER
JAN1961	450.4	16.9215	484.5	418.2
FEB1961	426.1	18.8590	464.2	390.3
MAR1961	480.1	24.0408	528.9	434.7
APR1961	492.8	27.2405	548.3	441.6
MAY1961	509.5	30.5863	572.0	452.1
JUN1961	584.2	37.6514	661.4	513.9
JUL1961	670.7	45.9957	765.3	585.1
AUG1961	668.2	48.4237	768.0	578.3
SEP1961	559.6	42.6271	647.7	480.7
OCT1961	498.3	39.7181	580.6	425.0
NOV1961	431.2	35.8240	505.5	365.2
DEC1961	478.9	41.3484	565.0	403.0

Output 32.1.2 Airline Model Parameter Estimates (Partial Output)**Airline Model Parameter Estimates**

EST	_STDERR_	_TVALUE_	_PVALUE_
0.3773	0.0820	4.6033	9.828E-6
0.5724	0.0780	7.3361	2.17E-11

Output 32.1.3 shows the number of times the airline model residuals are smaller than the Winters model residuals.

Output 32.1.3 Number of Times the Airline Model Residuals Are Smaller Than the Winters Model Residuals

<u>nbetter</u>
71.0000

Example 32.2: Fitting a Transfer Function Model

This example uses the gas furnace data from Box and Jenkins (1976). The data, called Series J by Box and Jenkins, contain sequentially recorded measurements of two variables: x , the input gas rate, and y , the output CO_2 . The data also include an index variable, time, which keeps track of the sequence number of each observation (essentially the row index). The TSMODEL procedure requires an ID variable that has a valid time interval associated with it to index the observations. In order to satisfy this requirement, the time variable is assigned as the time ID variable in the ID statement and its interval is specified as SECOND using the INTERVAL= option. The value INTERVAL=SECOND is one of the simplest interval types for sequential indexing. As shown in “Example 7.3 Model for Series J Data from Box and Jenkins” in the ARIMA chapter of *SAS/ETS User’s Guide*, a reasonable ARIMAX model for y turns out to be $y = \text{TFinput}(x) + \text{AR}(2)$, where $\text{TFinput}(x)$ is a transfer function term in x with a delay of 3, numerator polynomial of order 2, and

denominator polynomial of order 1, and where AR(2) is an error term of autoregressive order 2. The following statements show how to fit this model by using the objects in the TSM package:

```
proc tsmodel data=mylib.seriesj
  outobj=(jEst=mylib.jEst);
  id time interval=second;
  var x y;
  require tsm;
  submit;

  *** Transfer function modeling for seriesJ ***;
  declare object jModel(tsm);
  declare object jSpec(arimaspec);
  declare object jEst(tsmpest);

  array num[2]/nosymbols;
  array den[1]/nosymbols;
  array ar[2]/nosymbols;

  *** Set up the transfer function model spec: ***;
  rc = jSpec.Open( );

  *** Specify AR orders: p = (1 2) ***;
  ar[1] = 1;
  ar[2] = 2;
  rc = jSpec.AddARPoly(ar);

  rc = jSpec.AddTF('x', 3); *delay=3;
  num[1] = 1;
  num[2] = 2;
  rc = jSpec.AddTFNumPoly('x', num);
  den[1] = 1;
  rc = jSpec.AddTFDenPoly('x', den);
  *** done setting up ARIMA model ***;
  rc = jSpec.Close( );

  *** Set up and run the TSM object ***;
  rc = jModel.Initialize(jSpec);
  rc = jModel.SetY(y);
  rc = jModel.AddX(x);
  rc = jModel.Run( );

  *** output gas furnace model forecasts and estimates ***;
  rc = jEst.Collect(jModel);
endsubmit;
quit;
```

Output 32.2.1 shows the parameter estimates for the transfer function model.

Output 32.2.1 Parameter Estimates for the Transfer Function Model (Partial Output)

<u>_EST_</u>	<u>_STDERR_</u>	<u>_TVALUE_</u>	<u>_PVALUE_</u>
53.2630	0.1193	446.5	0
1.5329	0.0475	32.2472	6.25E-97
-0.6330	0.0501	-12.6434	2.27E-29
-0.5352	0.0748	-7.1534	7.21E-12
0.3760	0.1029	3.6553	0.000306
0.5189	0.1078	4.8124	2.425E-6
0.5484	0.0382	14.3499	1.72E-35

Example 32.3: Replaying a Previously Fitted Model

In some cases, it is useful to save the model specification and parameter estimates that are computed during an analysis for later use. For example, you can use the saved model specification and parameter estimates to produce model forecasts at a later stage (possibly with new measurements appended to the original data). This example shows how you can do the following:

1. Save the model specification and parameter estimates for later use by using the Collect method of the TSMSPEC and TSMPEST objects, respectively.
2. Reuse the previously saved model specification and parameter estimates to configure a TSM object by using the Replay method.
3. Produce the model forecasts by using this TSM object.

The following statements fit the airline model to the airline series (see [Example 32.1](#) for more information about the airline series and the airline model). The model specification and parameter estimates are stored in the tables mylib.airOSpec and mylib.airEst, respectively.

```
proc tsmmodel data=mylib.air
  outobj=(airEst=mylib.airEst airOSpec=mylib.airOSpec) ;
  id date interval=month;
  var air;
  require tsm;
  submit;

  *** Analysis based on airline model ***;
  declare object airModel(tsm);
  declare object airSpec(arimaspec);
  declare object airEst(tsmpest);
  declare object airOSpec(tsmspec);

  array diff[2]/nosymbols;
  array ma[1]/nosymbols;

  *** Set up the airline model spec:   ***;
  ** Model: log(air) ~ (0,1,1) (0,1,1)12 noint ***;
  rc = airSpec.Open();
```

```

*** Specify differencing orders ***;
diff[1] = 1;
diff[2] = 12;
rc = airSpec.SetDiff(diff,2);
*** Specify moving average orders: q = (1)(12) ***;
*** Use AddMAPoly twice for the two factors ***;
ma[1] = 1;
rc = airSpec.AddMAPoly(ma);
ma[1] = 12;
rc = airSpec.AddMAPoly(ma);
*** Specify NOINT ***;
rc = airSpec.SetOption('noint',1);
*** Specify the log transform ***;
rc = airSpec.SetTransform('log');
*** Done setting up the ARIMA model ***;
rc = airSpec.Close();

*** Set up and run the TSM object ***;
rc = airModel.Initialize(airSpec);
rc = airModel.SetY(Air);
rc = airModel.SetOption('lead',12);
rc = airModel.Run();

*** Output airline model spec and estimates ***;
rc = airEst.Collect(airModel);
rc = airOSpec.Collect(airModel);
endsubmit;
quit;

```

Output 32.3.1 shows the parameter estimates that are saved in mylib.airEst.

Output 32.3.1 Parameter Estimates for the Airline Model (Partial Output)

Airline Model Parameter Estimates

<u>_EST_</u>	<u>_STDERR_</u>	<u>_TVALUE_</u>	<u>_PVALUE_</u>
0.3773	0.0820	4.6033	9.828E-6
0.5724	0.0780	7.3361	2.17E-11

The following statements show how to forecast the airline series by using the previously saved model specification (mylib.airOSpec) and parameter estimates (mylib.airEst).

```

proc tsmode1 data=mylib.air
    outobj=(airFor=mylib.airFor)
    inobj=(airEst=mylib.airEst airSpec=mylib.airOSpec) ;
    id date interval=month;
    var air;
    require tsm;
    submit;

    *** Analysis based on the airline model ***;
    declare object airModel(tsm);
    declare object airSpec(tsm1nspec);
    declare object airEst(tsm1nrest);

```

```

declare object airFor(tsmfor);

*** Set up and run the TSM object ***;
rc = airModel.Initialize();
rc = airModel.SetY(Air);
rc = airModel.SetOption('lead',12);
rc = airModel.Replay(airSpec,airEst);
rc = airModel.Run();

*** Output the airline model forecasts ***;
rc = airFor.Collect(airModel);
endsubmit;
quit;

```

Output 32.3.2 shows the forecasts that are produced according to the fitted model.

Output 32.3.2 Replayed Forecasts (Partial Output)

Airline Model Predictions

DATE	PREDICT	STD	UPPER	LOWER
JAN1961	450.4	16.9215	484.5	418.2
FEB1961	426.1	18.8590	464.2	390.3
MAR1961	480.1	24.0408	528.9	434.7
APR1961	492.8	27.2405	548.3	441.6
MAY1961	509.5	30.5863	572.0	452.1
JUN1961	584.2	37.6514	661.4	513.9
JUL1961	670.7	45.9957	765.3	585.1
AUG1961	668.2	48.4237	768.0	578.3
SEP1961	559.6	42.6271	647.7	480.7
OCT1961	498.3	39.7181	580.6	425.0
NOV1961	431.2	35.8240	505.5	365.2
DEC1961	478.9	41.3484	565.0	403.0

Example 32.4: Performing Time Series Imputation Using an ARIMA Model

Time series imputation is the process of replacing missing values in a time series with reasonable values that reflect the existing pattern in the available data (such as trend, seasonal variations, or long-term cyclical variations). It is a popular technique that is used across various domains of science. This example shows how you can do the following:

- Fit an ARIMA model to a time series that contains missing values by using a TSM object.
- Store the imputed values of the time series in a table.

