SAS/ETS® 13.2 User's Guide

The MODEL Procedure

This document is an individual chapter from *SAS/ETS® 13.2 User's Guide*.

The correct bibliographic citation for the complete manual is as follows: SAS Institute Inc. 2014. *SAS/ETS® 13.2 User's Guide*. Cary, NC: SAS Institute Inc.

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

August 2014

SAS provides a complete selection of books and electronic products to help customers use SAS® software to its fullest potential. For more information about our offerings, visit **support.sas.com/bookstore** or call 1-800-727-3228.

Chapter 19
The MODEL Procedure

Contents

Overview: MODEL Procedure

The MODEL procedure analyzes models in which the relationships among the variables form a system of one or more nonlinear equations. Primary uses of the MODEL procedure are estimation, simulation, and forecasting of nonlinear simultaneous equation models.

PROC MODEL features include the following:

- SAS programming statements to define simultaneous systems of nonlinear equations
- tools to analyze the structure of the simultaneous equation system
- ARIMA, PDL, and other dynamic modeling capabilities
- tools to specify and estimate the error covariance structure
- tools to estimate and solve ordinary differential equations
- the following methods of parameter estimation:

  - – ordinary least squares (OLS)
  - – two-stage least squares (2SLS)
  - – seemingly unrelated regression (SUR) and iterative SUR (ITSUR)
  - – three-stage least squares (3SLS) and iterative 3SLS (IT3SLS)
  - – generalized method of moments (GMM)
  - – simulated method of moments (SMM)
  - – full information maximum likelihood (FIML)
  - – general log-likelihood maximization
- simulation and forecasting capabilities
- Monte Carlo simulation
- goal-seeking solutions

A system of equations can be nonlinear in the parameters, nonlinear in the observed variables, or nonlinear in both the parameters and the variables. *Nonlinear* in the parameters means that the mathematical relationship between the variables and parameters is not required to have a linear form. (A linear model is a special case of a nonlinear model.) A general nonlinear system of equations can be written as

$$\begin{aligned} q_1(y_{1,t}, y_{2,t}, \ldots, y_{g,t}, x_{1,t}, x_{2,t}, \ldots, x_{m,t}, \theta_1, \theta_2, \ldots, \theta_p) &= \epsilon_{1,t} \\ q_2(y_{1,t}, y_{2,t}, \ldots, y_{g,t}, x_{1,t}, x_{2,t}, \ldots, x_{m,t}, \theta_1, \theta_2, \ldots, \theta_p) &= \epsilon_{2,t} \\ &\vdots \\ q_g(y_{1,t}, y_{2,t}, \ldots, y_{g,t}, x_{1,t}, x_{2,t}, \ldots, x_{m,t}, \theta_1, \theta_2, \ldots, \theta_p) &= \epsilon_{g,t} \end{aligned}$$

where $y_{i,t}$ is an endogenous variable, $x_{i,t}$ is an exogenous variable, θ_i is a parameter, and ϵ_i is the unknown error. The subscript t represents time or some index to the data.

In econometrics literature, the observed variables are either *endogenous* (dependent) variables or *exogenous* (independent) variables. This system can be written more succinctly in vector form as

$$\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) = \boldsymbol{\epsilon}_t$$

This system of equations is in *general form* because the error term is by itself on one side of the equality. Systems can also be written in *normalized form* by placing the endogenous variable on one side of the equality, with each equation defining a predicted value for a unique endogenous variable. A normalized form equation system can be written in vector notation as

$$\mathbf{y}_t = \mathbf{f}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) + \boldsymbol{\epsilon}_t.$$

PROC MODEL handles equations written in both forms.

Econometric models often explain the current values of the endogenous variables as functions of past values of exogenous and endogenous variables. These past values are referred to as *lagged* values, and the variable x_{t-i} is called lag i of the variable x_t. Using lagged variables, you can create a *dynamic*, or time-dependent, model. In the preceding model systems, the lagged exogenous and endogenous variables are included as part of the exogenous variables.

If the data are time series, so that t indexes time (see Chapter 3, "Working with Time Series Data," for more information on time series), it is possible that $\boldsymbol{\epsilon}_t$ depends on $\boldsymbol{\epsilon}_{t-i}$ or, more generally, the $\boldsymbol{\epsilon}_t$'s are not identically and independently distributed. If the errors of a model system are autocorrelated, the standard error of the estimates of the parameters of the system will be inflated.

Sometimes the $\boldsymbol{\epsilon}_i$'s are not identically distributed because the variance of $\boldsymbol{\epsilon}$ is not constant. This is known as *heteroscedasticity*. Heteroscedasticity in an estimated model can also inflate the standard error of the estimates of the parameters. Using a weighted estimation can sometimes eliminate this problem. Alternately, a variance model such as GARCH or EGARCH can be estimated to correct for heteroscedasticity. If the proper weighting scheme and the form of the error model is difficult to determine, generalized methods of moments (GMM) estimation can be used to determine parameter estimates with very few assumptions about the form of the error process.

Other problems can also arise when estimating systems of equations. Consider the following system of equations, which is nonlinear in its parameters and cannot be estimated with linear regression:

$$\begin{aligned} y_{1,t} &= \theta_1 + (\theta_2 + \theta_3\theta_4^t)^{-1} + \theta_5 y_{2,t} + \epsilon_{1,t} \\ y_{2,t} &= \theta_6 + (\theta_7 + \theta_8\theta_9^t)^{-1} + \theta_{10} y_{1,t} + \epsilon_{2,t} \end{aligned}$$

This system of equations represents a rudimentary predator-prey process with y_1 as the prey and y_2 as the predator (the second term in both equations is a logistics curve). The two equations must be estimated simultaneously because of the cross-dependency of y's. This cross-dependency makes ϵ_1 and ϵ_2 violate the assumption of independence. Nonlinear ordinary least squares estimation of these equations produce biased and inconsistent parameter estimates. This is called *simultaneous equation bias*.

One method to remove simultaneous equation bias, in the linear case, is to replace the endogenous variables on the right-hand side of the equations with predicted values that are uncorrelated with the error terms. These predicted values can be obtained through a preliminary, or "first-stage," *instrumental variable regression*. *Instrumental variables*, which are uncorrelated with the error term, are used as regressors to model the predicted values. The parameter estimates are obtained by a second regression by using the predicted values of the regressors. This process is called *two-stage least squares*.

In the nonlinear case, nonlinear ordinary least squares estimation is performed iteratively by using a linearization of the model with respect to the parameters. The instrumental solution to simultaneous equation bias in the nonlinear case is the same as the linear case, except the linearization of the model with respect to the parameters is predicted by the instrumental regression. Nonlinear two-stage least squares is one of several instrumental variables methods available in the MODEL procedure to handle simultaneous equation bias.

When you have a system of several regression equations, the random errors of the equations can be correlated. In this case, the large-sample efficiency of the estimation can be improved by using a joint generalized least squares method that takes the cross-equation correlations into account. If the equations are not simultaneous (no dependent regressors), then *seemingly unrelated regression* (SUR) can be used. The SUR method requires an estimate of the cross-equation error covariance matrix, Σ. The usual approach is to first fit the equations by using OLS, compute an estimate $\hat{\Sigma}$ from the OLS residuals, and then perform the SUR estimation based on $\hat{\Sigma}$. The MODEL procedure estimates Σ by default, or you can supply your own estimate of Σ.

If the equation system is simultaneous, you can combine the 2SLS and SUR methods to take into account both simultaneous equation bias and cross-equation correlation of the errors. This is called *three-stage least squares* or 3SLS.

A different approach to the simultaneous equation bias problem is the full information maximum likelihood (FIML) estimation method. FIML does not require instrumental variables, but it assumes that the equation

errors have a multivariate normal distribution. 2SLS and 3SLS estimation do not assume a particular distribution for the errors.

Other nonnormal error distribution models can be estimated as well. The centered *t* distribution with estimated degrees of freedom and nonconstant variance is an additional built-in likelihood function. If the distribution of the equation errors is not normal or *t* but known, then the log likelihood can be specified by using the ERRORMODEL statement.

Once a nonlinear model has been estimated, it can be used to obtain forecasts. If the model is linear in the variables you want to forecast, a simple linear solve can generate the forecasts. If the system is nonlinear, an iterative procedure must be used. The preceding example system is linear in its endogenous variables. The MODEL procedure's SOLVE statement is used to forecast nonlinear models.

One of the main purposes of creating models is to obtain an understanding of the relationship among the variables. There are usually only a few variables in a model you can control (for example, the amount of money spent on advertising). Often you want to determine how to change the variables under your control to obtain some target goal. This process is called *goal seeking*. PROC MODEL allows you to solve for any subset of the variables in a system of equations given values for the remaining variables.

The nonlinearity of a model creates two problems with the forecasts: the forecast errors are not normally distributed with zero mean, and no formula exists to calculate the forecast confidence intervals. PROC MODEL provides Monte Carlo techniques, which, when used with the covariance of the parameters and error covariance matrix, can produce approximate error bounds on the forecasts. The following distributions on the errors are supported for multivariate Monte Carlo simulation:

- Cauchy
- chi-squared
- empirical
- *F*
- Poisson
- *t*
- uniform

A transformation technique is used to create a covariance matrix for generating the correct innovations in a Monte Carlo simulation.

Getting Started: MODEL Procedure

This section introduces the MODEL procedure and shows how to use PROC MODEL for several kinds of nonlinear regression analysis and nonlinear systems simulation problems.

Nonlinear Regression Analysis

One of the most important uses of PROC MODEL is to estimate unknown parameters in a nonlinear model. A simple nonlinear model has the form:

$$y = f(\mathbf{x}, \boldsymbol{\theta}) + \epsilon$$

where $\mathbf{x}$ is a vector of exogenous variables. To estimate unknown parameters by using PROC MODEL, do the following:

1. Use the DATA= option in a PROC MODEL statement to specify the input SAS data set that contains $\mathbf{y}$ and $\mathbf{x}$, the observed values of the variables.
2. Write the equation for the model by using SAS programming statements, including all parameters and arithmetic operators but leaving off the unobserved error component, ϵ.
3. Use a FIT statement to fit the model equation to the input data to determine the unknown parameters, $\boldsymbol{\theta}$.

An Example

The SASHELP library contains the data set CITIMON, which contains the variable LHUR, the monthly unemployment figures, and the variable IP, the monthly industrial production index. You suspect that the unemployment rates are inversely proportional to the industrial production index. Assume that these variables are related by the following nonlinear equation:

$$lhur = \frac{1}{a \cdot \mathrm{ip} + b} + c + \epsilon$$

In this equation a, b, and c are unknown coefficients and ϵ is an unobserved random error.

The following statements illustrate how to use PROC MODEL to estimate values for a, b, and c from the data in SASHELP.CITIMON.

```
proc model data=sashelp.citimon;
   lhur = 1/(a * ip + b) + c;
   fit lhur;
run;
```

Notice that the model equation is written as a SAS assignment statement. The variable LHUR is assumed to be the dependent variable because it is named in the FIT statement and is on the left-hand side of the assignment.

PROC MODEL determines that LHUR and IP are observed variables because they are in the input data set. A, B, and C are treated as unknown parameters to be estimated from the data because they are not in the input data set. If the data set contained a variable named A, B, or C, you would need to explicitly declare the parameters with a PARMS statement.

In response to the FIT statement, PROC MODEL estimates values for A, B, and C by using nonlinear least squares and prints the results. The first part of the output is a "Model Summary" table, shown in Figure 19.1.

Figure 19.1 Model Summary Report

The MODEL Procedure

Model Summary	
Model Variables	1
Parameters	3
Equations	1
Number of Statements	1

Model Variables	LHUR
Parameters	a b c
Equations	LHUR

This table details the size of the model, including the number of programming statements that define the model, and lists the dependent variables (LHUR in this case), the unknown parameters (A, B, and C), and the model equations. In this case the equation is named for the dependent variable, LHUR.

PROC MODEL then prints a summary of the estimation problem, as shown in Figure 19.2.

Figure 19.2 Estimation Problem Report

The Equation to Estimate is
LHUR = F(a, b, c(1))

The notation used in the summary of the estimation problem indicates that LHUR is a function of A, B, and C, which are to be estimated by fitting the function to the data. If the partial derivative of the equation with respect to a parameter is a simple variable or constant, the derivative is shown in parentheses after the parameter name. In this case, the derivative with respect to the intercept C is 1. The derivatives with respect to A and B are complex expressions and so are not shown.

Next, PROC MODEL prints an estimation summary as shown in Figure 19.3.

Figure 19.3 Estimation Summary Report

The MODEL Procedure
OLS Estimation Summary

Data Set Options
DATA= SASHELP.CITIMON

Minimization Summary	
Parameters Estimated	3
Method	Gauss
Iterations	10

Figure 19.3 *continued*

Final Convergence Criteria	
R	0.000737
PPC(b)	0.003943
RPC(b)	0.00968
Object	4.784E-6
Trace(S)	0.533325
Objective Value	0.522214

Observations Processed	
Read	145
Solved	145
Used	144
Missing	1

The estimation summary provides information on the iterative process used to compute the estimates. The heading "OLS Estimation Summary" indicates that the nonlinear ordinary least squares (OLS) estimation method is used. This table indicates that all three parameters were estimated successfully by using 144 nonmissing observations from the data set SASHELP.CITIMON. Calculating the estimates required 10 iterations of the GAUSS method. Various measures of how well the iterative process converged are also shown. For example, the "RPC(B)" value 0.00968 means that on the final iteration the largest relative change in any estimate was for parameter B, which changed by 0.968 percent. See the section "Convergence Criteria" on page 1151 for details.

PROC MODEL then prints the estimation results. The first part of this table is the summary of residual errors, shown in Figure 19.4.

Figure 19.4 Summary of Residual Errors Report

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors								
Equation	**DF Model**	**DF Error**	**SSE**	**MSE**	**Root MSE**	**R-Square**	**Adj R-Sq**	**Label**
LHUR	3	141	75.1989	0.5333	0.7303	0.7472	0.7436	UNEMPLOYMENT RATE: ALL WORKERS, 16 YEARS

This table lists the sum of squared errors (SSE), the mean squared error (MSE), the root mean squared error (root MSE), and the R^2 and adjusted R^2 statistics. The R^2 value of 0.7472 means that the estimated model explains approximately 75 percent more of the variability in LHUR than a mean model explains.