For the purposes of this example, the following statements introduce two artificial missing values into the airline series (see [Example 32.1](#) for more information about the airline series and the airline model). The modified series is stored in the `mylib.airMiss` table. These statements assume that your engine libref is named `mylib`, but you can substitute any appropriately defined engine libref.

```

data mylib.airmiss;
  set mylib.air;
  airmiss = air;
  if date = '01JUL1955'd then airmiss = .;
  if date = '01AUG1955'd then airmiss = .;
run;

```

The following statements plot the modified airline series along with the two artificial missing values that were introduced. The results are shown in [Output 32.4.1](#).

```

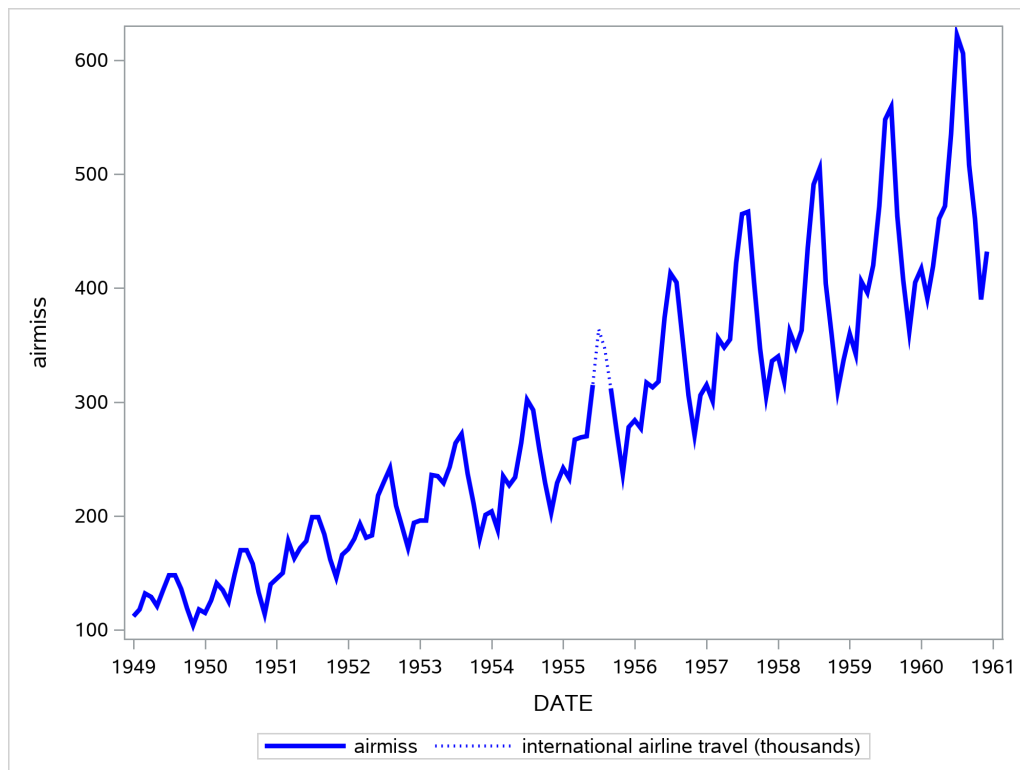
proc sort data=mylib.airmiss out=airmiss;
  by date;
run;

proc sgplot data=airmiss;
  series x=date y = airmiss / break lineattrs=(color=blue thickness=3);
  series x=date y = air / lineattrs=(thickness=2 pattern=dot color=blue);
run;

```

In [Output 32.4.1](#), the solid blue line represents all the nonmissing data values in the modified airline series. The dotted blue line between the years 1955 and 1956 represents the two actual values that were set to missing values.

Output 32.4.1 Airline Passenger Time Series with Artificial Missing Values



The following statements fit the airline model to the modified airline series and store the imputed time series in the mylib.airImpute table:

```

proc tsmodel data=mylib.airmiss
  outobj=(airImpute=mylib.airImpute) ;
  id date interval=month;
  var airmiss;
  require tsm;
  submit;

  *** Analysis based on airline model ***;
  declare object airModel(tsm);
  declare object airSpec(arimaspec);
  declare object airImpute(tsmfor('SMOOTH','YES'));

  array diff[2]/nosymbols;
  array ma[1]/nosymbols;

  *** Set up the airline model spec: ***;
  ** Model: log(air) ~ (0,1,1)(0,1,1)12 noint ***;
  rc = airSpec.Open();
  *** Specify differencing orders ***;
  diff[1] = 1;
  diff[2] = 12;
  rc = airSpec.SetDiff(diff,2);
  *** Specify moving average orders: q = (1)(12) ***;
  *** Use AddMAPoly twice for the two factors ***;
  ma[1] = 1;
  rc = airSpec.AddMAPoly(ma);
  ma[1] = 12;
  rc = airSpec.AddMAPoly(ma);
  *** Specify NOINT ***;
  rc = airSpec.SetOption('noint',1);
  *** Specify the log transform ***;
  rc = airSpec.SetTransform('log');
  *** Done setting up the ARIMA model ***;
  rc = airSpec.Close();

  *** Set up and run the TSM object ***;
  rc = airModel.Initialize(airSpec);
  rc = airModel.SetY(Airmiss);
  rc = airModel.Run();

  *** Output the imputed airline time series ***;
  rc = airImpute.Collect(airModel);
endsubmit;
quit;

```

The following statements plot the modified airline series along with its imputed version. The results are shown in [Output 32.4.2](#).

```

data mylib.airimpute;
  merge mylib.airimpute mylib.airmiss;
  by date;
run;

```

```

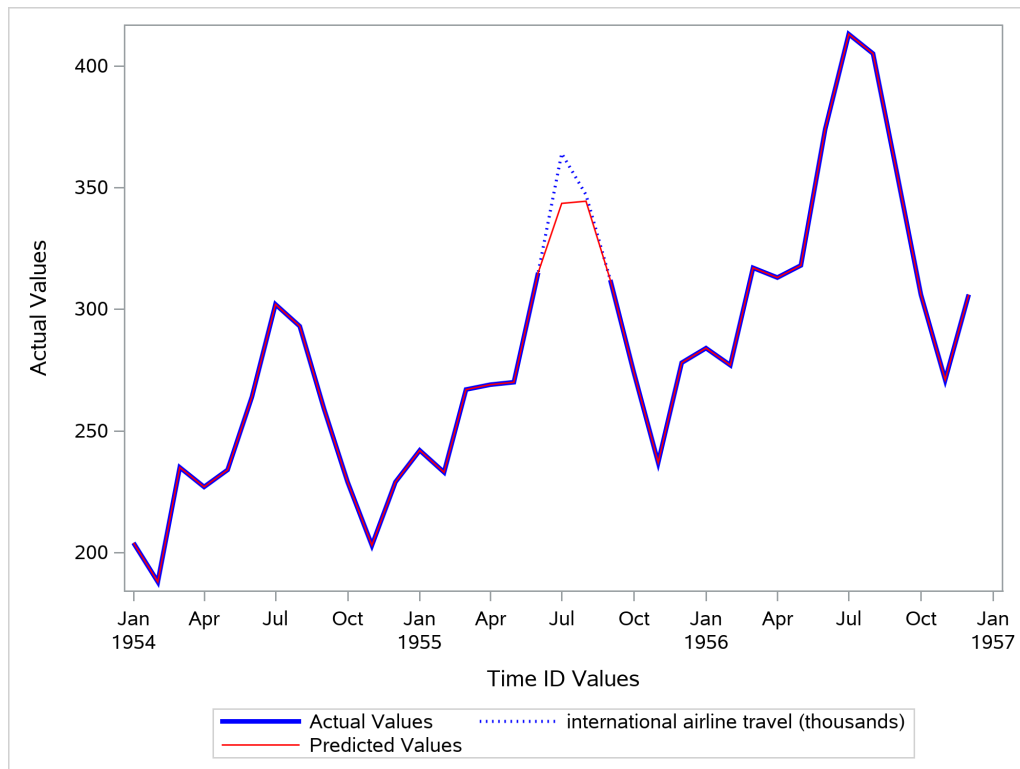
proc sort data=mylib.airimpute out=airimpute;
  by date;
run;

proc sgplot data=airimpute;
  series x=date y = actual / break lineattrs=(thickness=3 color=blue);
  series x=date y = air / lineattrs=(thickness=2 pattern=dot color=blue);
  series x=date y = predict / lineattrs=(thickness=1 color=red);
  where year(date) >= 1954 and year(date) <= 1956;
run;

```

In [Output 32.4.2](#), the thick blue line represents the modified airline series, and the thin red line represents its imputed version. The time axis range is restricted to the dates between the years 1954 and 1956 in order to facilitate the visualization. Notice how the thin red line (imputed values) closely mimics the dotted blue line (original actual values) over the region between the years 1955 and 1956 where the artificial missing values were introduced.

Output 32.4.2 Imputed Airline Passenger Time Series



Example 32.5: Combining Forecasts

This example shows how you can use the objects in the TSM package to combine multiple model forecasts to generate an improved forecast. In particular, it shows how you can do the following:

1. Fit the airline model, $\text{ARIMA}(0,1,1) \times (0,1,1)_{12}$ NOINT, to the airline series.
2. Fit the Winters exponential smoothing model to the airline series.
3. Use the CFC object to produce a combined forecast that is more accurate than the individual forecasts.
4. Store fit statistics for all models in an output table for later comparison.

```
proc tsmodel data=mylib.air outobj=(fitstats=mylib.fitstats) lead=12;
  id date interval=month;
  var air;
  require tsm;
  submit;

  **Temporary work arrays used in ARIMA spec;
  array diff[2]/nosymbols;
  array ma[1]/nosymbols;

  *** Declare a collector object for fit statistics ***;
  declare object fitstats(tsmstat);

  *** Perform analysis based on airline model ***;
  declare object airModel(tsm);
  declare object airSpec(arimaspec);

  **Set up the airline model spec:**;
  ** Model: log(air) ~ (0,1,1)(0,1,1)12 noint *;
  rc = airSpec.Open( );
  *** Specify differencing orders ***;
  diff[1] = 1;
  diff[2] = 12;
  rc = airSpec.SetDiff(diff,2);
  *** Specify moving average orders: q = (1)(12) ***;
  *** Use AddMAPoly twice for the two factors ***;
  ma[1] = 1;
  rc = airSpec.AddMAPoly(ma);
  ma[1] = 12;
  rc = airSpec.AddMAPoly(ma);
  *** Specify NOINT ***;
  rc = airSpec.SetOption('noint',1);
  *** Specify the log transform ***;
  rc = airSpec.SetTransform('log');
  *** Done setting up the ARIMA model ***;
  rc = airSpec.Close( );

  *** Set up and run the TSM object ***;
  rc = airModel.Initialize(airSpec);
```



```

rc = airModel.SetY(Air);
rc = airModel.SetOption('lead', 3);
rc = airModel.SetOption('back', 3);
rc = airModel.Run( );

*** Output airline model statistics of fit ***;
rc = fitstats.Collect(airModel);

*** Perform analysis based on ESM model ***;
declare object esmModel(tsm);
declare object esmspec(esmspec);
rc = esmspec.Open( );
rc = esmspec.SetOption('method', 'winters');
rc = esmspec.Close( );

*** Set up and run the TSM object ***;
rc = esmModel.Initialize(esmspec);
rc = esmModel.SetY(Air);
rc = esmModel.SetOption('lead', 3);
rc = esmModel.SetOption('back', 3);
rc = esmModel.Run( );

*** Output ESM model statistics of fit ***;
rc = fitstats.Collect(esmModel);

declare object airCFC(cfc);
rc = airCFC.Initialize( );
rc = airCFC.SetY(Air);
rc = airCFC.AddModel(airModel);
rc = airCFC.AddModel(esmModel);
rc = airCFC.SetOption('weight', 'ols');
rc = airCFC.SetOption('lead', 3);
rc = airCFC.SetOption('back', 3);
rc = airCFC.Run( );

*** Output combined model statistics of fit ***;
rc = fitstats.Collect(airCFC);
endsubmit;
quit;

```

Output 32.5.1 shows the value of the root mean square error (RMSE) fit statistic for the airline, ESM, and forecast combination models over both the fit and forecast regions. Lower RMSE values indicate a more accurate model. Notice how the combined model forecast improves in accuracy over both of the independent forecasts in both the fit and forecast regions.

Output 32.5.1 Fit Statistics for the Airline, ESM, and Forecast Combination Models**Root Mean Square Error (RMSE) Statistic**

<u>_MODEL_</u>	<u>_REGION_</u>	<u>RMSE</u>
airModel	FIT	10.7047
airModel	FORECAST	8.7732
esmModel	FIT	10.5826
esmModel	FORECAST	10.6018
airCFC	FIT	10.3719
airCFC	FORECAST	8.6551

Example 32.6: Recurrent Neural Network Forecasting

This example illustrates how to use the RNNSPEC object to fit a recurrent neural network forecasting model to the airline passenger data (see [Example 32.1](#) for more information about the airline series).

The following statements perform forecasting by using the LSTM recurrent neural network on the example data set. Specifying the value 12 for the 'NINPUT' argument means that the length of the input subsequence is 12. The value, 12, of the 'NVALIDATION' argument corresponds to the range of the validation period. The STD normalization is imposed. SetY specifies the AIR variable as the target variable. The number of layers is 4, which means that the neural network consists of one input layer, two hidden layers, and one output layer. The number of neurons in each hidden node is 30. When you specify 'ADAM' in the 'ALGORITHM' optimizer option, the stochastic gradient descent method uses the Adam optimization algorithm.

```
data mylib.airtrain;
    set sashelp.air;
run;

proc tsmodel data=mylib.airtrain nosummary outlog=mylib.outlog
    outobj=( outfor = mylib.outfor(replace=YES)
            outstat = mylib.outstat(replace=YES)
            outrnnstat = mylib.outrnnstat(replace=YES)
            outopt = mylib.outopt(replace=YES)
            );
var air;
id date interval=month;
require tsm;
submit;
    declare object tsmmodel(tsm);
    declare object rnnspec(rnnspec);
    declare object outrnnstat(tsmrnnstat);
        declare object outstat(tsmstat);
    declare object outopt(tsmoutopt);
    declare object outfor(tsmfor);
    rc = rnnSpec.Open( );
    rc = rnnSpec.SetOption(
        'RNNTYPE', 'LSTM',
        'NINPUT', 12,
        'NVALIDATION', 12,
        'NLAYER', 4,
```

```

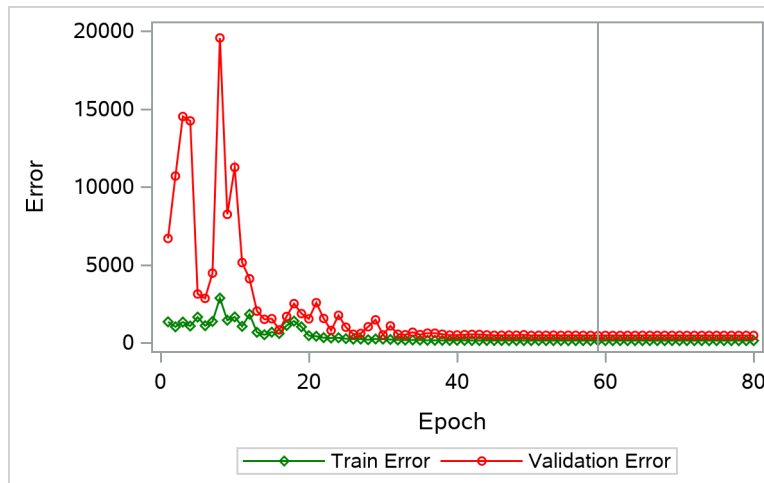
        'NNEURONH', 30,
        'NORMALIZE', 'STD',
        'POSTTRAIN', 'YES',
        'SEED', 12345
    );
    rc = rnnSpec.SetOptimizer(
        'ALGORITHM', 'ADAM',
        'LEARNINGRATE', 0.1,
        'BETA1', 0.8,
        'BETA2', 0.9,
        'LEARNINGPOLICY', 'STEP',
        'STEPsize', 5,
        'GAMMA', 0.5,
        'STAGNATION', 20,
        'MINIBATCHSIZE', 8,
        'WARMUPEPOCHS', 15,
        'MAXEPOCHS', 100
    );
    rc = rnnSpec.Close();
    rc = tsmmodel.Initialize(rnnSpec);
    rc = tsmmodel.SetY(air);
    rc = tsmmodel.SetOption('lead', 12);
    rc = tsmmodel.SetOption('alpha', 0.05);
    rc = tsmmodel.Run(); if rc < 0 then stop;
    rc = outfor.Collect(tsmmodel); if rc < 0 then stop;
    rc = outstat.Collect(tsmmodel); if rc < 0 then stop;
    rc = outrnnstat.Collect(tsmmodel); if rc < 0 then stop;
    rc = outopt.Collect(tsmmodel); if rc < 0 then stop;
    endsubmit;
    print outlog;
    run;

data null;
    set mylib.outrnnstat;
    call symputx('optepoch', OPTEPOCH);
run;

proc sgplot data=mylib.outopt;
    series x=epoch y=trnerror/ LegendLabel="Train Error"
        lineattrs=(color=green) markers
        markerattrs=(color=green symbol=diamond size=5);
    series x=epoch y=vlderror/ LegendLabel="Validation Error"
        lineattrs=(color=red) markers
        markerattrs=(color=red size=5);
    refline &optepoch/ axis=x ;
    yaxis label="Error";
run;

```

Output 32.6.1 shows the training and validation errors of the optimization history. The vertical reference line shows where the SGD optimum occurs.

Output 32.6.1 Training and Validation Errors at Each Epoch

The following statements print the output table that is produced by the TSMFOR collect method:

Output 32.6.2 shows part of the forecast value table of the target series, *air*, that is produced by the LSTM recurrent neural network. The forecast errors and confidence intervals are also shown in the table.

Output 32.6.2 Forecast Table for Airline Passenger Data

Obs	Variable Name	Time ID Values	Actual Values	Predicted Values	Prediction Errors	Prediction Standard Errors	Upper Confidence Limits	Lower Confidence Limits
140	AIR	AUG1960	606	580.52893066	25.471069336	13.172509395	606.34657466	554.71128666
141	AIR	SEP1960	508	507.43154907	0.5684509277	13.172509395	533.24919307	481.61390507
142	AIR	OCT1960	461	453.6411438	7.3588562012	13.172509395	479.4587878	427.8234998
143	AIR	NOV1960	390	406.93655396	-16.93655396	13.172509395	432.75419795	381.11890996
144	AIR	DEC1960	432	425.925354	6.0746459961	13.172509395	451.742998	400.10771
145	AIR	JAN1961	.	449.0279541	.	13.172509395	474.8455981	423.2103101
146	AIR	FEB1961	.	422.8638916	.	14.926328973	452.11895881	393.60882439
147	AIR	MAR1961	.	440.79019165	.	16.358533813	472.85232876	408.72805454
148	AIR	APR1961	.	503.12182617	.	16.9620821	536.36689619	469.87675615
149	AIR	MAY1961	.	504.62615967	.	17.142506943	538.22485588	471.02746345
150	AIR	JUN1961	.	569.63146973	.	17.406278837	603.74714935	535.5157901
151	AIR	JUL1961	.	604.09667969	.	17.499851019	638.39575742	569.79760196
152	AIR	AUG1961	.	590.88989258	.	17.553449512	625.29402143	556.48576373
153	AIR	SEP1961	.	543.19689941	.	17.575083777	577.64343064	508.75036819
154	AIR	OCT1961	.	491.10217285	.	17.60588388	525.60907117	456.59527453
155	AIR	NOV1961	.	430.46447754	.	17.672488783	465.10191907	395.82703601
156	AIR	DEC1961	.	456.65841675	.	17.984226252	491.90685249	421.409981

The following statements generate the plot of the actual values and forecast values:

```
proc sgplot data=mylib.outfor;
  series x=date y=actual /
    legendlabel="Actual Value" name="actual"
    lineattrs=(color=green) markers
```

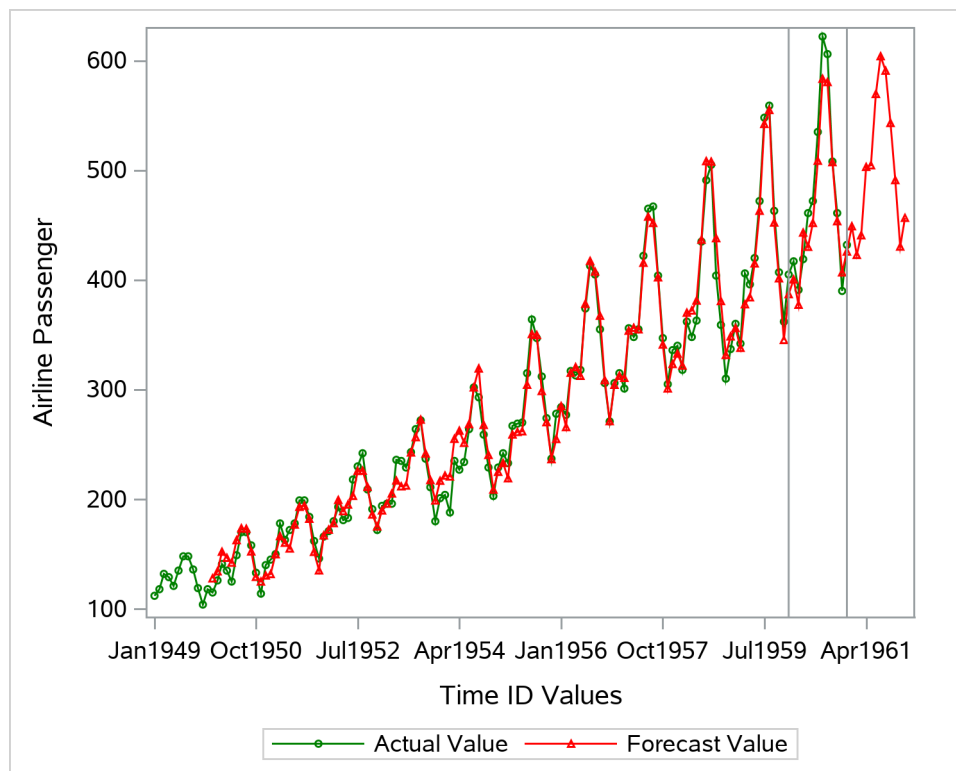
```

        markerattrs=(color=green size=4);
series x=date y=predict /
        LegendLabel="Forecast Value" name="forecast"
        lineattrs=(color=red) markers
        markerattrs=(color=red size=4 symbol=triangle);
xaxis values=('01jan49'd to '01dec61'd by month);
refline '01dec59'd/ axis=x;
refline '01dec60'd/ axis=x;
keylegend "actual" "forecast";
yaxis label="Airline Passenger";
run;

```

Output 32.6.3 shows the plot of forecasts and their actual values in the training and validation periods. It also shows forecasts of the 12 months after the validation period. The first reference line splits the training and validation periods, and the second reference line indicates the end of the validation period.

Output 32.6.3 Forecast Plot of Airline Passenger Data



References

- Box, G. E. P., and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*. Rev. ed. San Francisco: Holden-Day.
- Glorot, X., and Bengio, Y. (2010). "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>.

Chapter 33

Utility Package

Contents

Overview: UTL Package	2446
UTL Package Summary	2446
Using the UTL Package	2447
Common Argument Types	2447
Return Codes	2448
CLIMITS Object	2448
CLIMITS Synopsis	2449
CLIMITS Methods	2450
OUTNVP Object	2452
OUTNVP Synopsis	2453
OUTNVP Methods	2453
UTLSTAT Object	2454
UTLSTAT Synopsis	2457
UTLSTAT Methods	2458
UTL Object	2459
UTL Synopsis	2459
UTL Methods	2459
Examples: UTL Package	2460
Using CAS Sessions and CAS Engine Librefs	2460
Example 33.1: Collecting Forecast Fit Statistics and Ad Hoc Numeric Variables into Tables	2461
Example 33.2: Computing Prediction Standard Errors and Confidence Limits for an Ad Hoc External Forecast	2465

Overview: UTL Package

This chapter describes the various utility object classes that the utility (UTL) package contains. The UTL package provides a set of multipurpose tools that simplify the development of time series analysis programs for the TSMODEL procedure. The following types of computation are currently supported by the UTL package:

- computing prediction standard errors and confidence limits for specified actual and predicted time series
- computing and storing model forecast fit statistics in tables for specified actual and predicted time series
- storing ad hoc numeric variables that are defined in a user program in tables
- timing the execution of arbitrary portions of the time series analysis program that you submit as input to the TSMODEL procedure

The UTL package is object-oriented. To use the UTL package, you must declare instances of the object classes that the package contains. Declaring an object instance is the object-oriented equivalent of declaring a program variable. As with simple program variables, the declaration assigns the instance a name of your choosing and assigns a type, which is defined by the object's class. Unlike simple program variables, the object instance requires a different syntax for interacting with it and offers different functions (methods) that are contextual to the object. The object can offer highly sophisticated capabilities with a simple-to-use interface.

UTL Package Summary

Table 33.1 summarizes the objects in the UTL package.

Table 33.1 Objects in the UTL Package

Object	Description
CLIMITS	Object that computes prediction standard errors and confidence limits for specified actual and predicted time series
OUTNVP	Ad hoc name, value pair collector object for storing numeric scalar or array variables that are found in a user program in a table
UTL	Object that provides miscellaneous, stateless utility functions
UTLSTAT	Collector object for computing forecast fit statistics for specified actual and predicted time series and storing those statistics in a table

Using the UTL Package

The objects in the UTL package are divided into three categories:

- stateful computational objects (CLIMITS object)
- collector objects (OUTNVP and UTLSTAT objects)
- stateless functions (UTL object)

Collector objects provide a mechanism to create a snapshot of results (either from stateful objects or from plain program variables) and store those results in tables. Each collector object defines a table schema that is determined by the collector object's design. The collector objects in the UTL package (OUTNVP and UTLSTAT) follow a common method pattern. The basic execution follows this sequence of operations:

1. **Declare:** Create the collector object by using the object declaration statement.
2. **Collect:** Use the Collect method to store results in a table. The input arguments of the Collect method are specific to the collector object. For example, the UTLSTAT collector object's Collect method requires an actual and predicted time series as arguments. It then uses the specified series to compute forecast fit statistics and stores the results in a table. In contrast, the OUTNVP collector object's Collect method takes an ad hoc numerical scalar or array variable from the user program and stores it in a table. Rows that are collected are automatically appended to the collector's associated table at the end of each BY group, and the collector object's saved row set is automatically reset. The Nrows attribute returns the current row count in the collector. A missing value is returned if nothing has been collected. The data, now stored in tables, can then be used to produce reports or for further computations.

Common Argument Types

Table 33.2 defines the common argument types that are used in this chapter. The symbol *x* corresponds to the variable name.

Table 33.2 Common Argument Types

SAS Data Type	Declaration Syntax
Numeric	<i>x</i> or LENGTH <i>x</i> 8;
Numeric array	ARRAY <i>x</i> [<i>n</i>]/NOSYMBOLS;
Status	<i>x</i> or LENGTH <i>x</i> 8;
String	LENGTH <i>x</i> \$ <i>n</i> ;

Return Codes

Table 33.3 shows the return code (*rc* in method statements) status values that are used in this package. These status code values are returned after a method that is associated with an object is called; they can help determine whether the method executed successfully.

Table 33.3 Return Codes

Status	Meaning
< 0	An unrecoverable error occurred. No result was produced.
= 0	Unconditional success. The requested action completed and a normal result was produced.
> 0	Conditional success or warning. A result was produced subject to conditions.

Upon returning a negative status code, most methods in the UTL package objects also write a message to the output log that explains the causes of the related failure. These messages provide useful information during the process of debugging a user program. In the TSMODEL procedure, the output log is stored in the table that is specified in the OUTLOG= option in the PROC TSMODEL statement. For more information about how to enable and configure logging and about how to access the output log after an invocation of the TSMODEL procedure, see Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*).

CLIMITS Object

The CLIMITS object provides a mechanism for computing both the prediction standard errors and confidence limits of an *external model* (that is, a user-defined model) forecast, which is described by a pair of actual and predicted time series. The first stage of the computational process involves validating both input series. This validation is accomplished by ascertaining that the actual and predicted series have nonmissing observation values under at least one matching index. In addition, the predicted series is checked for the presence of extreme values. Next, the prediction standard errors are computed from the prediction errors (that is, the model residuals). Finally, the confidence limits are computed from the prediction standard errors. You can optionally supply the value of the confidence level that is used to compute the confidence limits. Note that the CLIMITS object retains all computed results in its internal memory. Individual forecast series can be queried via the [GetForecast](#) method, which is a method in the CLIMITS object.

Table 33.4 summarizes the methods that are associated with the CLIMITS object.