Following the summary of residual errors is the parameter estimates table, shown in Figure 19.5.

Figure 19.5 Parameter Estimates

Nonlinear OLS Parameter Estimates				
Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
a	0.009046	0.00343	2.63	0.0094
b	-0.57059	0.2617	-2.18	0.0309
c	3.337151	0.7297	4.57	<.0001

Because the model is nonlinear, the standard error of the estimate, the *t* value, and its significance level are only approximate. These values are computed using asymptotic formulas that are correct for large sample sizes but only approximately correct for smaller samples. Thus, you should use caution in interpreting these statistics for nonlinear models, especially for small sample sizes. For linear models, these results are exact and are the same as standard linear regression.

The last part of the output produced by the FIT statement is shown in Figure 19.6.

Figure 19.6 System Summary Statistics

Number of Observations		Statistics for System	
Used	144	Objective	0.5222
Missing	1	Objective*N	75.1989

This table lists the objective value for the estimation of the nonlinear system. Since there is only a single equation in this case, the objective value is the same as the residual MSE for LHUR except that the objective value does not include a degrees-of-freedom correction. This can be seen in the fact that "Objective*N" equals the residual SSE, 75.1989. N is 144, the number of observations used.

Convergence and Starting Values

Computing parameter estimates for nonlinear equations requires an iterative process. Starting with an initial guess for the parameter values, PROC MODEL tries different parameter values until the objective function of the estimation method is minimized. (The objective function of the estimation method is sometimes called the *fitting function*.) This process does not always succeed, and whether it does succeed depends greatly on the starting values used. By default, PROC MODEL uses the starting value 0.0001 for all parameters.

Consequently, in order to use PROC MODEL to achieve convergence of parameter estimates, you need to know two things: how to recognize convergence failure by interpreting diagnostic output, and how to specify reasonable starting values. The MODEL procedure includes alternate iterative techniques and grid search capabilities to aid in finding estimates. See the section "Troubleshooting Convergence Problems" on page 1153 for more details.

Nonlinear Systems Regression

If a model has more than one endogenous variable, several facts need to be considered in the choice of an estimation method. If the model has endogenous regressors, then an instrumental variables method such as 2SLS or 3SLS can be used to avoid simultaneous equation bias. Instrumental variables must be provided

to use these methods. A discussion of possible choices for instrumental variables is provided in the section "Choice of Instruments" on page 1202 in this chapter.

The following is an example of the use of 2SLS and the INSTRUMENTS statement:

```
proc model data=test2;
   exogenous x1 x2;
   parms a1 a2 b2 2.5 c2 55 d1;

   y1 = a1 * y2 + b2 * x1 * x1 + d1;
   y2 = a2 * y1 + b2 * x2 * x2 + c2 / x2 + d1;

   fit y1 y2 / 2sls;
   instruments b2 c2 _exog_;
run;
```

The estimation method selected is added after the slash (/) on the FIT statement. The INSTRUMENTS statement follows the FIT statement and in this case selects all the exogenous variables as instruments with the _EXOG_ keyword. The parameters B2 and C2 in the instruments list request that the derivatives with respect to B2 and C2 be additional instruments.

Full information maximum likelihood (FIML) can also be used to avoid simultaneous equation bias. FIML is computationally more expensive than an instrumental variables method and assumes that the errors are normally distributed. On the other hand, FIML does not require the specification of instruments. FIML is selected with the FIML option on the FIT statement.

The preceding example is estimated with FIML by using the following statements:

```
proc model data=test2;
   exogenous x1 x2;
   parms a1 a2 b2 2.5 c2 55 d1;

   y1 = a1 * y2 + b2 * x1 * x1 + d1;
   y2 = a2 * y1 + b2 * x2 * x2 + c2 / x2 + d1;

   fit y1 y2 / fiml;
run;
```

General Form Models

The single equation example shown in the preceding section was written in normalized form and specified as an assignment of the regression function to the dependent variable LHUR. However, sometimes it is impossible or inconvenient to write a nonlinear model in normalized form.

To write a general form equation, give the equation a name with the prefix "EQ.". This EQ.-prefixed variable represents the equation error. Write the equation as an assignment to this variable.

For example, suppose you have the following nonlinear model that relates the variables x and y :

$$\epsilon = a + b \ln(cy + dx)$$

Naming this equation 'one', you can fit this model with the following statements:

```
proc model data=xydata;
   eq.one = a + b * log( c * y + d * x );
   fit one;
run;
```

The use of the EQ. prefix tells PROC MODEL that the variable is an error term and that it should not expect actual values for the variable ONE in the input data set.

Supply and Demand Models

General form specifications are often useful when you have several equations for the same dependent variable. This is common in supply and demand models, where both the supply equation and the demand equation are written as predictions for quantity as functions of price.

For example, consider the following supply and demand system:

$$\begin{aligned} \text{(supply)} \quad & \text{quantity} = \alpha_1 + \alpha_2 \text{ price} + \epsilon_1 \\ \text{(demand)} \quad & \text{quantity} = \beta_1 + \beta_2 \text{ price} + \beta_3 \text{ income} + \epsilon_2 \end{aligned}$$

Assume the *quantity* of interest is the amount of energy consumed in the U.S., the *price* is the price of gasoline, and the *income* variable is the consumer debt. When the market is at equilibrium, these equations determine the market price and the equilibrium quantity. These equations are written in general form as

$$\epsilon_1 = quantity - (\alpha_1 + \alpha_2\ price)$$

$$\epsilon_2 = quantity - (\beta_1 + \beta_2\ price + \beta_3\ income)$$

Note that the endogenous variables *quantity* and *price* depend on two error terms so that OLS should not be used. The following example uses three-stage least squares estimation.

Data for this model is obtained from the SASHELP.CITIMON data set.

```
title1 'Supply-Demand Model using General-form Equations';
proc model data=sashelp.citimon;
   endogenous eegp eec;
   exogenous exvus cciutc;
   parameters a1 a2 b1 b2 b3 ;
   label eegp   = 'Gasoline Retail Price'
         eec    = 'Energy Consumption'
         cciutc = 'Consumer Debt';

   /* -------- Supply equation ------------- */
   eq.supply = eec - (a1 + a2 * eegp );

   /* -------- Demand equation ------------- */
   eq.demand = eec - (b1 + b2 * eegp + b3 * cciutc);

   /* -------- Instrumental variables -------*/
   lageegp = lag(eegp); lag2eegp=lag2(eegp);
```

```
    /* -------- Estimate parameters --------- */
    fit supply demand / n3sls fsrsq;
    instruments _EXOG_ lageegp lag2eegp;
run;
```

The FIT statement specifies the two equations to estimate and the method of estimation, N3SLS. Note that '3SLS' is an alias for N3SLS. The option FSRSQ is selected to get a report of the first stage R^2 to determine the acceptability of the selected instruments.

Since three-stage least squares is an instrumental variables method, instruments are specified with the INSTRUMENTS statement. The instruments selected are all the exogenous variables, selected with the _EXOG_ option, and two lags of the variable EEGP: LAGEEGP and LAG2EEGP.

The data set CITIMON has four observations that generate missing values because values for EEGP, EEC, or CCIUTC are missing. This is revealed in the "Observations Processed" output shown in Figure 19.7. Missing values are also generated when the equations cannot be computed for a given observation. Missing observations are not used in the estimation.

Figure 19.7 Supply-Demand Observations Processed

Supply-Demand Model using General-form Equations

The MODEL Procedure
3SLS Estimation Summary

Observations Processed	
Read	145
Solved	143
First	3
Last	145
Used	139
Missing	4
Lagged	2

The lags used to create the instruments also reduce the number of observations used. In this case, the first two observations were used to fill the lags of EEGP.

The data set has a total of 145 observations, of which four generated missing values and two were used to fill lags, which left 139 observations for the estimation. In the estimation summary, in Figure 19.8, the total degrees of freedom for the model and error is 139.

Figure 19.8 Supply-Demand Parameter Estimates

Supply-Demand Model using General-form Equations

The MODEL Procedure

Nonlinear 3SLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
supply	2	137	43.2677	0.3158	0.5620		
demand	3	136	39.5791	0.2910	0.5395		

Figure 19.8 *continued*

Nonlinear 3SLS Parameter Estimates					
Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	1st Stage R-Square
a1	7.30952	0.3799	19.24	<.0001	1.0000
a2	-0.00853	0.00328	-2.60	0.0103	0.9617
b1	6.82196	0.3788	18.01	<.0001	1.0000
b2	-0.00614	0.00303	-2.02	0.0450	0.9617
b3	9E-7	3.165E-7	2.84	0.0051	1.0000

One disadvantage of specifying equations in general form is that there are no actual values associated with the equation, so the R^2 statistic cannot be computed.

Solving Simultaneous Nonlinear Equation Systems

You can use a SOLVE statement to solve the nonlinear equation system for some variables when the values of other variables are given.

Consider the supply and demand model shown in the preceding example. The following statement computes equilibrium price (EEGP) and quantity (EEC) values for given observed cost (CCIUTC) values and stores them in the output data set EQUILIB.

```
title1 'Supply-Demand Model using General-form Equations';
proc model data=sashelp.citimon(where=(eec ne .));
   endogenous eegp eec;
   exogenous exvus cciutc;
   parameters a1 a2 a3 b1 b2 ;
   label eegp   = 'Gasoline Retail Price'
         eec    = 'Energy Consumption'
         cciutc = 'Consumer Debt';

   /* -------- Supply equation ------------- */
   eq.supply = eec - (a1 + a2 * eegp + a3 * cciutc);

   /* -------- Demand equation ------------- */
   eq.demand = eec - (b1 + b2 * eegp );

   /* -------- Instrumental variables -------*/
   lageegp = lag(eegp); lag2eegp=lag2(eegp);

   /* -------- Estimate parameters --------- */
   instruments _EXOG_ lageegp lag2eegp;
   fit supply demand / n3sls ;
   solve eegp eec / out=equilib;
run;
```

As a second example, suppose you want to compute points of intersection between the square root function and hyperbolas of the form $a + b/x$. That is, you want to solve the system:

$$\text{(square root)} \quad y = \sqrt{x}$$

$$\text{(hyperbola)} \quad y = a + \frac{b}{x}$$

The following statements read parameters for several hyperbolas in the input data set TEST and solve the nonlinear equations. The SOLVEPRINT option in the SOLVE statement prints the solution values. The ID statement is used to include the values of A and B in the output of the SOLVEPRINT option.

```
title1 'Solving a Simultaneous System';
data test;
   input a b @@;
datalines;
  0 1   1 1   1 2
;

proc model data=test;
   eq.sqrt      = sqrt(x) - y;
   eq.hyperbola = a + b / x - y;
   solve x y / solveprint;
   id a b;
run;
```

The printed output produced by this example consists of a model summary report, a listing of the solution values for each observation, and a solution summary report. The model summary for this example is shown in Figure 19.9.

Figure 19.9 Model Summary Report

Solving a Simultaneous System

The MODEL Procedure

Model Summary	
Model Variables	2
ID Variables	2
Equations	2
Number of Statements	2

Model Variables	x y
Equations	sqrt hyperbola

The output produced by the SOLVEPRINT option is shown in Figure 19.10.

Figure 19.10 Solution Values for Each Observation

Solving a Simultaneous System

The MODEL Procedure
Simultaneous Simulation

Observation	1	**a**	0	**b**	1.0000	**eq.hyperbola**	0.000000
		Iterations	17	**CC**	0.000000		

Solution Values	
x	**y**
1.000000	1.000000

Observation	2	**a**	1.0000	**b**	1.0000	**eq.hyperbola**	0.000000
		Iterations	5	**CC**	0.000000		

Solution Values	
x	**y**
2.147899	1.465571

Observation	3	**a**	1.0000	**b**	2.0000	**eq.hyperbola**	0.000000
		Iterations	4	**CC**	0.000000		

Solution Values	
x	**y**
2.875130	1.695621

For each observation, a heading line is printed that lists the values of the ID variables for the observation and information about the iterative process used to compute the solution. The number of iterations required, and the convergence measure (labeled CC) are printed. This convergence measure indicates the maximum error by which solution values fail to satisfy the equations. When this error is small enough (as determined by the CONVERGE= option), the iterations terminate. The equation with the largest error is indicated. For example, for observation 3 the HYPERBOLA equation has an error of 4.42×10^{-13} while the error of the SQRT equation is even smaller. Following the heading line for the observation, the solution values are printed.

The last part of the SOLVE statement output is the solution summary report shown in Figure 19.11. This report summarizes the solution method used (Newton's method by default), the iteration history, and the observations processed.

Figure 19.11 Solution Summary Report

Solving a Simultaneous System

The MODEL Procedure
Simultaneous Simulation

Data Set Options	
DATA=	TEST

Solution Summary	
Variables Solved	2
Implicit Equations	2
Solution Method	NEWTON
CONVERGE=	1E-8
Maximum CC	9.176E-9
Maximum Iterations	17
Total Iterations	26
Average Iterations	8.666667

Observations Processed	
Read	3
Solved	3

Variables Solved For	x y
Equations Solved	sqrt hyperbola

Monte Carlo Simulation

The RANDOM= option is used to request Monte Carlo (or stochastic) simulation to generate confidence intervals for a forecast. The confidence intervals are implied by the model's relationship to implicit random error term ϵ and the parameters.

The Monte Carlo simulation generates a random set of additive error values, one for each observation and each equation, and computes one set of perturbations of the parameters. These new parameters, along with the additive error terms, are then used to compute a new forecast that satisfies this new simultaneous system. Then a new set of additive error values and parameter perturbations is computed, and the process is repeated the requested number of times.