Table 33.4 Methods of the CLIMITS Object

Method	Description
Compute	Compute the prediction standard errors and confidence limits for specified actual and predicted time series
GetForecast	Retrieve a computed forecast series by name

CLIMITS Synopsis

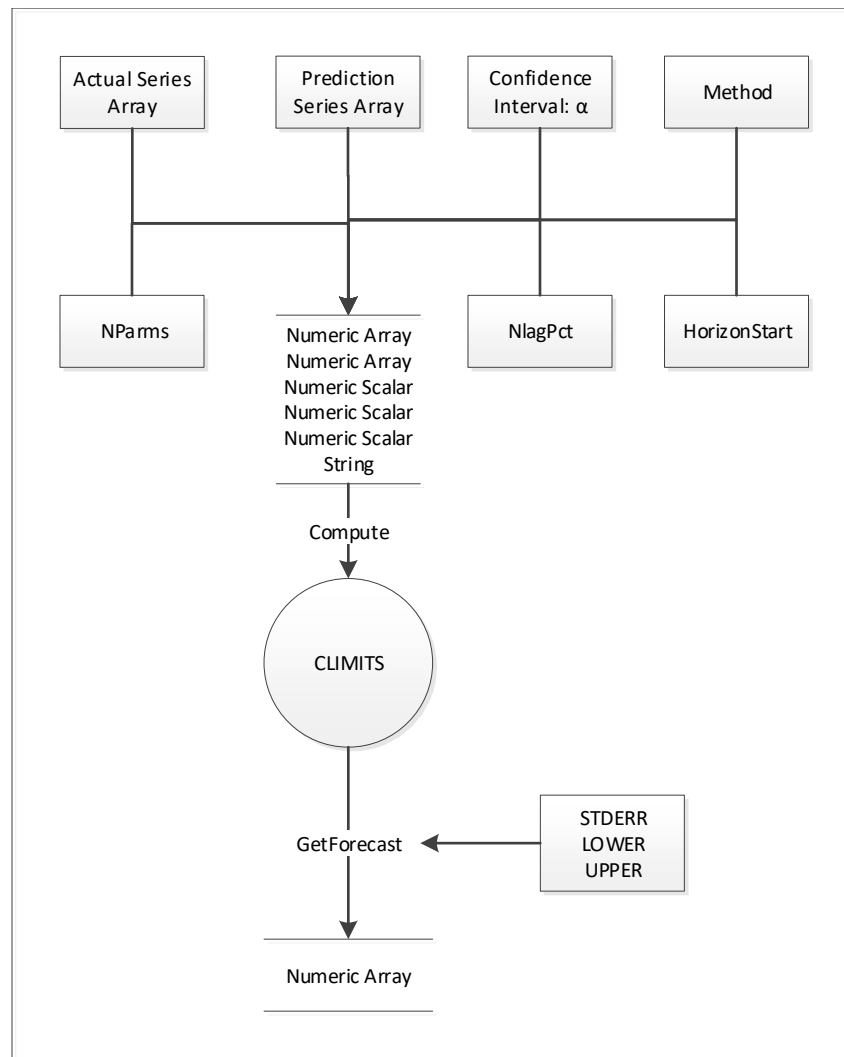
DECLARE OBJECT *obj* (CLIMITS) ;

Method syntax, in order of typical usage:

rc=obj.Compute (*Actual,Predicted,< Alpha>,< NParms>,< Method>,< NlagPct>,< HorizonStart>*
);

rc=obj.GetForecast (*Which,Result*);

Figure 33.1 outlines the programmatic data flow through the CLIMITS object; each arrow represents a different object method.

Figure 33.1 CLIMITS Object Data Flow

CLIMITS Methods

CLIMITS.Compute Method

`rc=obj.Compute (Actual, Predicted,< Alpha>,< NParms>, < Method>,< NlagPct>, < HorizonStart> ) ;`

Computes the prediction standard errors and confidence limits for specified actual and predicted time series. Both input series are validated by ascertaining the presence of nonmissing observation values under at least one matching index. Also, the predicted series is checked for the presence of extreme values. The computed forecast series are stored in the object's internal memory and can be individually queried via the [GetForecast](#) method into a numeric array defined in the user program. A negative return code indicates that the validation of an input series failed (for example, the predicted series has extreme values or all missing values), you specified an out-of-range *Alpha* argument value, or a computational failure occurred (for example, an out-of-memory error).

Input Arguments

You must specify the following input arguments:

- Actual* specifies a numeric array that corresponds to the actual time series.
- Predicted* specifies a numeric array that corresponds to the predicted time series.

You can also specify the following input arguments:

- Alpha* takes a numeric value between 0 and 1, exclusive, that specifies the significance level for forecast confidence bands. The default value is 0.05.

- HorizonStart* takes a SAS date, datetime, or time value that specifies the start of the forecast horizon. When you specify this argument, the effective forecast horizon for any particular BY group might differ from another because of differences in the lengths of the historical data across the BY groups, but all forecasts will start at the same date as determined by the *HorizonStart* value. When you specify a missing value (.), the start of the forecast horizon is automatically set as the first time period following the last nonmissing observation of the dependent series. The default value is a missing value.

- Method* takes a string that specifies the external model to be used to approximate the prediction standard errors. You can specify one of the following values:

- ACF** indicates that autocorrelation is used.
- ERRORACF** indicates that prediction error autocorrelation is used.
- NONE** indicates that no prediction error autocorrelation is used.
- PERFECT** indicates that perfect autocorrelation is assumed.
- WN** indicates that the prediction error autocorrelation is white noise.

The default value is PERFECT.

- NlagPct* takes a numeric value between 0 and 100 that specifies the percentage of error series count for autocorrelation computations. The default value is 25 (25%).

- NParms* takes a nonnegative integer value that specifies the number of parameters that the external model uses to generate the forecasts. The default value is 0.

CLIMITS.GetForecast Method

```
rc=obj.GetForecast (Which,Result) ;
```

Places the specified forecast series (*Which*) from the CLIMITS object into the specified numeric array (*Result*). Forecast series have the same length as the predicted series that is supplied to the [Compute](#) method via its *Predicted* argument. The GetForecast method returns a negative status code if the [Compute](#) method returned a nonzero value or if it was not yet executed (that is, no results exist to be queried).

Input Arguments

You must specify the following input argument:

Which is a case-insensitive character string that specifies the type of forecast series to return. You can specify one of the following values:

LOWER returns a lower confidence limit series.
STDERR returns a prediction standard error series.
UPPER returns an upper confidence limit series.

Output Arguments

You must specify the following output argument:

Result specifies a numeric array to receive the forecast series. If the array length is longer than the requested series, it is padded with missing values.

OUTNVP Object

The OUTNVP object collects any ad hoc numeric variables that are defined in the user program into tables. The OUTNVP collector object accepts any of the following numeric types: scalar literal, scalar variable, and array variable.

Table 33.5 shows the contents of the OUTNVP object.

Table 33.5 Contents of the OUTNVP Object

Column	Type	Description
NAME	String	Name of the dependent variable
CALL	Numeric	Call count within the BY group
UTAG	Numeric	User-defined numeric tag
VIX	Numeric	Value index (1-based) for the row
VALUE	Numeric	Actual value for variable's row

Table 33.6 summarizes the methods that are associated with the OUTNVP object.

Table 33.6 Methods of the OUTNVP Object

Method	Description
<code>Collect</code>	Collect value for numeric data type
<code>nrows</code>	Get the OUTNVP instance row count

OUTNVP Synopsis

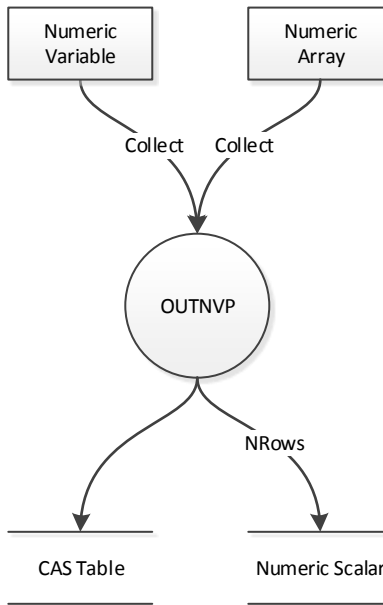
DECLARE OBJECT *obj* (OUTNVP) ;

Method syntax, in order of typical usage:

```
rc=obj.Collect ( Variable,< Utag> ) ;  
nrows=obj.nrows () ;
```

Figure 33.2 outlines the programmatic data flow through the OUTNVP object.

Figure 33.2 OUTNVP Object Data Flow



OUTNVP Methods

OUTNVP.Collect Method

```
rc=obj.Collect ( Variable,< Utag> ) ;
```

Stores a numeric type, either scalar literal or variable or array variable in the OUTNVP table. When the *Variable* parameter is a numeric scalar literal or variable, this method collects a single row into the OUTNVP table. When the *Variable* parameter is a numeric array variable, this method collects a sequence of rows for the span of indices in the array. If the optional parameter *Utag* is specified, its value is included in the `_UTAG_` column of each collected OUTNVP row. The name of the collected variable is also included in the `_NAME_` column of each collected OUTNVP row. Similarly, the value of a counter that counts the number of calls to this method that are made within a BY group is also included in column `_CALL_` of each OUTNVP row. A negative return value indicates that an error occurred while storing results into a table.

Input Arguments

You must specify the following input arguments:

Variable specifies a numeric array to be collected into the OUTNVP table.

You can also specify the following input argument:

Utag takes a numeric value that is included in the _UTAG_ column of the OUTNVP table. The default value is a missing value.

OUTNVP.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the table.

Arguments

There are no arguments associated with this method.

UTLSTAT Object

The UTLSTAT object conveniently computes a number of forecast fit statistics for an ad hoc pair of user-specified actual and predicted time series. The computed forecast fit statistics are automatically stored in a table. For each ad hoc pair of actual and predicted time series that is input into a UTLSTAT collector object, a single row of forecast fit statistics is added to the underlying table. [Table 33.7](#) shows the contents of the UTLSTAT object. For more information about forecast fit statistics, see the section “Statistics of Fit” in Chapter 17, “Forecasting Details” (*SAS Visual Forecasting: Time Series Packages*).