Consider the following exchange rate model for the U.S. dollar with the German mark and the Japanese yen:

$$rate\_jp = a_1 + b_1 im\_jp + c_1 di\_jp;$$

$$rate\_wg = a_2 + b_2 im\_wg + c_1 di\_wg;$$

where $rate\_jp$ and $rate\_wg$ are the exchange rate of the Japanese yen and the German mark versus the U.S. dollar, respectively; *im_jp* and *im_wg* are the imports from Japan and Germany in 1984 dollars, respectively; and *di_jp* and *di_wg* are the differences in inflation rate of Japan and the U.S., and Germany and the U.S., respectively. The Monte Carlo capabilities of the MODEL procedure are used to generate error bounds on a forecast by using this model.

```
proc model data=exchange;
   endo im_jp im_wg;
   exo di_jp di_wg;
   parms a1 a2 b1 b2 c1 c2;
   label rate_jp = 'Exchange Rate of Yen/$'
         rate_wg = 'Exchange Rate of Gm/$'
         im_jp = 'Imports to US from Japan in 1984 $'
         im_wg = 'Imports to US from WG in 1984 $'
         di_jp = 'Difference in Inflation Rates US-JP'
         di_wg = 'Difference in Inflation Rates US-WG';

   rate_jp = a1 + b1*im_jp + c1*di_jp;
   rate_wg = a2 + b2*im_wg + c2*di_wg;

            /* Fit the EXCHANGE data */
   fit rate_jp rate_wg / sur outest=xch_est outcov outs=s;

            /* Solve using the WHATIF data set */
   solve rate_jp rate_wg / data=whatif estdata=xch_est sdata=s
          random=100 seed=123 out=monte forecast;
   id yr;
   range yr=1986;
run;
```

Data for the EXCHANGE data set was obtained from the Department of Commerce and the yearly "Economic Report of the President."

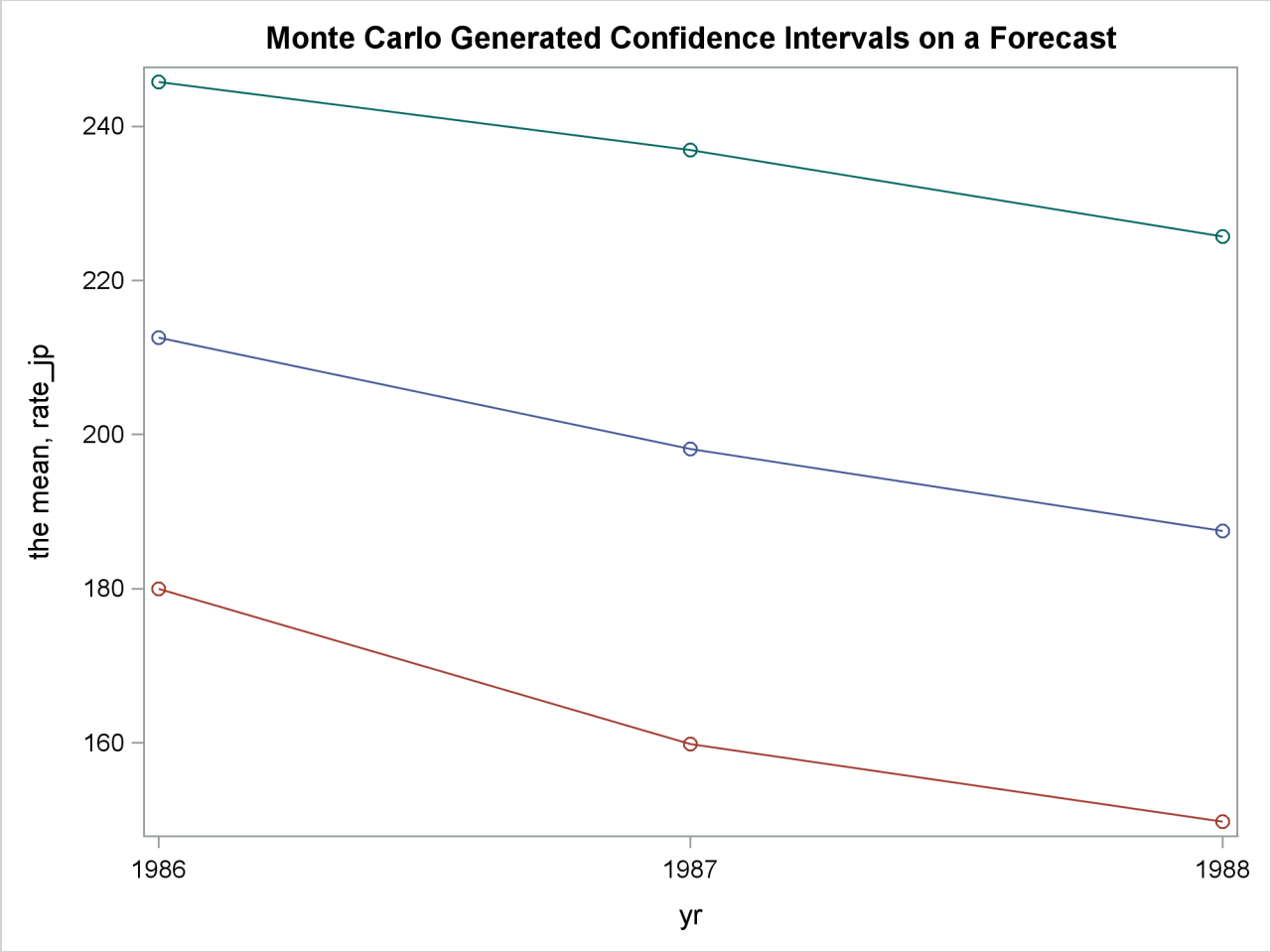
First, the parameters are estimated using SUR selected by the SUR option in the FIT statement. The OUTEST= option is used to create the XCH_EST data set which contains the estimates of the parameters. The OUTCOV option adds the covariance matrix of the parameters to the XCH_EST data set. The OUTS= option is used to save the covariance of the equation error in the data set S.

Next, Monte Carlo simulation is requested by using the RANDOM= option in the SOLVE statement. The data set WHATIF is used to drive the forecasts. The ESTDATA= option reads in the XCH_EST data set which contains the parameter estimates and covariance matrix. Because the parameter covariance matrix is included, perturbations of the parameters are performed. The SDATA= option causes the Monte Carlo simulation to use the equation error covariance in the S data set to perturb the equation errors. The SEED= option selects the number 123 as a seed value for the random number generator. The output of the Monte Carlo simulation is written to the data set MONTE selected by the OUT= option.

To generate a confidence interval plot for the forecast, use PROC UNIVARIATE to generate percentile bounds and use PROC SGPLOT to plot the graph. The following SAS statements produce the graph in Figure 19.12.

```
proc sort data=monte;
   by yr;
run;

proc univariate data=monte noprint;
   by yr;
   var rate_jp rate_wg;
   output out=bounds mean=mean p5=p5 p95=p95;
run;

title "Monte Carlo Generated Confidence Intervals on a Forecast";
proc sgplot data=bounds noautolegend;
  series x=yr y=mean / markers;
  series x=yr y=p5 / markers;
  series x=yr y=p95 / markers;
run;
```

Figure 19.12 Monte Carlo Confidence Interval Plot



Syntax: MODEL Procedure

The following statements can be used with the MODEL procedure:

PROC MODEL *options* **;**
ABORT ;
ARRAY *arrayname variable-list . . .* **;**
ATTRIB *variable-list1 attribute-list1 < variable-list2 attribute-list2 . . . >* **;**
BOUNDS *bound1 < , bound2 . . . >* **;**
BY *variable-list* **;**
CALL *name* **;**
CALL *name(expression1 < , expression2 . . . >)* **;**
CONTROL *variable < value > . . .* **;**
DELETE ;

DO ;
DO *variable = expression < TO expression > < BY expression >*
< , expression TO expression < BY expression > . . . > < WHILE expression >
< UNTIL expression > ;
END ;
DROP *variable . . .* ;
ENDOGENOUS *variable < initial-values > . . .* ;
ERRORMODEL *equation-name ~ distribution < CDF=(CDF(options)) >* ;
ESTIMATE *item1 < , item2 . . . > < ,/ options >* ;
EXOGENOUS *variable < initial values > . . .* ;
FIT *equations < PARMS=(parameter values . . .) > < START=(parameter values . . .) >*
< DROP=(parameters) > < / options > ;
FORMAT *variable-list < format > < DEFAULT= default-format >* ;
GOTO *statement-label* ;
ID *variable-list* ;
IF *expression* ;
IF *expression THEN programming-statement1 ; < ELSE programming-statement2 >* ;
variable *= expression* ;
variable *+ expression* ;
INCLUDE *model-file . . .* ;
INSTRUMENTS *< instruments > < _EXOG_ > < EXCLUDE=(parameters) > < / options >* ;
KEEP *variable . . .* ;
LABEL *variable ='label' . . .* ;
LENGTH *variable-list < $> length . . . < DEFAULT=length >* ;
LINK *statement-label* ;
MOMENT *variable-list = moment-specification . . .* ;
OUTVARS *variable . . .* ;
PARAMETERS *variable1 < value1 > < variable2 < value2 . . . > >* ;
PUT *print-item . . . < @ > < @@ >* ;
RANGE *variable < = first > < TO last >* ;
RENAME *old-name1 = new-name1 < . . . old-name2 = new-name2 >* ;
RESET *options* ;
RESTRICT *restriction1 < , restriction2 . . . >* ;
RETAIN *variable-list1 value1 < variable-list2 value2 . . . >* ;
RETURN ;
SOLVE *variable-list < SATISFY=(equations) > < / options >* ;
SUBSTR *(variable, index, length) = expression* ;
SELECT *< (expression) >* ;
OTHERWISE *programming-statement* ;
STOP ;
TEST *< "name" > test1 < , test2 . . . > < ,/ options >* ;
VAR *variable < initial-values > . . .* ;
WEIGHT *variable* ;
WHEN *(expression) programming-statement* ;

Functional Summary

The statements and options in the MODEL procedure are summarized in the following table.

Description	Statement	Option
Data Set Options		
Specifies the input data set for the variables	FIT, SOLVE	DATA=
Specifies the input data set for parameters	FIT, SOLVE	ESTDATA=
Specifies the method for handling missing values	FIT	MISSING=
Specifies the input data set for parameters	MODEL	PARMSDATA=
Requests that the procedure produce graphics via the Output Delivery System	MODEL	PLOTS=
Specifies the output data set for residual, predicted, or actual values	FIT	OUT=
Specifies the output data set for solution mode results	SOLVE	OUT=
Writes the actual values to OUT= data set	FIT	OUTACTUAL
Selects all output options	FIT	OUTALL
Writes the covariance matrix of the estimates	FIT	OUTCOV
Writes the parameter estimates to a data set	FIT	OUTEST=
Writes the parameter estimates to a data set	MODEL	OUTPARMS=
Writes the observations used to start the lags	SOLVE	OUTLAGS
Writes the predicted values to the OUT= data set	FIT	OUTPREDICT
Writes the residual values to the OUT= data set	FIT	OUTRESID
Writes the covariance matrix of the equation errors to a data set	FIT	OUTS=
Writes the **S** matrix used in the objective function definition to a data set	FIT	OUTSUSED=
Writes the estimate of the variance matrix of the moment generating function	FIT	OUTV=
Reads the covariance matrix of the equation errors	FIT, SOLVE	SDATA=
Reads the covariance matrix for GMM and ITGMM	FIT	VDATA=
Specifies the name of the time variable	FIT, SOLVE, MODEL	TIME=
Selects the estimation type to read	FIT, SOLVE	TYPE=
General ESTIMATE Statement Options		
Specifies the name of the data set in which the estimate of the functions of the parameters are to be written	ESTIMATE	OUTEST=

Description	Statement	Option
Writes the covariance matrix of the functions of the parameters to the OUTEST= data set	ESTIMATE	OUTCOV
Prints the covariance matrix of the functions of the parameters	ESTIMATE	COVB
Prints the correlation matrix of the functions of the parameters	ESTIMATE	CORRB
Printing Options for FIT Tasks		
Prints the modified Breusch-Pagan test for heteroscedasticity	FIT	BREUSCH
Prints the Chow test for structural breaks	FIT	CHOW=
Prints collinearity diagnostics	FIT	COLLIN
Prints the correlation matrices	FIT	CORR
Prints the correlation matrix of the parameters	FIT	CORRB
Prints the correlation matrix of the residuals	FIT	CORRS
Prints the covariance matrices	FIT	COV
Prints the covariance matrix of the parameters	FIT	COVB
Prints the covariance matrix of the residuals	FIT	COVS
Prints Durbin-Watson d statistics	FIT	DW
Prints first-stage R^2 statistics	FIT	FSRSQ
Prints Godfrey's tests for autocorrelated residuals for each equation	FIT	GODFREY
Prints Hausman's specification test	FIT	HAUSMAN
Prints tests of normality of the model residuals	FIT	NORMAL
Prints the predictive Chow test for structural breaks	FIT	PCHOW=
Specifies all the printing options	FIT	PRINTALL
Prints White's test for heteroscedasticity	FIT	WHITE
Options to Control FIT Iteration Output		
Prints the inverse of the crossproducts Jacobian matrix	FIT	I
Prints a summary iteration listing	FIT	ITPRINT
Prints a detailed iteration listing	FIT	ITDETAILS
Prints the crossproduct Jacobian matrix	FIT	XPX
Specifies all the iteration printing-control options	FIT	ITALL
Options to Control the Minimization Process		
Specifies the convergence criteria	FIT	CONVERGE=
Selects the Hessian approximation used for FIML	FIT	HESSIAN=

Description	Statement	Option
Specifies the local truncation error bound for the integration	FIT, SOLVE, MODEL	LTEBOUND=
Specifies the maximum number of iterations allowed	FIT	MAXITER=
Specifies the maximum number of subiterations allowed	FIT	MAXSUBITER=
Selects the iterative minimization method to use	FIT	METHOD=
Specifies the smallest allowed time step to be used in the integration	FIT, SOLVE, MODEL	MINTIMESTEP=
Modify the iterations for estimation methods that iterate the **S** matrix or the **V** matrix	FIT	NESTIT
Specifies the smallest pivot value	MODEL, FIT, SOLVE	SINGULAR
Specifies the number of minimization iterations to perform at each grid point	FIT	STARTITER=
Specifies a weight variable	WEIGHT	
Options to Read and Write Model Files		
Deletes a model from a model file	DELETEMODEL	MODNAME=
Reads a model from one or more input model files	INCLUDE	MODEL=
Suppresses the default output of the model file	MODEL, RESET	NOSTORE
Specifies the name of an output model file	MODEL, RESET	OUTMODEL=
Deletes the current model	RESET	PURGE
Options to List or Analyze the Structure of the Model		
Identifies equations in a dependency analysis	EQGROUP	
Identifies variables in a dependency analysis	VARGROUP	
Prints a dependency analysis of a simulation model	SOLVE	ANALYZEDEP=
Prints a dependency structure of a normal form model	MODEL	BLOCK
Prints a graph of the dependency structure of a normal form model	MODEL	GRAPH
Prints the model program and variable lists	MODEL	LIST
Prints the derivative tables and compiled model program code	MODEL	LISTCODE
Prints a dependency list	MODEL	LISTDEP
Prints a table of derivatives	MODEL	LISTDER
Prints a cross-reference of the variables	MODEL	XREF
General Printing Control Options		
Expands parts of the printed output	FIT, SOLVE	DETAILS