Table 33.7 Contents of the UTLSTAT Object

Column	Type	Description
NAME	String	Variable name of actual series
MODEL	String	Variable name of predicted series
AADJRSQ	Numeric	Amemiya’s adjusted R-square
ADJRSQ	Numeric	Adjusted R-square
AIC	Numeric	Akaike’s information criterion
AICC	Numeric	Finite sample corrected AIC
APC	Numeric	Amemiya’s prediction criterion
DFE	Numeric	Degrees of freedom error
GMAPE	Numeric	Geometric mean absolute percentage error

Table 33.7 *continued*

Column	Type	Description
GMAPES	Numeric	Geometric mean absolute error as a percentage of standard deviation
GMAPPE	Numeric	Geometric mean absolute predictive percentage error
GMRAE	Numeric	Geometric mean relative absolute error
GMASPE	Numeric	Geometric mean absolute symmetric percentage error
IMASE	Numeric	In-sample mean absolute scaled error
MAE	Numeric	Mean absolute error
MAPE	Numeric	Mean absolute percentage error
MAPES	Numeric	Mean absolute error as a percentage of standard deviation
MAPPE	Numeric	Mean absolute predictive percentage error
MASE	Numeric	Mean absolute scaled error
MAXAPES	Numeric	Maximum absolute error as a percentage of standard deviation
MAXERR	Numeric	Maximum error
MAXPE	Numeric	Maximum percentage error
MAXPPE	Numeric	Maximum predictive percentage error
MAXRE	Numeric	Maximum relative error
MAXSPE	Numeric	Maximum symmetric percentage error
MDAPE	Numeric	Median absolute percentage error
MDAPES	Numeric	Median absolute error as a percentage of standard deviation
MDAPPE	Numeric	Median absolute predictive percentage error
MDASPE	Numeric	Median absolute symmetric percentage error
MDRAE	Numeric	Median relative absolute error
ME	Numeric	Mean error
MINAPES	Numeric	Minimum absolute error as a percentage of standard deviation
MINERR	Numeric	Minimum error
MINPE	Numeric	Minimum percentage error
MINPPE	Numeric	Minimum predictive percentage error
MINRE	Numeric	Minimum relative error
MINSPE	Numeric	Minimum symmetric percentage error

Table 33.7 *continued*

Column	Type	Description
MPE	Numeric	Mean percentage error
MPPE	Numeric	Mean predictive percentage error
MRAE	Numeric	Mean relative absolute error
MRE	Numeric	Mean relative error
MSE	Numeric	Mean square error
MSPE	Numeric	Mean symmetric percentage error
N	Numeric	Number of observations that were used. This is the number of observations in the block of observations from the first to the last nonmissing error, which includes any embedded missing errors.
NMISSA	Numeric	Number of missing actual values
NMISSP	Numeric	Number of missing predicted values
NOBS	Numeric	Number of observations
NPARMS	Numeric	Number of parameters
RMSE	Numeric	Root mean square error
RMSSE	Numeric	Root mean square scaled error
RSQUARE	Numeric	R-square
RWRSQ	Numeric	Random walk R-square
SBC	Numeric	Schwarz Bayesian information criterion
SMAPE	Numeric	Symmetric mean absolute percentage error
SSE	Numeric	Sum of square error
SST	Numeric	Corrected total sum of squares
TSS	Numeric	Total sum of squares
UMSE	Numeric	Unbiased mean square error
URMSE	Numeric	Unbiased root mean square error

Table 33.8 summarizes the methods that are associated with the UTLSTAT object.

Table 33.8 Methods of the UTLSTAT Object

Method	Description
<code>Collect</code>	Compute and collect forecast fit statistics for a specified ad hoc pair of actual and predicted time series.
<code>nrows</code>	Get the UTLSTAT instance row count

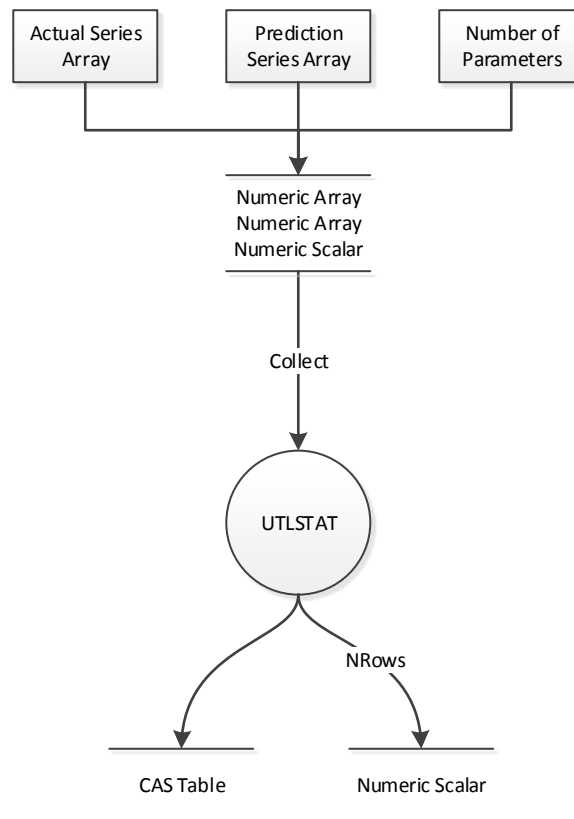
UTLSTAT Synopsis

DECLARE OBJECT *obj* (UTLSTAT) ;

Method syntax, in order of typical usage:

```
rc=obj.Collect (Actual, Predicted, < Nparms > ) ;
nrows=obj.nrows () ;
```

Figure 33.3 outlines the programmatic data flow through the UTLSTAT object.

Figure 33.3 UTLSTAT Object Data Flow

UTLSTAT Methods

UTLSTAT.Collect Method

```
rc=obj.Collect (Actual, Predicted < , Nparms < , Back < , ModelName>>>) ;
```

Computes and collects forecast fit statistics for an ad hoc pair of specified actual time series and predicted time series. Each call collects up to two rows and saves them in the UTLSTAT table; these two rows contain the forecast fit statistics that are listed in Table 33.7. A negative return value indicates that an error occurred either during the computation of the forecast fit statistics or during storage of the results in a table. The values of the forecast fit statistics that are computed and the number of rows that are collected by this method depend on the values of the optional arguments *Nparms* and *Back*.

Input Arguments

You must specify the following input arguments:

- Actual* specifies a numeric array that corresponds to the actual time series.
- Predicted* specifies a numeric array that corresponds to the predicted time series.

You can also specify the following optional input arguments, in the order shown:

- Nparms* takes a numeric value that specifies the number of parameters used by the model that generated the predicted time series. The default value is 0.
- Back* takes a nonnegative integer value that specifies the number of time periods to use for computing out-of-sample statistics.

If you specify the value *B* for *Back*, then the UTLSTAT object computes and writes two sets of statistics to the UTLSTAT table. The first set is referred to as in-sample statistics and corresponds to time periods 1 to $L - B$, where L is the length of the series. The second set is referred to as out-of-sample statistics and corresponds to time periods $L - B + 1$ to L . The variable `_REGION_` in the UTLSTAT table identifies each set. The `_REGION_='FIT'` row contains in-sample statistics, and the `_REGION_='FORECAST'` row contains out-of-sample statistics. If $B = 0$, then only in-sample statistics are computed and collected, and the UTLSTAT table contains only the `_REGION_='FIT'` row. If $B > L$, then the object does not write any statistics to the UTLSTAT table, and if you have specified an OUTLOG table, it writes an appropriate error message to that table.

If you omit this value or specify a missing value, the default value is 0.

- ModelName* takes a string value that specifies the name of the model that generated the predicted time series. The object writes this name to the variable `_MODEL_` in the collected UTLSTAT table. If you omit this value or specify a missing value, the default value is the name that you specify for the *Predicted* array.

UTLSTAT.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the table.

Arguments

There are no arguments associated with this method.

UTL Object

The UTL object provides a method that enables you to measure the execution time of arbitrary portions of the time series analysis program that you submit as input to the TSMODEL procedure. [Table 33.9](#) summarizes this method.

Table 33.9 Method of the UTL Object

Method	Description
GetTime	Get the current time

UTL Synopsis

```
DECLARE OBJECT obj (UTL) ;
```

Method syntax:

```
currentTime=obj.GetTime () ;
```

UTL Methods

UTL.GetTime Method

```
currentTime=obj.GetTime () ;
```

Returns the time (in seconds) relative to the start of the execution of the TSMODEL procedure.

Arguments

There are no arguments associated with this method.

Details

You can use this method to measure the execution time of arbitrary portions of your time series analysis program. If you invoke this method twice and subtract the resulting *currentTime* values, then you can compute a time difference that has a precision of approximately 10 microseconds. The following statements illustrate this functionality:

```

/* Create an instance of the UTL object */
DECLARE OBJECT pkgObj(UTL);

/* Get the current time prior to performing some work */
tStartSecs = pkgObj.GetTime();

/* Perform some arbitrary work */
...

/* Get the current time after performing the work */
tEndSecs = pkgObj.GetTime();

/* Compute the amount of time (in seconds) taken to perform the work */
tDiffSecs = tEndSecs - tStartSecs;

/* Print the time difference in the log */
put tDiffSecs;

```

In particular, the preceding statements store in `tDiffSecs` the measured execution time of an arbitrary amount of work and then print the execution time in the log by passing `tDiffSecs` as the argument to the `PUT` statement. For information about how to log the output of the `PUT` statement, see the `OUTLOG=`, `LOGCONTROL=`, and `PUTTOLOG=` options in the `PROC TSMODEL` statement in Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*).

Examples: UTL Package

Throughout this section, it is assumed that you have already started a CAS session and that the data tables that are used in this section are in `mylib`, a CAS library that you have the necessary permissions to work with. This section assumes that you are familiar with the general workings of the `TSMODEL` procedure; for more information, see Chapter 16, “TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*).