Description	Statement	Option
Prints a message for each statement as it is executed	FIT, SOLVE	FLOW
Selects the maximum number of execution errors that can be printed	FIT, SOLVE	MAXERRORS=
Requests a comprehensive memory usage summary	FIT, SOLVE, MODEL, RESET	MEMORYUSE
Selects the number of decimal places shown in the printed output	FIT, SOLVE	NDEC=
Suppresses the normal printed output	FIT, SOLVE	NOPRINT
Turns off the NOPRINT option	RESET	PRINT
Specifies all the noniteration printing options	FIT, SOLVE	PRINTALL
Prints tables which summarize missing value calculations	FIT, SOLVE, MODEL	REPORTMISSINGS
Prints the result of each operation as it is executed	FIT, SOLVE	TRACE
Statements that Declare Variables		
Associates a name with a list of variables and constants	ARRAY	
Declares a variable to have a fixed value	CONTROL	
Declares a variable to be a dependent or endogenous variable	ENDOGENOUS	
Declares a variable to be an independent or exogenous variable	EXOGENOUS	
Specifies identifying variables	ID	
Assigns a label to a variable	LABEL	
Selects additional variables to be output	OUTVARS	
Declares a variable to be a parameter	PARAMETERS	
Forces a variable to hold its value from a previous observation	RETAIN	
Declares a model variable	VAR	
Declares an instrumental variable	INSTRUMENTS	
Omits the default intercept term in the instruments list	INSTRUMENTS	NOINT
General FIT Statement Options		
Omits parameters from the estimation	FIT	DROP=
Associates a variable with an initial value as a parameter or a constant	FIT	INITIAL=
Bypasses OLS to get initial parameter estimates for GMM, ITGMM, or FIML	FIT	NOOLS
Bypasses 2SLS to get initial parameter estimates for GMM, ITGMM, or FIML	FIT	NO2SLS
Specifies the parameters to estimate	FIT	PARMS=

Description	Statement	Option
Requests confidence intervals on estimated parameters	FIT	PRL=
Selects a grid search	FIT	START=
Options to Control the Estimation Method Used		
Specifies nonlinear ordinary least squares	FIT	OLS
Specifies iterated nonlinear ordinary least squares	FIT	ITOLS
Specifies seemingly unrelated regression	FIT	SUR
Specifies iterated seemingly unrelated regression	FIT	ITSUR
Specifies two-stage least squares	FIT	2SLS
Specifies iterated two-stage least squares	FIT	IT2SLS
Specifies three-stage least squares	FIT	3SLS
Specifies iterated three-stage least squares	FIT	IT3SLS
Specifies full information maximum likelihood	FIT	FIML
Specifies simulated method of moments	FIT	NDRAW
Specifies number of draws for the V matrix	FIT	NDRAWV
Specifies number of initial observations for SMM	FIT	NPREOBS
Selects the variance-covariance estimator used for FIML	FIT	COVBEST=
Specifies generalized method of moments	FIT	GMM
Specifies the kernel for GMM and ITGMM	FIT	KERNEL=
Specifies iterated generalized method of moments	FIT	ITGMM
Specifies the type of generalized inverse used for the covariance matrix	FIT	GINV=
Specifies the denominator for computing variances and covariances	FIT	VARDEF=
Specifies adding the variance adjustment for SMM	FIT	ADJSMMV
Specifies variance correction for heteroscedasticity	FIT	HCCME=
Specifies GMM variance under arbitrary weighting matrix	FIT	GENGMMV
Specifies GMM variance under optimal weighting matrix	FIT	NOGENGMMV
Solution Mode Options		
Selects a subset of the model equations	SOLVE	SATISFY=
Solves only for missing variables	SOLVE	FORECAST
Solves for all solution variables	SOLVE	SIMULATE

Description	Statement	Option
Solution Mode Options: Lag Processing		
Uses solved values in the lag functions	SOLVE	DYNAMIC
Uses actual values in the lag functions	SOLVE	STATIC
Produces successive forecasts to a fixed forecast horizon	SOLVE	NAHEAD=
Selects the observation to start dynamic solutions	SOLVE	START=
Solution Mode Options: Numerical Methods		
Specifies the maximum number of iterations allowed	SOLVE	MAXITER=
Specifies the maximum number of subiterations allowed	SOLVE	MAXSUBITER=
Specifies the convergence criteria	SOLVE	CONVERGE=
Computes a simultaneous solution using a Jacobi-like iteration	SOLVE	JACOBI
Computes a simultaneous solution using a Gauss-Seidel-like iteration	SOLVE	SEIDEL
Computes a simultaneous solution using Newton's method	SOLVE	NEWTON
Computes a nonsimultaneous solution	SOLVE	SINGLE
Monte Carlo Simulation Options		
Specifies quasi-random number generator	SOLVE	QUASI=
Specifies pseudo-random number generator	SOLVE	PSUEDO=
Repeats the solution multiple times	SOLVE	RANDOM=
Initializes the pseudo-random number generator	SOLVE	SEED=
Specifies copula options	SOLVE	COPULA=
Solution Mode Printing Options		
Prints between data points integration values for the DERT. variables and the auxiliary variables	FIT, SOLVE, MODEL	INTGPRINT
Prints the solution approximation and equation errors	SOLVE	ITPRINT
Prints the solution values and residuals at each observation	SOLVE	SOLVEPRINT
Prints various summary statistics	SOLVE	STATS
Prints tables of Theil inequality coefficients	SOLVE	THEIL
Specifies all printing control options	SOLVE	PRINTALL

Description	Statement	Option
General TEST Statement Options		
Specifies that a Wald test be computed	TEST	WALD
Specifies that a Lagrange multiplier test be computed	TEST	LM
Specifies that a likelihood ratio test be computed	TEST	LR
Request all three types of tests	TEST	ALL
Specifies the name of an output SAS data set that contains the test results	TEST	OUT=
Miscellaneous Statements		
Specifies the range of observations to be used	RANGE	
Subsets the data set with BY variables	BY	

PROC MODEL Statement

PROC MODEL *options* **;**

The following options can be specified in the PROC MODEL statement. All of the nonassignment options (the options that do not accept a value after an equal sign) can have NO prefixed to the option name in the RESET statement to turn the option off. The default case is not explicitly indicated in the discussion that follows. Thus, for example, the option DETAILS is documented in the following, but NODETAILS is not documented since it is the default. Also, the NOSTORE option is documented because STORE is the default.

Data Set Options

DATA=*SAS-data-set*

names the input data set. Variables in the model program are looked up in the DATA= data set and, if found, their attributes (type, length, label, format) are set to be the same as those in the input data set (if not previously defined otherwise). The values for the variables in the program are read from the input data set when the model is estimated or simulated by FIT and SOLVE statements.

OUTPARMS=*SAS-data-set*

writes the parameter estimates to a SAS data set. See the section "Output Data Sets" on page 1226 for details.

PARMSDATA=*SAS-data-set*

names the SAS data set that contains the parameter estimates. In PROC MODEL, you have several options to specify starting values for the parameters to be estimated. When more than one option is specified, the options are implemented in the following order of precedence (from highest to lowest): the START= option, the PARMS statement initialization value, the ESTDATA= option, and the PARMSDATA= option. If no options are specified for the starting value, the default value of 0.0001 is used. See the section "Input Data Sets" on page 1221 for details.

PLOTS< *(global-plot-options)* > < *=(plot-request . . .)* >
selects plots that the MODEL procedure produces via the Output Delivery System. For general information about ODS Graphics, see Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*). The *global-plot-options* apply to all relevant plots generated by the MODEL procedure. The *global-plot-options* and specific *plot-request* options supported by the MODEL procedure follow.

Global Plot Options

ONLY suppresses the default plots. Only the plots specifically requested are produced.

UNPACKPANEL displays each graph separately. (By default, some graphs can appear together in a single panel.)

Specific Plot Options

ALL requests that all plots appropriate for the particular analysis be produced.

ACF produces the autocorrelation function plot.

DEPENDENCY< (OUTLINE=ON | OFF) > produces the dependency analysis plots. Specifying the OUTLINE= option displays, or suppresses outlines around the dependency cells.

IACF produces the inverse autocorrelation function plot of residuals.

PACF produces the partial autocorrelation function plot of residuals.

FITPLOT plots the predicted and actual values.

COOKSD produces the Cook's D plot.

QQ produces a QQ plot of residuals.

RESIDUAL | RES plots the residuals.

STUDENTRESIDUAL plots the studentized residuals.

RESIDUALHISTOGRAM | RESIDHISTOGRAM plots the histogram of residuals.

NONE suppresses all plots.

Options to Read and Write Model Files

MODEL=*model-name*

MODEL=*(model-list)*
reads the model from one or more input model files created by previous PROC MODEL executions. Model files are written by the OUTMODEL= option.

NOSTORE
suppresses the default output of the model file. This option is applicable only when FIT or SOLVE statements are not used, the MODEL= option is not used, and when a model is specified.

OUTCAT=(*outcat-name* **MODNAME=***model-key* **<** *outcat-options* **>)**

SLIST=(*outcat-name* **MODNAME=***model-key* **<** *outcat-options* **>)**
specifies the name and *model-key* for writing fitted model files. The *model-key* is a SAS name. Files written using the OUTCAT= option are used by SAS Risk Dimensions. The OUTCAT= option only applies to FIT statements. You can specify the following *outcat-options*:

DIM=*n* specifies the dimensionality of the model.

GROUP|MODGROUP=*group* specifies a SAS name which is the group for the model.

INTERVAL=*interval* specifies the time interval between observations.

MODLABEL=*label* specifies a label for the model.

STARTDATE=*date* specifies the starting date of the model.

OUTMODEL=*model-name*

specifies the name of an output model file to which the model is to be written. Starting with SAS 9.2, model files are being stored as XML-based SAS data sets instead of being stored as members of a SAS catalog as in earlier releases. This makes MODEL files more readily extendable in the future and enables Java-based applications to read the MODEL files directly. To change this behavior, use the SAS *global-CMPMODEL-options*. You can choose the format in which the output model file is stored and read by using the CMPMODEL=*global-CMPMODEL-options* in an OPTIONS statement as follows.

OPTIONS CMPMODEL=*global-CMPMODEL-options***;**

You can specify the following *global-CMPMODEL-options*:

CATALOG specifies that model files be written and read from SAS catalogs only.

XML specifies that model files be written and read from XML data sets only.

BOTH specifies that model files be written to both XML and CATALOG formats. When BOTH is specified, model files are read from the data set first and read from the SAS catalog only if the data set is not found. This is the default.

Options to List or Analyze the Structure of the Model

These options produce reports on the structure of the model or list the programming statements that define the models. These options are automatically reset (turned off) after the reports are printed. To turn these options back on after a RUN statement has been entered, use the RESET statement or specify the options in a FIT or SOLVE statement.

ANALYZEDEP=(*dependency-plot1* < *dependency-plot2 . . .* >**)**

plots analyses of the dependencies among equations and solve variables. Each *dependency-plot* is one of the following:

BLOCK specifies a block dependency matrix of the entire system.

BLOCK(*eq-list*,*var-list*) specifies a block dependency matrix for a subset of equations and solve variables.

DETAILS specifies a dependency matrix of all equations and solve variables.

DETAILS(*eq-list*,*var-list*) specifies a dependency matrix for a subset of equations and solve variables.

NOLISTBLOCK suppresses the listing of dependency blocks.

You can specify which equations and solve variables are included in the dependency analysis by qualifying both the BLOCK and DETAILS *dependency-plot* options with a pair of lists. The first list in the pair is the *eq-list*. It specifies which equations to include in the dependency analysis. You can specify a mix of equation names and equation group labels in the *eq-list*. The MODEL procedure replaces each equation group label in the *eq-list* with the list of equations that are specified in the corresponding EQGROUP statement. The second list in the pair is the *var-list*. It specifies which solve variables to include in the dependency analysis. You can specify a mix of variable names and variable group labels in the *var-list*. The MODEL procedure replaces each variable group label in the *var-list* with the list of variables that are specified in the corresponding VARGROUP statement. By default, when you specify a BLOCK option, a listing of the equations and solve variables that form each dependency block is generated. The NOLISTBLOCK option suppresses this listing. The ANALYZEDEP= option applies only to SOLVE steps. For more information about the analyses that are performed by the ANALYZEDEP= option, see the section "Diagnostics and Debugging" on page 1284.

BLOCK

prints an analysis of the structure of the model given by the assignments to model variables that appear in the model program. This analysis includes a classification of model variables into endogenous (dependent) and exogenous (independent) groups based on the presence of the variable on the left side of an assignment statement. The endogenous variables are grouped into simultaneously determined blocks. The dependency structure of the simultaneous blocks and exogenous variables is also printed. The BLOCK option cannot analyze dependencies implied by general form equations.

GRAPH

prints the graph of the dependency structure of the model. The GRAPH option also invokes the BLOCK option and produces a graphical display of the information listed by the BLOCK option.

LIST

prints the model program and variable lists, including the statements added by PROC MODEL and macros.

LISTALL

selects the LIST, LISTDEP, LISTDER, and LISTCODE options.

LISTCODE

prints the derivative tables and compiled model program code. LISTCODE is a debugging feature and is not normally needed.

LISTDEP

prints a report that lists for each variable in the model program the variables that depend on it and that it depends on. These lists are given separately for current-period values and for lagged values of the variables.

The information displayed is the same as that used to construct the BLOCK report but differs in that the information is listed for all variables (including parameters, control variables, and program variables), not just for the model variables. Classification into endogenous and exogenous groups and analysis of simultaneous structure is not done by the LISTDEP report.

LISTDER
prints a table of derivatives for FIT and SOLVE tasks. (The LISTDER option is applicable only for the default NEWTON method for SOLVE tasks.) The derivatives table shows each nonzero derivative computed for the problem. The derivative listed can be a constant, a variable in the model program, or a special derivative variable created to hold the result of the derivative expression. This option is turned on by the LISTCODE and PRINTALL options.

XREF
prints a cross-reference of the variables in the model program that shows where each variable was referenced or given a value. The XREF option is normally used in conjunction with the LIST option. A more detailed description is given in the section "Diagnostics and Debugging" on page 1284.

General Printing Control Options

DETAILS
specifies the detailed printout. Parts of the printed output are expanded when the DETAILS option is specified. The following additional graphs of the residuals are produced when graphics output is enabled: ACF, PACF, IACF, white noise, and QQ plot versus the normal.

FLOW
prints a message for each statement in the model program as it is executed. This debugging option is needed very rarely and produces voluminous output.

MAXERRORS=*n*
specifies the maximum number of execution errors that can be printed. The default is MAXERRORS=50.

MEMORYUSE
prints a report of the memory required for the various parts of the analysis.

NDEC=*n*
specifies the precision of the format that PROC MODEL uses when printing various numbers. The default is NDEC=3, which means that PROC MODEL attempts to print values by using the D format but ensures that at least three significant digits are shown. If the NDEC= value is greater than nine, the BEST. format is used. The smallest value allowed is NDEC=2.

The NDEC= option affects the format of most, but not all, of the floating point numbers that PROC MODEL can print. For some values (such as parameter estimates), a precision limit one or two digits greater than the NDEC= value is used. This option does not apply to the precision of the variables in the output data set.

NOPRINT
suppresses the normal printed output but does not suppress error listings. Using any other print option turns the NOPRINT option off. The PRINT option can be used with the RESET statement to turn off NOPRINT.

PRINTALL
turns on all the printing-control options. The options set by PRINTALL are DETAILS; the model information options LIST, LISTDEP, LISTDER, XREF, BLOCK, and GRAPH; the FIT task printing options FSRSQ, COVB, CORRB, COVS, CORRS, DW, and COLLIN; and the SOLVE task printing options STATS, THEIL, SOLVEPRINT, and ITPRINT.

REPORTMISSINGS
prints tables that summarize missing values that are encountered during a SOLVE or FIT task. The missing values that are summarized in these tabular reports can be produced by missing values in the DATA= data set or by calculations in the model program that generate missing values. The number of missing values that are reported can be limited by using the MAXERRORS= option.

TRACE
prints the result of each operation in each statement in the model program as it is executed, in addition to the information printed by the FLOW option. This debugging option is needed very rarely and produces voluminous output.

FIT Task Options

The following options are used in the FIT statement (parameter estimation) and can also be used in the PROC MODEL statement: COLLIN, CONVERGE=, CORR, CORRB, CORRS, COVB, COVBEST=, COVS, DW, FIML, FSRSQ, GMM, HESSIAN=, I, INTGPRINT, ITALL, ITDETAILS, ITGMM, ITPRINT, ITOLS, ITSUR, IT2SLS, IT3SLS, KERNEL=, LTEBOUND=, MAXITER=, MAXSUBITER=, METHOD=, MINTIMESTEP=, NESTIT, N2SLS, N3SLS, OLS, OUTPREDICT, OUTRESID, OUTACTUAL, OUTLAGS, OUTALL, OUTCOV, SINGULAR=, STARTITER=, SUR, TIME=, VARDEF, and XPX. See the section "FIT Statement" on page 1107 for a description of these options.

When used in the PROC MODEL or RESET statement, these are default options for subsequent FIT statements. For example, the statement

```
proc model n2sls ... ;
```

makes two-stage least squares the default parameter estimation method for FIT statements that do not specify an estimation method.

SOLVE Task Options

The following options for the SOLVE statement can also be used in the PROC MODEL statement: CONVERGE=, DYNAMIC, FORECAST, INTGPRINT, ITPRINT, JACOBI, LTEBOUND=, MAXITER=, MAXSUBITER=, MINTIMESTEP=, NAHEAD=, NEWTON, OUTPREDICT, OUTRESID, OUTACTUAL, OUTLAGS, OUTERRORS, OUTALL, SEED=, SEIDEL, SIMULATE, SINGLE, SINGULAR=, SOLVEPRINT, START=, STATIC, STATS, THEIL, TIME=, and TYPE=. For more information about these options, see section "SOLVE Statement" on page 1123.

When used in the PROC MODEL or RESET statement, these options provide default values for subsequent SOLVE statements.

BOUNDS Statement

BOUNDS *bound1* < , *bound2* . . . > **;**

The BOUNDS statement imposes simple boundary constraints either on the parameters in an estimation or on the solution variables specified in a solve operation. A BOUNDS statement that applies to parameters constrains the parameters estimated in the preceding FIT statement or, in the absence of a preceding FIT

statement, in the following FIT statement. A BOUNDS statement that is applied to solution variables constrains the solution of the preceding SOLVE statement or, in the absence of a preceding SOLVE statement, of the following SOLVE statement. You can specify any number of BOUNDS statements.

Each *bound* is composed of either parameters or solution variables, constants, and inequality operators:

item operator item **<** *operator item* **<** *operator item* **... > >**

For BOUNDS statements that apply to FIT statements, each *item* is a constant, the name of an estimated parameter, or a list of parameter names. For BOUNDS statements that apply to SOLVE statements, each *item* is a constant, the name of a solution variable, or a list of solution variables. Each *operator* is <, >, <=, or >=.

You can use either the BOUNDS statement or the RESTRICT statement to impose boundary constraints when estimating parameters or solving for solution variables.

The BOUNDS statement provides a simpler syntax for specifying boundary constraints than the RESTRICT statement. For more information about the computational details of estimation and solutions with inequality restrictions, see the section "RESTRICT Statement" on page 1121.

Parameter Estimates

Each active boundary constraint on estimated parameters is associated with a Lagrange multiplier. In the printed output and in the OUTEST= data set, the Lagrange multiplier estimates are identified with the names BOUND0, BOUND1, and so forth. The probabilities of the Lagrange multipliers are computed by using a beta distribution (LaMotte 1994). To give the constraints more descriptive names, use the RESTRICT statement instead of the BOUNDS statement.

The following BOUNDS statement constrains the estimates of the parameters A and B and the ten parameters P1 through P10 to be between 0 and 1. This example illustrates the use of parameter lists to specify boundary constraints.

```
bounds 0 < a b p1-p10 < 1;
```

The following statements show how to use the BOUNDS statement, and they produce the output shown in Figure 19.13:

```
title 'Holzman Function (1969), Himmelblau No. 21, N=3';
data zero;
   do i = 1 to 99;
      output;
   end;
run;

proc model data=zero;
   parms x1= 100 x2= 12.5 x3=  3;
   bounds .1 <= x1 <= 100,
           0 <= x2 <=  25.6,
           0 <= x3 <=   5;

   t = 2 / 3;
   u = 25 + (-50 * log(0.01 * i )) ** t;
   v = (u - x2) ** x3;
   w = exp(-v / x1);
   eq.foo = -.01 * i + w;
```

```
   fit foo / method=marquardt;
run;
```

Figure 19.13 Output from Bounded Estimation

Holzman Function (1969), Himmelblau No. 21, N=3

The MODEL Procedure

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
x1	49.99999	0	.	.
x2	25	0	.	.
x3	1.5	0	.	.

Solution Variables

Boundary constraints on solution variables can be used to specify which solution is reported when an equation has multiple solutions. The BOUNDS statement in the following example causes its associated SOLVE statement to compute only the negative value of the solution variable shown in Figure 19.14:

```
data d;
   date = 0;
run;

proc model data=d;
   endo x;
   bounds x < 0;

   eq.sqrt = x**2 - 4;

   solve / optimize out=o;
run;

proc print data = o; run;
```

Figure 19.14 Listing of OUT= Data Set Created by a Bounded SOLVE Statement

Obs	_TYPE_	_MODE_	_ERRORS_	x
1	PREDICT	SIMULATE	0	-2

BY Statement

BY *variables* **;**

A BY statement is used with the FIT statement to obtain separate estimates for observations in groups defined by the BY variables. If an output model file is written using the OUTMODEL= option, the parameter values that are stored are those from the last BY group processed. To save parameter estimates for each BY group, use the OUTEST= option in the FIT statement.

A BY statement is used with the SOLVE statement to obtain solutions for observations in groups defined by the BY variables. If the BY variables in the DATA= data set and the ESTDATA= data set are identical, then the two data sets are synchronized and the calculations are performed by using the data and parameters for each BY group. This holds for BY variables in the SDATA= data set as well. If the BY variables do not match, BY-group processing is abandoned in either the ESTDATA= data set or the SDATA= data set, whichever has the missing BY value. If the DATA= data set does not contain BY variables and the ESTDATA= data set or the SDATA= data set does, then BY-group processing is performed for the ESTDATA= data set and the SDATA= data set by reusing the data in the DATA= data set for each BY group.

If both FIT and SOLVE tasks require BY-group processing, then two separate BY statements are needed. If parameters for each BY group in the OUTEST = data set that is obtained from the FIT task are to be used for the corresponding BY group for the SOLVE task, then one of the two BY statements must appear after the SOLVE statement.

The following linear regression example illustrates the use of BY-group processing. Both the data sets A and D to be used for fitting and solving, respectively, have three groups.

```
/*------ data set for fit task------ */
data a ;
   do group = 1 to 3 ;
      do i = 1 to 100 ;
         x = normal(1);
         y = 2 + 3*x + rannor(1) ;
         output ;
      end ;
   end ;
run ;

/*------ data set for solve task------ */
data d ;
   do group = 1 to 3 ;
      x = normal(1) ;
      output ;
   end ;
run ;

/* ------  2 BY statements, one of them appear after SOLVE statement ------ */
proc model data = a ;
   by group ;
   y = a0 + a1*x ;
   fit y / outest = b1 ;
   solve y / data = d estdata = b1 out = c1 ;
   by group ;
run;

proc print data = b1 ;run;
proc print data = c1 ; run;
```

Each of the parameter estimates obtained from the BY group processing in the FIT statement shown in Figure 19.15 is used in the corresponding BY group variables in the SOLVE statement. The output dataset is shown in Figure 19.16.

Figure 19.15 Listing of OUTEST= Data Set Created in the FIT Statement with Two BY Statements

Obs	group	_NAME_	_TYPE_	_STATUS_	_NUSED_	a0	a1
1	1		OLS	0 Converged	100	2.00338	3.00298
2	2		OLS	0 Converged	100	2.05091	3.08808
3	3		OLS	0 Converged	100	2.15528	3.04290

Figure 19.16 Listing of OUT= Data Set Created in the SOLVE Statement with Two BY Statements

Obs	group	_TYPE_	_MODE_	_ERRORS_	y	x
1	1	PREDICT	SIMULATE	0	7.42322	1.80482
2	2	PREDICT	SIMULATE	0	1.80413	-0.07992
3	3	PREDICT	SIMULATE	0	3.36202	0.39658

If only one BY statement is used and it appears before the SOLVE statement, then parameters for the last BY group in the OUTEST = data set are used for all BY groups for the SOLVE task.

```
/*------ 1 BY statement that appears before SOLVE statement------ */
proc model data = a ;
   by group ;
   y = a0 + a1*x ;
   fit y / outest = b2 ;
   solve y / data = d estdata = b2 out = c2 ;
run;

proc print data = b2 ; run;
proc print data = c2 ; run;
```

The estimates of the parameters are shown in Figure 19.17, and the output data set of the SOLVE statement is shown in Figure 19.18. Hence, the estimates and the predicted values obtained in the last BY group variable of both DATA C1 and C2 are the same while the others do not match.

Figure 19.17 Listing of OUTEST= Data Set Created in the FIT Statement with One BY Statement That Appears before the SOLVE Statement

Obs	group	_NAME_	_TYPE_	_STATUS_	_NUSED_	a0	a1
1	1		OLS	0 Converged	100	2.00338	3.00298
2	2		OLS	0 Converged	100	2.05091	3.08808
3	3		OLS	0 Converged	100	2.15528	3.04290

Figure 19.18 Listing of OUT= Data Set Created in the SOLVE Statement with One BY Statement That Appears before the SOLVE Statement

Obs	_TYPE_	_MODE_	_ERRORS_	y	x
1	PREDICT	SIMULATE	0	7.64717	1.80482
2	PREDICT	SIMULATE	0	1.91211	-0.07992
3	PREDICT	SIMULATE	0	3.36202	0.39658

If only one BY statement is used and it appears after the SOLVE statement, then BY group processing does not apply to the FIT task. In this case, the OUTEST=data set does not contain the BY variable, and the single set of parameter estimates obtained from the FIT task are used for all BY groups during the SOLVE task.

```
/*------ 1 BY statement that appears after SOLVE statement------*/
proc model data = a ;
   y = a0 + a1*x ;
   fit y / outest = b3 ;
   solve y / data = d estdata = b3 out = c3 ;
   by group ;
run;

proc print data = b3 ; run;
proc print data = c3 ; run;
```

The output data B3 and C3 are listed in Figure 19.19 and Figure 19.20, respectively.

Figure 19.19 Listing of OUTEST= Data Set Created in the FIT Statement with One BY Statement That Appears after the SOLVE Statement

Obs	_NAME_	_TYPE_	_STATUS_	_NUSED_	a0	a1
1		OLS	0 Converged	300	2.06624	3.04219

Figure 19.20 Listing of OUT= Data Set Created in the First SOLVE Statement with One BY Statement That Appears after the SOLVE Statement

Obs	group	_TYPE_	_MODE_	_ERRORS_	y	x
1	1	PREDICT	SIMULATE	0	7.55686	1.80482
2	2	PREDICT	SIMULATE	0	1.82312	-0.07992
3	3	PREDICT	SIMULATE	0	3.27270	0.39658

CONTROL Statement

CONTROL *variable* < *value* > . . . **;**

The CONTROL statement declares control variables and specifies their values. A control variable is like a parameter except that it has a fixed value and is not estimated from the data. You can use control variables for constants in model equations that you might want to change in different solution cases. You can use control variables to vary the program logic. Unlike the retained variables, these values are fixed across iterations.