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```

proc options option=(CASHOST CASPORT);
run;

```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mylib cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mylib` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Example 33.1: Collecting Forecast Fit Statistics and Ad Hoc Numeric Variables into Tables

This example demonstrates the capabilities of the UTLSTAT and OUTNVP collector objects from the UTL package. The UTLSTAT collector object is used to compute and collect forecast fit statistics for a specified pair of actual and predicted time series. In contrast, the OUTNVP collector object is used to store ad hoc numeric variables that are found in the user-defined program into tables. The example starts by using a DATA step to load a sample time series data set called `Sashelp.Air` into a table. The TSMODEL procedure is then invoked and the Time Series Model (TSM) object (available in the TSM package) is used to generate a model for the Air series in the `Sashelp.Air` data set. Next, both the actual Air series and its predicted counterpart (generated by the TSM object) are input into the UTLSTAT object to compute and collect numerous forecast fit statistics (see [Table 33.7](#)) into a table. Because this example processes only a single time series (the Air series) and a single BY group, a single row of output is stored in the UTLSTAT table. The example then uses the OUTNVP collector object to store into a table various forecast series that are retrieved from the TSM object. Each forecast series is first queried into a numeric array via the GetForecast method in the TSM object. Each array is then input directly, one by one, into the OUTNVP collector object. Finally, some results are retrieved from the resulting UTLSTAT and OUTNVP tables, sorted, and printed for further inspection.

The following DATA step loads the `Sashelp.Air` data set onto the CAS server. This DATA step assumes that your CAS engine libref is named `mylib`, but you can substitute any appropriately defined CAS engine libref.

```
data mylib.air (replace=yes);
  set Sashelp.Air;
run;
```

The following statements use the TSMODEL procedure to perform time series modeling on a single BY group. Because no ACCUMULATE= option is specified in the ID or VAR statements, its default value of TOTAL is used, which accumulates observations within a time period as a total sum of the nonmissing values.

```

proc tsmmodel data=mylib.air
  outarray = mylib.outarray (replace=yes)
  outscalar = mylib.outscalar (replace=yes)
  outobj=(
    utlstatobj = mylib.utlstat (replace = YES)
    outnvpmodelobj = mylib.outnvpmodel (replace = YES)
  )
  outlog = mylib.outlog (replace=yes)
  lead=12;
id date interval=month start='01jan1949'd end='01dec1960'd;
outarray predict error stderr lcl ucl;
outscalar rc1 rc2 rc3 rc4 rc5 rc6 rc7 rc8;
var air;
require tsm utl;
submit;
  /* Declare the "Time Series Model" (TSM) object and perform fit */
  declare object esm(tsm);
  rc1 = esm.Initialize();
  if rc1 < 0 then do; stop; end;
  rc1 = esm.SetY(air);
  if rc1 < 0 then do; stop; end;
  rc1 = esm.Run();
  if rc1 < 0 then do; stop; end;

  /* Retrieve forecast series computed internally by the TSM object */
  rc2 = esm.GetForecast('predict',predict); /*Predicted series*/
  if rc2 < 0 then do; stop; end;
  rc2 = esm.GetForecast('error',error);      /*Forecast error series*/
  if rc2 < 0 then do; stop; end;
  rc2 = esm.GetForecast('stderr',stderr);    /*Prediction std. errors series*/
  if rc2 < 0 then do; stop; end;
  rc2 = esm.GetForecast('lower',lcl);        /*Lower conf. limits series*/
  if rc2 < 0 then do; stop; end;
  rc2 = esm.GetForecast('upper',ucl);        /*Upper conf. limits series*/
  if rc2 < 0 then do; stop; end;

  /* Collect forecast series computed by the TSM object into a table */
  declare object outnvpmodelobj(outnvp);
  rc3 = outnvpmodelobj.Collect(air,_SERIES_);
  if rc3 < 0 then do; stop; end;
  rc3 = outnvpmodelobj.Collect(predict,_SERIES_);
  if rc3 < 0 then do; stop; end;
  rc3 = outnvpmodelobj.Collect(error,_SERIES_);
  if rc3 < 0 then do; stop; end;
  rc3 = outnvpmodelobj.Collect(stderr,_SERIES_);
  if rc3 < 0 then do; stop; end;
  rc3 = outnvpmodelobj.Collect(lcl,_SERIES_);
  if rc3 < 0 then do; stop; end;
  rc3 = outnvpmodelobj.Collect(ucl,_SERIES_);
  if rc3 < 0 then do; stop; end;

  /* Compute and collect a vast number of forecast fit statistics */
  declare object utlstatobj(utlstat);

```



```

rc4 = utlstatobj.Collect(air, predict);
if rc4 < 0 then do; stop; end;
endsubmit;
run;

```

You can use the PRINT procedure to display a small subset of the 55 different forecast fit statistics that are collected by the UTLSTAT object. PROC PRINT can access CAS tables directly; thus, there is no need to retrieve the UTLSTAT table from CAS and save it in a local data set prior to display.

```

/* Print a few forecast fit statistics for the single BY group */
proc print data=mylib.utlstat;
  var _NAME_ _MODEL_ NOBS RMSE MAPE MAE RSQUARE AIC;
run;

```

Output 33.1.1 shows that a single row of data was collected. This row corresponds to the forecast fit statistics that were collected for a single forecast (that is, one pair of actual and predicted series) within the single BY group that was processed by the TSMODEL procedure call.

Output 33.1.1 Sample of the Forecast Fit Statistics Computed and Collected by the UTLSTAT Object

Obs	_NAME_	_MODEL_	NOBS	RMSE	MAPE	MAE	RSQUARE	AIC
1	AIR	predict	144	10.579085435	3.0845016398	8.0064787209	0.9921692375	679.35714621

You can use the PRINT procedure again to display a small subset of observations from the following six different series that were collected by the OUTNVP object into the mylib.outnvpmodel table:

- The actual series (that is, the Air series).
- The predicted series that was generated by the TSM object.
- The forecast error series that was generated by the TSM object.
- The prediction standard errors series that was generated by the TSM object.
- The lower confidence limits series that was generated by the TSM object.
- The upper confidence limits series that was generated by the TSM object.

You can print in sequence the values of the first three observations in each of these six series, for a total of 18 rows. To accomplish this, you must sort the mylib.outnvpmodel table in a manner that sequentially aligns all rows that correspond to each unique observation index in all collected series (that is, all rows that correspond to the first observation in all six series, followed by all rows that correspond to the second observation in all six series, and so on). You can use the SORT procedure to simultaneously sort a table and store the results in a local data set as follows:

```

/* Sort OUTNVP table by "row index" and "BY group Collect() call count" */
/* (that is, _VIX_ and _CALL_ columns). */
proc sort data=mylib.outnvpmodel out=outnvpmodel;
  by _VIX_ _CALL_;
run;

/* Print the values of the first 18 rows, which correspond to the values */

```

```

/* of the first 3 observations in the six collected series. These 18 rows */
/* correspond to the condition "1 <= _VIX_ <= 3" in the table called */
/* "mylib.OUTNVPMODEL". */
proc print data=outnvpmodel (obs=18);
run;

```

Output 33.1.2 Sample of the Six Ad Hoc Series Collected by the OUTNVP Object

Obs	_VAR_	_CALL_	_UTAG_	_VIX_	_VALUE_
1	AIR	1	0	1	112
2	predict	2	0	1	111.44700541
3	error	3	0	1	0.5529945856
4	stderr	4	0	1	10.691036547
5	lcl	5	0	1	90.492958825
6	ucl	6	0	1	132.401052
7	AIR	1	0	2	118
8	predict	2	0	2	119.94581467
9	error	3	0	2	-1.945814668
10	stderr	4	0	2	10.691036547
11	lcl	5	0	2	98.991768079
12	ucl	6	0	2	140.89986126
13	AIR	1	0	3	132
14	predict	2	0	3	135.17351431
15	error	3	0	3	-3.173514311
16	stderr	4	0	3	10.691036547
17	lcl	5	0	3	114.21946772
18	ucl	6	0	3	156.1275609

The sequence that is displayed in [Output 33.1.2](#) was obtained by using the SORT procedure to sort the data in the OUTNVP table in increasing order of the columns `_VIX_` (series row index) and `_CALL_` (that is, the OUTNVP object's Collect method call count within BY group). For example, rows 1–6 contain the values of the first observation in each of the six collected series, which correspond to the rows in the `mylib.outnvpmodel` table where column `_VIX_` = 1: row 1 is the value of the first observation in the Air series (that is, column `_VAR_` = 'AIR') and rows 2–6 are the values of the first observations in the Predict, Error, StdErr, LCL, and UCL forecast series that were retrieved from the TSM object via its GetForecast method. The same applies for the second observation in each of the six collected series as described by rows 7–12 (that is, rows where column `_VIX_` = 2) and for the third observation in each of the six collected series as described by rows 13–18 (rows where column `_VIX_` = 3).

Example 33.2: Computing Prediction Standard Errors and Confidence Limits for an Ad Hoc External Forecast

This example uses the TSMODEL procedure to compute the prediction standard errors and confidence limits of an ad hoc forecast that includes an actual and predicted time series. The example starts by using a DATA step to create a synthetic data set called ExternalModel. The synthetic data set contains three time series that make up an ad hoc *external model* (that is, a user-defined model) forecast: an actual series called Air, a simulated forecast error series called Error, and a simulated predicted series called Predict. This synthetic, ad hoc external forecast lacks the prediction standard errors and confidence limits, which will be computed by the CLIMITS object. Notice from the DATA step code that the actual time series is simply a copy of the Air series taken from the Sashelp.Air data set. The predicted time series is generated by simply adding a small amount of noise to the actual Air series. The added noise corresponds to random samples taken of the uniform distribution (which ranges from 0.0 to 1.0) multiplied by a factor of 10. Thus, the added noise samples range from 0.0 to 10.0 and have an expected value of 5.0. This expected value is important because it corresponds to the average forecast error of the simulated predicted series, a value that over many samples should approximate the prediction standard errors that will be computed by the CLIMITS object for this synthetic, ad hoc external forecast. Next, the synthetic data set is uploaded to a table and the TSMODEL procedure is invoked. The actual and predicted time series are input to the CLIMITS object's **Compute** method to compute the prediction standard errors and confidence limits of the ad hoc external forecast. The CLIMITS object's **GetForecast** method is then used to retrieve the resulting three forecast series into numeric arrays that are defined in the user program. Finally, the OUTNVP collector object collects a total of six series into a table: the actual Air series, the simulated predicted and forecast error series, and the three forecast series that were computed by the CLIMITS object. This was similarly done in [Example 33.1](#) to store into a table the actual series plus the five forecast series that were retrieved from the TSM object via its own GetForecast method. Finally, some results are retrieved from tables, sorted, and printed for further inspection:

The following DATA step creates a synthetic data set that contains actual and simulated predicted time series:

```
data ExternalModel (replace=yes);
    set Sashelp.Air;                /* The actual series: Sashelp.Air */
    error = floor(10*ranuni(246));  /* Simulated forecast error series */
    predict = air + error;          /* Simulated predicted series */
run;
```

The following DATA step loads the ExternalModel data set onto the CAS server. This DATA step assumes that your CAS engine libref is named mylib, but you can substitute any appropriately defined CAS engine libref.