DELETEMODEL Statement

DELETEMODEL *model* < *MODNAME=model-name* > **;**

The DELETEMODEL statement deletes a model created using the OUTMODEL= option in a previous PROC MODEL execution. The *model* argument specifies the catalog or XML-based data set containing the model to be deleted, and the *model-name* argument specifies which model is to be deleted.

ENDOGENOUS Statement

ENDOGENOUS *variable* < *initial-values* > . . . **;**

The ENDOGENOUS statement declares model variables and identifies them as endogenous. You can declare model variables with an ENDOGENOUS statement instead of with a VAR statement to help document the model or to indicate the default solution variables. The variables declared endogenous are solved when a SOLVE statement does not indicate which variables to solve. Valid abbreviations for the ENDOGENOUS statement are ENDOG and ENDO.

The DEPENDENT statement is equivalent to the ENDOGENOUS statement and is provided for the convenience of noneconometric practitioners.

The ENDOGENOUS statement optionally provides initial values for lagged dependent variables. See the section "Lag Logic" on page 1277 for more information.

EQGROUP Statement

EQGROUP *label*=*equation*. . . **;**

The EQGROUP statement applies a group label to the specified list of equations in the model program. Equation groups identify sets of related equations. The equation groups can be used by the ANALYZEDEP= option in a subsequent SOLVE statement to help specify and understand the role of groups of equations in a SOLVE step. If an equation appears in more than one EQGROUP statement, the label that is specified in the last EQGROUP statement is applied to that equation.

ERRORMODEL Statement

ERRORMODEL *equation-name* ∼ *distribution* < **CDF=** *CDF(options)* > **;**

The ERRORMODEL statement is the mechanism for specifying the distribution of the residuals. You must specify the dependent/endogenous variables or general form model name, a tilde (∼), and then a *distribution* with its parameters. You can specify the following options:

Options to Specify the Distribution

CAUCHY(< *location, scale* **>)**
specifies the Cauchy distribution. This option is supported only for simulation. The arguments correspond to the arguments of the SAS CDF function that computes the cumulative distribution function (ignoring the random variable argument).

CHISQUARED (*df* <, *nc*> **)**
specifies the χ^2 distribution. This option is supported only for simulation. The arguments correspond to the arguments of the SAS CDF function (ignoring the random variable argument).

GENERAL(*Likelihood* **<** *, parm1, parm2, . . . parmn* **>)**
specifies the negative of a general log-likelihood function that you construct by using SAS programming statements. The procedure minimizes the negative log-likelihood function specified. *parm1*, *parm2*, . . . *parmn* are optional parameters for this distribution and are used for documentation purposes only.

F(*ndf, ddf* **<** *, nc* **>)**
specifies the F distribution. This option is supported only for simulation. The arguments correspond to the arguments of the SAS CDF function (ignoring the random variable argument).

NORMAL(v_1 v_2 $\ldots$ v_n **)**
specifies a multivariate normal (Gaussian) distribution with mean *0* and variances v_1 through v_n.

POISSON(*mean* **)**
specifies the Poisson distribution. This option is supported only for simulation. The arguments correspond to the arguments of the SAS CDF function (ignoring the random variable argument).

T($v_1 v_2 \cdots v_n$, *df* **)**
specifies a multivariate *t* distribution with noncentrality *0*, variance v_1 through v_n, and common degrees of freedom df.

UNIFORM(< *left, right* **>)**
specifies the uniform distribution. This option is supported only for simulation. The arguments correspond to the arguments of the SAS CDF function (ignoring the random variable argument).

Options to Specify the CDF for Simulation

CDF=(*CDF(options)* **)**
specifies the univariate distribution that is used for simulation so that the estimation can be done for one set of distributional assumptions and the simulation for another. The *CDF* can be any of the distributions from the previous section with the exception of the general likelihood. In addition, you can specify the empirical distribution of the residuals.

EMPIRICAL= (<TAILS=*(options)***>)**
uses the sorted residual data to create an empirical CDF.

TAILS=(*tail-options* **)**
specifies how to handle the tails in computing the inverse CDF from an empirical distribution, where *tail-options* are:

NORMAL	specifies the normal distribution to extrapolate the tails.
T(*df*)	specifies the *t* distribution to extrapolate the tails.
PERCENT= *p*	specifies the percentage of the observations to use in constructing each tail. The default for the PERCENT= option is 10. A normal distribution or a *t* distribution is used to extrapolate the tails to infinity. The variance for the tail distribution is obtained from the data so that the empirical CDF is continuous.

ESTIMATE Statement

ESTIMATE *item* < *, item . . .* > < *,/ options* > **;**

The ESTIMATE statement computes estimates of functions of the parameters.

The ESTIMATE statement refers to the parameters estimated by the associated FIT statement (that is, to either the preceding FIT statement or, in the absence of a preceding FIT statement, to the following FIT statement). You can use any number of ESTIMATE statements.

Let $\mathbf{h}(\theta)$ denote the function of parameters that needs to be estimated. Let $\hat{\theta}$ denote the unconstrained estimate of the parameter of interest, θ. Let $\hat{\mathbf{V}}$ be the estimate of the covariance matrix of θ. Denote

$$\mathbf{A}(\theta) = \partial h(\theta)/\partial\theta \mid_{\hat{\theta}}$$

Then the standard error of the parameter function estimate is computed by obtaining the square root of $\mathbf{A}(\hat{\theta})\hat{\mathbf{V}}\mathbf{A}'(\hat{\theta})$. This is the same as the variance needed for a Wald type test statistic with null hypothesis $h(\theta) = 0$.

If the expression of the function in the ESTIMATE statement includes a variable, then the value used in computing the function estimate is the last observation of the variable in the DATA= data set.

If you specify options on the ESTIMATE statement, a comma is required before the "/" character that separates the test expressions from the options, since the "/" character can also be used within test expressions to indicate division. Each *item* is written as an optional name followed by an expression,

< "name" > expression

where *"name"* is a string used to identify the estimate in the printed output and in the OUTEST= data set.

Expressions can be composed of parameter names, arithmetic operators, functions, and constants. Comparison operators (such as = or <) and logical operators (such as &) cannot be used in ESTIMATE statement expressions. Parameters named in ESTIMATE expressions must be among the parameters estimated by the associated FIT statement.

You can use the following options in the ESTIMATE statement:

OUTEST=
specifies the name of the data set in which the estimate of the functions of the parameters are to be written. The format for this data set is identical to the OUTEST= data set for the FIT statement.

If you specify a *name* in the ESTIMATE statement, that name is used as the parameter name for the estimate in the OUTEST= data set. If no *name* is provided and the expression is just a symbol, the symbol name is used; otherwise, the string "_Estimate #" is used, where "#" is the variable number in the OUTEST= data set.

OUTCOV
writes the covariance matrix of the functions of the parameters to the OUTEST= data set in addition to the parameter estimates.

COVB
prints the covariance matrix of the functions of the parameters.

CORRB
prints the correlation matrix of the functions of the parameters.

The following statements are an example of the use of the ESTIMATE statement in a segmented model and produce the output shown in Figure 19.21:

```
data a;
   input y x @@;
datalines;
   .46 1  .47  2 .57  3 .61  4 .62  5 .68  6 .69  7
   .78 8  .70  9 .74 10 .77 11 .78 12 .74 13 .80 13
   .80 15 .78 16
;

title 'Segmented Model -- Quadratic with Plateau';
proc model data=a;

   x0 = -.5 * b / c;

   if x < x0 then y = a + b*x + c*x*x;
   else           y = a + b*x0 + c*x0*x0;

   fit y start=( a .45 b .5 c -.0025 );

   estimate 'Join point' x0 ,
            'plateau' a + b*x0 + c*x0**2 ;
run;
```

Figure 19.21 ESTIMATE Statement Output

Segmented Model -- Quadratic with Plateau

The MODEL Procedure

Nonlinear OLS Estimates

Term	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
Join point	12.7504	1.2785	9.97	<.0001	x0
plateau	0.777516	0.0123	63.10	<.0001	a + b*x0 + c*x0**2

EXOGENOUS Statement

EXOGENOUS *variable* < *initial-values* > . . . **;**

The EXOGENOUS statement declares model variables and identifies them as exogenous. You can declare model variables with an EXOGENOUS statement instead of with a VAR statement to help document the model or to indicate the default instrumental variables. The variables declared exogenous are used as instruments when an instrumental variables estimation method is requested (such as N2SLS or N3SLS) and an INSTRUMENTS statement is not used. Valid abbreviations for the EXOGENOUS statement are EXOG and EXO.

The INDEPENDENT statement is equivalent to the EXOGENOUS statement and is provided for the convenience of non-econometric practitioners.

The EXOGENOUS statement optionally provides initial values for lagged exogenous variables. See the section "Lag Logic" on page 1277 for more information.

FIT Statement

FIT *< equations > < PARMS=(parameter < values > . . .) > < START=(parameter values . . .) > < DROP=(parameter . . .) > < INITIAL=(variable < = parameter | constant > . . .) > < / options >* **;**

The FIT statement estimates model parameters by fitting the model equations to input data and optionally selects the equations to be fit. If the list of equations is omitted, all model equations that contain parameters are fitted.

The following options can be used in the FIT statement.

DROP= *(parameters . . .)*
specifies that the named parameters not be estimated. All the parameters in the equations fit are estimated except those listed in the DROP= option. The dropped parameters retain their previous values and are not changed by the estimation.

INITIAL= *(variable = < parameter | constant > . . .)*
associates a *variable* with an initial value as a *parameter* or a *constant*. This option applies only to ordinary differential equations. See the section "Ordinary Differential Equations" on page 1185 for more information.

PARMS= *(parameters [values] . . .)*
selects a subset of the parameters for estimation. When the PARMS= option is used, only the named parameters are estimated. Any parameters not specified in the PARMS= list retain their previous values and are not changed by the estimation.

In PROC MODEL, you have several options to specify starting values for the parameters to be estimated. When more than one option is specified, the options are implemented in the following order of precedence (from highest to lowest): the START= option, the PARMS statement initialization value, the ESTDATA= option, and the PARMSDATA= option. If no options are specified for the starting value, the default value of 0.0001 is used.

PRL= WALD | LR | BOTH
requests confidence intervals on estimated parameters. By default, the PRL option produces 95% likelihood ratio confidence limits. The coverage of the confidence interval is controlled by the ALPHA= option in the FIT statement.

START= *(parameter values . . .)*
supplies starting values for the parameter estimates. In PROC MODEL, you have several options to specify starting values for the parameters to be estimated. When more than one option is specified, the options are implemented in the following order of precedence (from highest to lowest): the START= option, the PARMS statement initialization value, the ESTDATA= option, and the PARMSDATA= option. If no options are specified for the starting value, the default value of 0.0001 is used. If the START= option specifies more than one starting value for one or more parameters, a grid search is

performed over all combinations of the values, and the best combination is used to start the iterations. For more information, see the STARTITER= option.

Options to Control the Estimation Method Used

ADJSMMV

specifies adding the variance adjustment from simulating the moments to the variance-covariance matrix of the parameter estimators. By default, no adjustment is made.

COVBEST=GLS | CROSS | FDA

specifies the variance-covariance estimator used for FIML. COVBEST=GLS selects the generalized least squares estimator. COVBEST=CROSS selects the crossproducts estimator. COVBEST=FDA selects the inverse of the finite difference approximation to the Hessian. The default is COVBEST=CROSS.

DYNAMIC

specifies dynamic estimation of ordinary differential equations. See the section "Ordinary Differential Equations" on page 1185 for more details.

FIML

specifies full information maximum likelihood estimation.

GINV=G2 | G4

specifies the type of generalized inverse to be used when computing the covariance matrix. G4 selects the Moore-Penrose generalized inverse. The default is GINV=G2.

Rather than deleting linearly related rows and columns of the covariance matrix, the Moore-Penrose generalized inverse averages the variance effects between collinear rows. When the option GINV=G4 is used, the Moore-Penrose generalized inverse is used to calculate standard errors and the covariance matrix of the parameters as well as the change vector for the optimization problem. For singular systems, a normal G2 inverse is used to determine the singular rows so that the parameters can be marked in the parameter estimates table. A G2 inverse is calculated by satisfying the first two properties of the Moore-Penrose generalized inverse; that is, $\mathbf{A}\mathbf{A}^{+}\mathbf{A} = \mathbf{A}$ and $\mathbf{A}^{+}\mathbf{A}\mathbf{A}^{+} = \mathbf{A}^{+}$. Whether or not you use a G4 inverse, if the covariance matrix is singular, the parameter estimates are not unique. Refer to Noble and Daniel (1977, pp. 337–340) for more details about generalized inverses.

GENGMMV

specify GMM variance under arbitrary weighting matrix. See the section "Estimation Methods" on page 1130 for more details.

This is the default method for GMM estimation.

GMM

specifies generalized method of moments estimation.

HCCME= 0 | 1 | 2 | 3 | NO

specifies the type of heteroscedasticity-consistent covariance matrix estimator to use for OLS, 2SLS, 3SLS, SUR, and the iterated versions of these estimation methods. The number corresponds to the

type of covariance matrix estimator to use as

$$\begin{aligned} HC_0: &\quad \hat{\epsilon}_t^2 \\ HC_1: &\quad \frac{n}{n-df}\hat{\epsilon}_t^2 \\ HC_2: &\quad \hat{\epsilon}_t^2/(1-\hat{h}_t) \\ HC_3: &\quad \hat{\epsilon}_t^2/(1-\hat{h}_t)^2 \end{aligned}$$

The default is NO.

ITGMM
specifies iterated generalized method of moments estimation.

ITOLS
specifies iterated ordinary least squares estimation. This is the same as OLS unless there are cross-equation parameter restrictions.

ITSUR
specifies iterated seemingly unrelated regression estimation

IT2SLS
specifies iterated two-stage least squares estimation. This is the same as 2SLS unless there are cross-equation parameter restrictions.

IT3SLS
specifies iterated three-stage least squares estimation.

KERNEL=(PARZEN | BART | QS, < c > , < e >)
KERNEL=PARZEN | BART | QS
specifies the kernel to be used for GMM and ITGMM. PARZEN selects the Parzen kernel, BART selects the Bartlett kernel, and QS selects the quadratic spectral kernel. $e \geq 0$ and $c \geq 0$ are used to compute the bandwidth parameter. The default is KERNEL=(PARZEN, 1, 0.2). See the section "Estimation Methods" on page 1130 for more details.