```
data mylib.ExternalModel;
    set ExternalModel;
run;
```

The following statements use the TSMODEL procedure to compute the prediction standard errors and confidence limits of the ad hoc external forecast that resides in the synthetic data set:

```
proc tsmodel data = mylib.externalmodel
    outarray = mylib.outarray (replace=yes)
    outscalar = mylib.outscalar (replace=yes)
    outlog = mylib.outlog (replace=yes)
    outobj=( outnvpmodelobj = mylib.outnvpmodel (replace = YES) );
```

```

id date interval=month start='01jan1949'd end='01dec1960'd;
var air predict error;
outarrays stderr lcl ucl;
outscalar rc1 rc2 rc3;
require utl;
submit;
    /* Compute the prediction standard errors and confidence limits */
    declare object clim(CLIMITS);
    rc1 = clim.Compute(air,predict,0.05); /* Confidence level: 0.05 */
    if rc1 < 0 then do; stop; end;

    /* Retrieve the forecast series stored internally in the CLIMITS object */
    rc2 = clim.GetForecast('stderr',stderr); /* Prediction sdt. errors series */
    if rc2 < 0 then do; stop; end;
    rc2 = clim.GetForecast('lower',lcl);      /* Lower conf. limits series */
    if rc2 < 0 then do; stop; end;
    rc2 = clim.GetForecast('upper',ucl);      /* Upper conf. limits series */
    if rc2 < 0 then do; stop; end;

    /* Collect the actual, predicted, and forecast error series, in addition */
    /* to the forecast series computed by the CLIMITS object into a table*/
    declare object outnvpmodelobj(outnvp);
    rc3 = outnvpmodelobj.Collect(air,_SERIES_);
    if rc3 < 0 then do; stop; end;
    rc3 = outnvpmodelobj.Collect(predict,_SERIES_);
    if rc3 < 0 then do; stop; end;
    rc3 = outnvpmodelobj.Collect(error,_SERIES_);
    if rc3 < 0 then do; stop; end;
    rc3 = outnvpmodelobj.Collect(stderr,_SERIES_);
    if rc3 < 0 then do; stop; end;
    rc3 = outnvpmodelobj.Collect(lcl,_SERIES_);
    if rc3 < 0 then do; stop; end;
    rc3 = outnvpmodelobj.Collect(ucl,_SERIES_);
    if rc3 < 0 then do; stop; end;
endsubmit;
run;

```

Following what was done in [Example 33.1](#), you can use the PRINT procedure to display a small subset of observations from the following six different series that were collected by the OUTNVP object into the mylib.outnvpmodel table:

- The actual series (Air series) of the ad hoc external forecast.
- The simulated predicted series of the ad hoc external forecast.
- The simulated forecast error series of the ad hoc external forecast.
- The prediction standard errors series that were computed by the CLIMITS object for the ad hoc external forecast.
- The lower confidence limits series that were computed by the CLIMITS object for the ad hoc external forecast.
- The upper confidence limits series that were computed by the CLIMITS object for the ad hoc external forecast.

You can print in sequence the values of the first three observations in each of these six series, for a total of 18 rows. To accomplish this, you must sort the `mylib.outnvpmodel` table in a manner that sequentially aligns all rows that correspond to each unique observation index in all collected series (all rows that correspond to the first observation in all six series, followed by all rows that correspond to the second observation in all six series, and so on). You can use the SORT procedure to simultaneously sort a table and store the results in a local data set as follows:

```
/* Sort OUTNVP table by "row index" and "BY group Collect() call count" */
/* (that is, _VIX_ and _CALL_ columns). */
proc sort data=mylib.outnvpmodel out=outnvpmodel;
    by _VIX_ _CALL_;
run;

/* Print the value of the first 18 rows, which corresponds to the values */
/* of the first 3 observations in the six collected series. These 18 rows */
/* correspond to the condition "1 <= _VIX_ <= 3" in the table called */
/* "mylib.OUTNVPMODEL". */
proc print data=outnvpmodel (obs=18);
run;
```

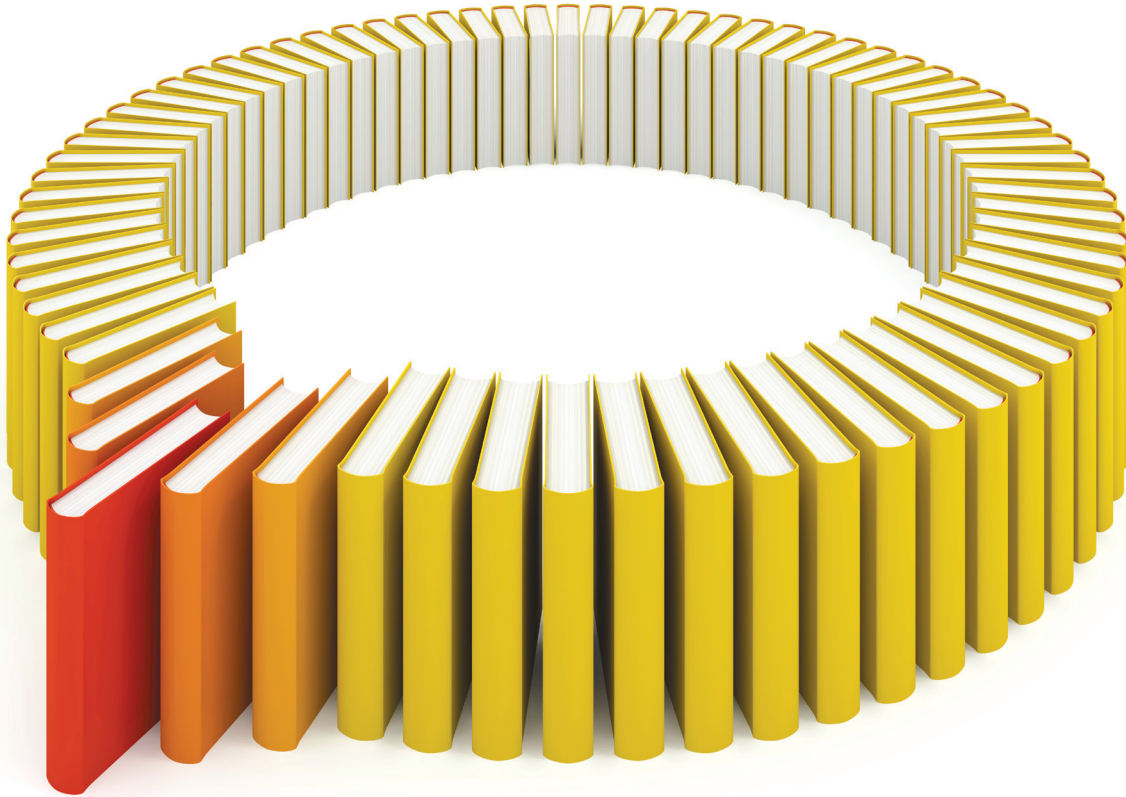
Output 33.2.1 Sample of the Six Ad Hoc Series Collected by the OUTNVP Object

Obs	_VAR_	_CALL_	_UTAG_	_VIX_	_VALUE_
1	AIR	1	0	1	112
2	predict	2	0	1	117
3	error	3	0	1	5
4	stderr	4	0	1	5.0408057116
5	lcl	5	0	1	107.12020235
6	ucl	6	0	1	126.87979765
7	AIR	1	0	2	118
8	predict	2	0	2	124
9	error	3	0	2	6
10	stderr	4	0	2	5.0408057116
11	lcl	5	0	2	114.12020235
12	ucl	6	0	2	133.87979765
13	AIR	1	0	3	132
14	predict	2	0	3	135
15	error	3	0	3	3
16	stderr	4	0	3	5.0408057116
17	lcl	5	0	3	125.12020235
18	ucl	6	0	3	144.87979765

The sequence displayed in [Output 33.2.1](#) was obtained by using the SORT procedure to sort the data in the OUTNVP table in increasing order of the columns `_VIX_` (that is, the series row index) and `_CALL_` (that is, the OUTNVP object's Collect method call count within BY group). For example, rows 1–6 contain the values of the first observation in each of the six collected series, which correspond to the rows in the `mylib.outnvpmodel` table where column `_VIX_` = 1: row 1 is the value of the first observation in the actual Air series (that is, column `_VAR_` = 'AIR'), rows 2–3 are the values of the first observations in the simulated Predict and Error series of the ad hoc external forecast, and rows 4–6 are the values of the first observations in the StdErr, LCL, and UCL forecast series that were computed by the CLIMITS object for the ad hoc external forecast and retrieved via its [GetForecast](#) method. The same applies for the second observation in each of

the six collected series as described by rows 7–12 (that is, rows where column `_VIX_` = 2) and for the third observation in each of the six collected series as described by rows 13–18 (rows where column `_VIX_` = 3).

Notice also in [Output 33.2.1](#) the value of 5.0408057116, which is reported for the first three observations of the prediction standard errors series, as shown by rows where column `_VAR_` = 'stderr'. This reported value should be close to the expected value of the simulated forecast error of the synthetic external forecast (5.0).



Gain Greater Insight into Your SAS[®] Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

 support.sas.com/bookstore
for additional books and resources.


THE POWER TO KNOW[®]

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613