N2SLS | 2SLS
specifies nonlinear two-stage least squares estimation. This is the default when an INSTRUMENTS statement is used.

N3SLS | 3SLS
specifies nonlinear three-stage least squares estimation.

NDRAW < =*number of draws* **>**
requests the simulation method for parameter estimation where the contribution of each observation to the estimation is approximated by using *number of draws* evaluations of the model program. If *number of draws* is not specified, the default value of 10 is used.

NOOLS
NO2SLS
specifies bypassing OLS or 2SLS to get initial parameter estimates for GMM, ITGMM, or FIML. This is important for certain models that are poorly defined in OLS or 2SLS, or if good initial parameter values are already provided. Note that for GMM, the **V** matrix is created by using the initial values specified and this might not be consistently estimated.

NO3SLS
specifies not to use 3SLS automatically for FIML initial parameter starting values.

NOGENGMMV
specifies not to use GMM variance under arbitrary weighting matrix. Use GMM variance under optimal weighting matrix instead. See the section "Estimation Methods" on page 1130 for more details.

NPREOBS =*number of obs to initialize*
specifies the initial number of observations to run the simulation before the simulated values are compared to observed variables. This option is most useful in cases where the program statements involve lag operations. Use this option to avoid the effect of the starting point on the simulation.

NVDRAW =*number of draws for V matrix*
specifies H', the number of draws for V matrix. If this option is not specified, the default H' is set to 20.

OLS
specifies ordinary least squares estimation. This is the default.

SUR
specifies seemingly unrelated regression estimation.

VARDEF=N | WGT | DF | WDF
specifies the denominator to be used in computing variances and covariances, MSE, root MSE measures, and so on. VARDEF=N specifies that the number of nonmissing observations be used. VARDEF=WGT specifies that the sum of the weights be used. VARDEF=DF specifies that the number of nonmissing observations minus the model degrees of freedom (number of parameters) be used. VARDEF=WDF specifies that the sum of the weights minus the model degrees of freedom be used. The default is VARDEF=DF. For FIML estimation the VARDEF= option does not affect the calculation of the parameter covariance matrix, which is determined by the COVBEST= option.

Data Set Options

DATA=*SAS-data-set*
specifies the input data set. Values for the variables in the program are read from this data set. If the DATA= option is not specified on the FIT statement, the data set specified by the DATA= option on the PROC MODEL statement is used.

ESTDATA=*SAS-data-set*
specifies a data set whose first observation provides initial values for some or all of the parameters.

MISSING=PAIRWISE | DELETE
specifies how missing values are handled. MISSING=PAIRWISE specifies that missing values are tracked on an equation-by-equation basis. MISSING=DELETE specifies that the entire observation is omitted from the analysis when any equation has a missing predicted or actual value for the equation. The default is MISSING=DELETE.

OUT=*SAS-data-set*
names the SAS data set to contain the residuals, predicted values, or actual values from each estimation. The residual values written to the OUT= data set are defined as the $actual - predicted$, which is the negative of RESID.*variable* as defined in the section "Equation Translations" on page 1272. Only the residuals are output by default.

OUTACTUAL
writes the actual values of the endogenous variables of the estimation to the OUT= data set. This option is applicable only if the OUT= option is specified.

OUTALL
selects the OUTACTUAL, OUTERRORS, OUTLAGS, OUTPREDICT, and OUTRESID options.

OUTCOV
COVOUT
writes the covariance matrix of the estimates to the OUTEST= data set in addition to the parameter estimates. The OUTCOV option is applicable only if the OUTEST= option is also specified.

OUTEST=*SAS-data-set*
names the SAS data set to contain the parameter estimates and optionally the covariance of the estimates.

OUTLAGS
writes the observations used to start the lags to the OUT= data set. This option is applicable only if the OUT= option is specified.

OUTPREDICT
writes the predicted values to the OUT= data set. This option is applicable only if OUT= is specified.

OUTRESID
writes the residual values computed from the parameter estimates to the OUT= data set. The OUTRESID option is the default if neither OUTPREDICT nor OUTACTUAL is specified. This option is applicable only if the OUT= option is specified. If the h.var equation is specified, the residual values written to the OUT= data set are the normalized residuals, defined as $actual - predicted$, divided by the square root of the h.var value. If the WEIGHT statement is used, the residual values are calculated as $actual - predicted$ multiplied by the square root of the WEIGHT variable.

OUTS=*SAS-data-set*
names the SAS data set to contain the estimated covariance matrix of the equation errors. This is the covariance of the residuals computed from the parameter estimates.

OUTSN=*SAS-data-set*
names the SAS data set to contain the estimated normalized covariance matrix of the equation errors. This is valid for multivariate *t* distribution estimation.

OUTSUSED=*SAS-data-set*
names the SAS data set to contain the **S** matrix used in the objective function definition. The OUTSUSED= data set is the same as the OUTS= data set for the methods that iterate the **S** matrix.

OUTUNWGTRESID
writes the unweighted residual values computed from the parameter estimates to the OUT= data set. These are residuals computed as $actual - predicted$ with no accounting for the WEIGHT statement, the _WEIGHT_ variable, or any variance expressions. This option is applicable only if the OUT= option is specified.

OUTV=*SAS-data-set*
names the SAS data set to contain the estimate of the variance matrix for GMM and ITGMM.

SDATA=*SAS-data-set*
specifies a data set that provides the covariance matrix of the equation errors. The matrix read from the SDATA= data set is used for the equation covariance matrix (**S** matrix) in the estimation. (The SDATA= S matrix is used to provide only the initial estimate of **S** for the methods that iterate the **S** matrix.)

TIME=*name*
specifies the name of the time variable. This variable must be in the data set.

TYPE=*name*
specifies the estimation type to read from the SDATA= and ESTDATA= data sets. The name specified in the TYPE= option is compared to the _TYPE_ variable in the ESTDATA= and SDATA= data sets to select observations to use in constructing the covariance matrices. When the TYPE= option is omitted, the last estimation type in the data set is used. Valid values are the estimation methods used in PROC MODEL.

VDATA=*SAS-data-set*
specifies a data set that contains a variance matrix for GMM and ITGMM estimation. See the section "Output Data Sets" on page 1226 for details.

Printing Options for FIT Tasks

BREUSCH=*(variable-list)*
specifies the modified Breusch-Pagan test, where *variable-list* is a list of variables used to model the error variance.

CHOW=*obs*

CHOW=*(obs1 obs2 . . . obsn)*
prints the Chow test for break points or structural changes in a model. The argument is the number of observations in the first sample or a parenthesized list of first sample sizes. If the size of the one of the two groups in which the sample is partitioned is less than the number of parameters, then a predictive Chow test is automatically used. See the section "Chow Tests" on page 1199 for details.

COLLIN
prints collinearity diagnostics for the Jacobian crossproducts matrix (**XPX**) after the parameters have converged. Collinearity diagnostics are also automatically printed if the estimation fails to converge.

CORR
prints the correlation matrices of the residuals and parameters. Using CORR is the same as using both CORRB and CORRS.

CORRB
prints the correlation matrix of the parameter estimates.

CORRS
prints the correlation matrix of the residuals.

COV
prints the covariance matrices of the residuals and parameters. Specifying COV is the same as specifying both COVB and COVS.

COVB
prints the covariance matrix of the parameter estimates.

COVS
prints the covariance matrix of the residuals.

DW < = >
prints Durbin-Watson *d* statistics, which measure autocorrelation of the residuals. When the residual series is interrupted by missing observations, the Durbin-Watson statistic calculated is d' as suggested by Savin and White (1978). This is the usual Durbin-Watson computed by ignoring the gaps. Savin and White show that it has the same null distribution as the DW with no gaps in the series and can be used to test for autocorrelation using the standard tables. The Durbin-Watson statistic is not valid for models that contain lagged endogenous variables.

You can use the DW= option to request higher-order Durbin-Watson statistics. Since the ordinary Durbin-Watson statistic tests only for first-order autocorrelation, the Durbin-Watson statistics for higher-order autocorrelation are called *generalized Durbin-Watson* statistics.

DWPROB
prints the significance level (*p*-values) for the Durbin-Watson tests. Since the Durbin-Watson *p*-values are computationally expensive, they are not reported by default. In the Durbin-Watson test, the null hypothesis is that there is autocorrelation at a specific lag.

See the section "Generalized Durbin-Watson Tests" for limitations of the statistic in the Chapter 8, "The AUTOREG Procedure."

FSRSQ
prints the first-stage R^2 statistics for instrumental estimation methods. These R^2 statistics measure the proportion of the variance retained when the Jacobian columns associated with the parameters are projected through the instruments space.

GODFREY

GODFREY=*n*
performs Godfrey's tests for autocorrelated residuals for each equation, where *n* is the maximum autoregressive order, and specifies that Godfrey's tests be computed for lags 1 through *n*. The default number of lags is one.

HAUSMAN
performs Hausman's specification test, or *m*-statistics.

NORMAL
performs tests of normality of the model residuals.

PCHOW=*obs*

PCHOW=*(obs1 obs2 . . . obsn)*
prints the predictive Chow test for break points or structural changes in a model. The argument is the number of observations in the first sample or a parenthesized list of first sample sizes. See the section "Chow Tests" on page 1199 for details.

PRINTALL
specifies the printing options COLLIN, CORRB, CORRS, COVB, COVS, DETAILS, DW, and FSRSQ.

WHITE
specifies White's test.

Options to Control Iteration Output

Details of the output produced are discussed in the section "Iteration History" on page 1162.

I
prints the inverse of the crossproducts Jacobian matrix at each iteration.

ITALL
specifies all iteration printing-control options (I, ITDETAILS, ITPRINT, and XPX). ITALL also prints the crossproducts matrix (labeled CROSS), the parameter change vector, and the estimate of the cross-equation covariance of residuals matrix at each iteration.

ITDETAILS
prints a detailed iteration listing. This includes the ITPRINT information and additional statistics.

ITPRINT
prints the parameter estimates, objective function value, and convergence criteria at each iteration.

XPX
prints the crossproducts Jacobian matrix at each iteration.

Options to Control the Minimization Process

The following options can be helpful when you experience a convergence problem:

CONVERGE=*value1*

CONVERGE=*(value1, value2)*
specifies the convergence criteria. The convergence measure must be less than *value1* before convergence is assumed. *value2* is the convergence criterion for the **S** and **V** matrices for **S** and **V** iterated methods. *value2* defaults to *value1*. See the section "Convergence Criteria" on page 1151 for details. The default value is CONVERGE=0.001.

HESSIAN=CROSS | GLS | FDA
specifies the Hessian approximation used for FIML. HESSIAN=CROSS selects the crossproducts approximation to the Hessian, HESSIAN=GLS selects the generalized least squares approximation to the Hessian, and HESSIAN=FDA selects the finite difference approximation to the Hessian. HESSIAN=GLS is the default.

LTEBOUND=*n*
specifies the local truncation error bound for the integration. This option is ignored if no ordinary differential equations (ODEs) are specified.

EPSILON =*value*
specifies the tolerance value used to transform strict inequalities into inequalities when restrictions on parameters are imposed. By default, EPSILON=1E–8. See the section "Restrictions and Bounds on Parameters" on page 1195 for details.

MAXITER=*n*
specifies the maximum number of iterations allowed. The default is MAXITER=100.

MAXSUBITER=*n*
specifies the maximum number of subiterations allowed for an iteration. For the GAUSS method, the MAXSUBITER= option limits the number of step halvings. For the MARQUARDT method, the MAXSUBITER= option limits the number of times λ can be increased. The default is MAXSUBITER=30. See the section "Minimization Methods" on page 1150 for details.

METHOD=GAUSS | MARQUARDT
specifies the iterative minimization method to use. METHOD=GAUSS specifies the Gauss-Newton method, and METHOD=MARQUARDT specifies the Marquardt-Levenberg method. The default is METHOD=GAUSS. If the default GAUSS method fails to converge, the procedure switches to the MARQUARDT method. See the section "Minimization Methods" on page 1150 for details.

MINTIMESTEP=*n*
specifies the smallest allowed time step to be used in the integration. This option is ignored if no ODEs are specified.

NESTIT
changes the way the iterations are performed for estimation methods that iterate the estimate of the equation covariance (**S** matrix). The NESTIT option is relevant only for the methods that iterate the estimate of the covariance matrix (ITGMM, ITOLS, ITSUR, IT2SLS, and IT3SLS). See the section "Details on the Covariance of Equation Errors" on page 1148 for an explanation of NESTIT.

SINGULAR=*value*
specifies the smallest pivot value allowed. The default 1.0E–12.

STARTITER=*n*
specifies the number of minimization iterations to perform at each grid point. The default is STARTITER=0, which implies that no minimization is performed at the grid points. See the section "Using the STARTITER Option" on page 1157 for more details.

Other Options

Other options that can be used on the FIT statement include the following that list and analyze the model: BLOCK, GRAPH, LIST, LISTCODE, LISTDEP, LISTDER, and XREF. The following printing control options are also available: DETAILS, FLOW, INTGPRINT, MAXERRORS=, NOPRINT, PRINTALL, and TRACE. For complete descriptions of these options, see the discussion of the PROC MODEL statement options earlier in this chapter.

ID Statement

ID *variables* **;**

The ID statement specifies variables to identify observations in error messages or other listings and in the OUT= data set. The ID variables are normally SAS date or datetime variables. If more than one ID variable is used, the first variable is used to identify the observations; the remaining variables are added to the OUT= data set.

INCLUDE Statement

INCLUDE *model-names . . .* **;**

The INCLUDE statement reads model files and inserts their contents into the current model. However, instead of replacing the current model as the RESET MODEL= option does, the contents of included model files are inserted into the model program at the position that the INCLUDE statement appears.

INSTRUMENTS Statement

INSTRUMENTS *variables* < *_EXOG_* > **;**

INSTRUMENTS < *variables-list* > < *_EXOG_* > < *EXCLUDE =(parameters)* > < */ options* > **;**

INSTRUMENTS *(equation, variables) (equation, variables) . . .* **;**

The INSTRUMENTS statement specifies the instrumental variables to be used in the N2SLS, N3SLS, IT2SLS, IT3SLS, GMM, and ITGMM estimation methods.

There are three ways of specifying the INSTRUMENTS statement. The first form of the INSTRUMENTS statement is declared before a FIT statement and defines the default instruments list. The items specified as instruments can be variables or the special keyword _EXOG_. The keyword _EXOG_ indicates that all the model variables declared EXOGENOUS are to be added to the instruments list. If a single INSTRUMENTS statement of the first form is declared before multiple FIT statements, then it serves as the default instruments list for each of the FIT statements. However, if any of these FIT statements are followed by separate INSTRUMENTS statement, then the latter take precedence over the default list. Hence, in the case of multiple FIT statements, the INSTRUMENTS statement for a particular FIT statement is written below the FIT statement if instruments other than the default are required. For a single FIT statement, you can declare the INSTRUMENTS statement of the first form either preceding or following the FIT statement.

The second form of the INSTRUMENTS statement is used only after the FIT statement and before the next RUN statement. The items specified as instruments for the second form can be variables, names of parameters to be estimated, or the special keyword _EXOG_. If you specify the name of a parameter in the instruments list, the partial derivatives of the equations with respect to the parameter (that is, the columns of the Jacobian matrix associated with the parameter) are used as instruments. The parameter itself is not used as an instrument. These partial derivatives should not depend on any of the parameters to be estimated. Only the names of parameters to be estimated can be specified.

Note that an INSTRUMENTS statement of only the first form declared before multiple FIT statements serves as the default instruments list. Hence, in the cases of multiple as well as single FIT statements, you can declare the second form of INSTRUMENTS statements only following the FIT statements.

In the case where a FIT statement is preceded by an INSTRUMENTS statement of the second form in error and not followed by any INSTRUMENTS statement, then the default list is used. This default list is given

by the INSTRUMENTS statement of the first form as explained above. If such a list is not declared, all the model variables declared EXOGENOUS comprise the default.

A third form of the INSTRUMENTS statement is used to specify instruments for each equation. No explicit intercept is added, parameters cannot be specified to represent instruments, and the _EXOG_ keyword is not allowed. Equations not explicitly assigned instruments use all the instruments specified for the other equations as well as instruments not assigned specific equations. In the following statements, z1, z2, and z3 are instruments used with equation y1, and z2, z3, and z4 are instruments used with equation y2.

```
proc model data=data_sim;
   exogenous x1 x2;
   parms a b c d e f;

   y1 =a*x1**2 + b*x2**2 + c*x1*x2   ;
   y2 =d*x1**2 + e*x2**2 + f*x1*x2**2;

   fit y1 y2 / 3sls ;
   instruments (y1, z1 z2 z3) (y2,z2 z3 z4);
run;
```

EXCLUDE=*(parameters)*

specifies that the derivatives of the equations with respect to all of the parameters to be estimated (except the parameters listed in the EXCLUDE list) be used as instruments, in addition to the other instruments specified. If you use the EXCLUDE= option, you should be sure that the derivatives with respect to the nonexcluded parameters in the estimation are independent of the endogenous variables and not functions of the parameters estimated.

The following options can be specified on the INSTRUMENTS statement following a slash (/):

NOINTERCEPT

NOINT

excludes the constant of 1.0 (intercept) from the instruments list. An intercept is included as an instrument while using the first or second form of the INSTRUMENTS statement unless NOINTERCEPT is specified.

When a FIT statement specifies an instrumental variables estimation method and no INSTRUMENTS statement accompanies the FIT statement, the default instruments are used. If no default instruments list has been specified, all the model variables declared EXOGENOUS are used as instruments. See the section "Choice of Instruments" on page 1202 for more details.

INTONLY

specifies that only the intercept be used as an instrument. This option is used for GMM estimation where the moments have been specified explicitly.

LABEL Statement

LABEL *variable='label' . . .* **;**

The LABEL statement specifies a label of up to 255 characters for parameters and other variables used in the model program. Labels are used to identify parts of the printout of FIT and SOLVE tasks. The labels are displayed in the output if the LINESIZE= option is large enough.

MOMENT Statement

MOMENT *variables = moment specification* **;**

In many scenarios, endogenous variables are observed from data. From the models, you can simulate these endogenous variables based on a fixed set of parameters. The goal of simulated method of moments (SMM) is to find a set of parameters such that the moments of the simulated data match the moments of the observed variables. If there are many moments to match, the code might be tedious. The following MOMENT statement provides a way to generate some commonly used moments automatically. Multiple MOMENT statements can be used.

variables can be one or more endogenous variables.

moment specification can have the following four types:

- (*number list*) specifies that the endogenous variable is raised to the power specified by each number in *number list*. For example,

  ```
  moment y = (2 3);
  ```

  adds the following two equations to be estimated:

  ```
  eq._moment_1 = y**2 - pred.y**2;
  eq._moment_2 = y**3 - pred.y**3;
  ```

- ABS(*number list*) specifies that the absolute value of the endogenous variable is raised to the power specified by each number in *number list*. For example,

  ```
  moment y = ABS(3);
  ```

  adds the following equation to be estimated:

  ```
  eq._moment_2 = abs(y)**3 - abs(pred.y)**3;
  ```

- LAG*num* (*number list*) specifies that the endogenous variable is multiplied by the *num* th lag of the endogenous variable, and this product is raised to the power specified by each number in *number list*. For example,

  ```
  moment y = LAG4(3);
  ```

  adds the following equation to be estimated:

  ```
  eq._moment_3 = (y*lag4(y))**3 - (pred.y*lag4(pred.y))**3;
  ```

- ABS_LAG*num* (*number list*) specifies that the endogenous variable is multiplied by the *num* th lag of the endogenous variable, and the absolute value of this product is raised to the power specified by each number in *number list*. For example,

  ```
  moment y = ABS_LAG4(3);
  ```

adds the following equation to be estimated:

```
eq._moment_4 = abs(y*lag4(y))**3 - abs(pred.y*lag4(pred.y))**3;
```

The following PROC MODEL statements use the MOMENT statement to generate 24 moments and fit these moments using SMM.

```
proc model data=_tmpdata list;
   parms a b .5 s 1;
   instrument _exog_ / intonly;

   u = rannor( 10091 );
   z = rannor( 97631 );

   lsigmasq = xlag(sigmasq,exp(a));

   lnsigmasq = a + b * log(lsigmasq) + s * u;
   sigmasq = exp( lnsigmasq );

   y = sqrt(sigmasq) * z;

   moment y = (2 4) abs(1 3) abs_lag1(1 2) abs_lag2(1 2);
   moment y = abs_lag3(1 2) abs_lag4(1 2)
              abs_lag5(1 2) abs_lag6(1 2)
              abs_lag7(1 2) abs_lag8(1 2)
              abs_lag9(1 2) abs_lag10(1 2);

   fit y  / gmm npreobs=20 ndraw=10;
   bound s > 0, 1>b>0;

run;
```

OUTVARS Statement

OUTVARS *variables* **;**

The OUTVARS statement specifies additional variables defined in the model program to be output to the OUT= data sets. The OUTVARS statement is not needed unless the variables to be added to the output data set are not referred to by the model, or unless you want to include parameters or other special variables in the OUT= data set. The OUTVARS statement includes additional variables, whereas the KEEP statement excludes variables.

PARAMETERS Statement

PARAMETERS *variable* < *value* > < *variable* < *value* > > . . . **;**

The PARAMETERS statement declares the parameters of a model and optionally sets their initial values. Valid abbreviations are PARMS and PARM.

Each parameter has a single value associated with it, which is the same for all observations. Lagging is not relevant for parameters. If a value is not specified in the PARMS statement (or by the PARMS= option of a FIT statement), the value defaults to 0.0001 for FIT tasks and to a missing value for SOLVE tasks.

Programming Statements

To define the model, you can use most of the programming statements that are allowed in the SAS DATA step. See the *SAS Language Reference: Dictionary* for more information.

RANGE Statement

RANGE *variable* < = *first* > < *TO last* > **;**

The RANGE statement specifies the range of observations to be read from the DATA= data set. For FIT tasks, the RANGE statement controls the period of fit for the estimation. For SOLVE tasks, the RANGE statement controls the simulation period or forecast horizon.

The RANGE variable must be a numeric variable in the DATA= data set that identifies the observations, and the data set must be sorted by the RANGE variable. The first observation in the range is identified by *first*, and the last observation is identified by *last*.

PROC MODEL uses the first *l* observations prior to *first* to initialize the lags, where *l* is the maximum number of lags needed to evaluate any of the equations to be fit or solved, or the maximum number of lags needed to compute any of the instruments when an instrumental variables estimation method is used. There should be at least *l* observations in the data set before *first*. If *last* is not specified, all the nonmissing observations starting with *first* are used.

If *first* is omitted, the first *l* observations are used to initialize the lags, and the rest of the data, until *last*, is used. If a RANGE statement is used but both *first* and *last* are omitted, the RANGE statement variable is used to report the range of observations processed.

The RANGE variable should be nonmissing for all observations. Observations that contain missing RANGE values are deleted.

The following are examples of RANGE statements:

```
range year = 1971 to 1988;               /* yearly  data  */
range date = '1feb73'd to '1nov82'd;     /* monthly data  */
range time = 60.5;                       /* time in years */
range year to 1977;         /* use all years through 1977 */
range date; /* use values of date to report period-of-fit */
```

If no RANGE statements follow multiple FIT statements and if a single RANGE statement is declared before all the FIT statements, estimation in each of the multiple FIT statements is based on the data specified in the single RANGE statement. A single RANGE statement that follows multiple FIT statements affects only the fit immediately preceding it.

If the FIT statement is both followed by and preceded by RANGE statements, the following RANGE statement takes precedence over the preceding RANGE statement.

In the case where a range of data is to be used for a particular SOLVE task, the RANGE statement should be specified following the SOLVE statement in the case of either single or multiple SOLVE statements.

RESET Statement

RESET *options* **;**

All the options of the PROC MODEL statement can be reset by the RESET statement. In addition, the RESET statement supports one additional option:

PURGE
deletes the current model so that a new model can be defined.

When the MODEL= option is used in the RESET statement, the current model is deleted before the new model is read.

RESTRICT Statement

RESTRICT *restriction1* **<** *, restriction2 . . .* **>** **;**

The RESTRICT statement is used to impose linear and nonlinear restrictions either on the parameters in an estimation or on the solution variables that are specified in a solve operation.

Each *restriction* is written as an optional name, followed by an expression, followed by an equality operator (=) or an inequality operator (<, >, <=, >=), followed by a second expression:

< *"name"* **>** *expression operator expression*

The optional *"name"* is a string used to identify the restriction. The *operator* can be =, <, >, <= , or >=. The *operator* and second *expression* are optional. When they are omitted, the default *operator* is > and the default second *expression* is 0.

Each RESTRICT statement is associated with the preceding FIT statement or SOLVE statement. When there is no preceding FIT or SOLVE statement, the RESTRICT statement is associated with the following FIT or SOLVE statement. You can specify any number of RESTRICT statements.

Parameter Estimates

Expressions in RESTRICT statements that apply to the parameters estimated by a FIT statement can be composed of parameter names, arithmetic operators, functions, and constants. Comparison operators (such as = or <) and logical operators (such as &) cannot be used in RESTRICT statement expressions. Parameters that are named in restriction expressions must be among the parameters estimated by the associated FIT statement. Expressions can refer to variables defined in the program.

The restriction expressions can be linear or nonlinear functions of the parameters.

The optional *"name"* is a string used to identify the restriction in the printed output and in the OUTEST= data set.

The following example shows how to use the RESTRICT statement:

```
proc model data=one;
   endogenous y1 y2;
   exogenous x1 x2;
   parms a b c;
   restrict b*(b+c) <= a;

   eq.one = -y1/c + a/x2 + b * x1**2 + c * x2**2;
   eq.two = -y2 * y1 + b * x2**2 - c/(2 * x1);

   fit one two / fiml;
run;
```

Solution Variables

Expressions in RESTRICT statements that apply to the solution variables in a SOLVE statement can be composed of any variables in the model. Unlike restriction expressions that are used in parameter estimation, exogenous model variables can be used in restriction expressions that involve solution variables because each observation is solved independently in a SOLVE statement. To include constraints that are imposed by RESTRICT inequalities in a solution, you must specify the OPTIMIZE option in the SOLVE statement.

The following example illustrates how multiple solutions to a nonlinear system of equations can be found by using a RESTRICT expression that depends on exogenous variables. Two of the four possible solutions are presented in Figure 19.22.

```
data d;
   do i = 0 to 1;
      date=i;
      if i = 0 then r = -1;
      else          r = +1;
      output;
   end;
run;


proc model data=d ;
   endo x y;

   eq.a = x*x - 4;
   eq.b = y*y - 9;

   restrict x*y*r > 1;

   solve /  optimize out=o outall;
quit;

proc print data = o; run;
```

Figure 19.22 Listing of OUT= Data Set Created by a Nonlinear Restriction

Obs	_TYPE_	_MODE_	_ERRORS_	_OBJVAL_	x	y	r
1	ACTUAL	SIMULATE	0	0	2	-3	-1
2	PREDICT	SIMULATE	0	0	2	-3	-1
3	RESIDUAL	SIMULATE	0	0	.	.	-1
4	ERROR	SIMULATE	0	0	.	.	-1
5	VIOL	SIMULATE	0	0	.	.	-1
6	ACTUAL	SIMULATE	0	0	-2	-3	1
7	PREDICT	SIMULATE	0	0	-2	-3	1
8	RESIDUAL	SIMULATE	0	0	.	.	1
9	ERROR	SIMULATE	0	0	.	.	1
10	VIOL	SIMULATE	0	0	.	.	1

SOLVE Statement

SOLVE *variables* < *SATISFY= equations* > < */options* > **;**

The SOLVE statement specifies that the model be simulated or forecast for input data values and, optionally, selects the variables to be solved. If the list of variables is omitted, all of the model variables declared ENDOGENOUS are solved. If no model variables are declared ENDOGENOUS, then all model variables are solved.

The following specification can be used in the SOLVE statement:

SATISFY=*equation*

SATISFY=*(equations)*

specifies a subset of the model equations that the solution values are to satisfy. If the SATISFY= option is not used, the solution is computed to satisfy all the model equations. Note that the number of equations must equal the number of variables solved.

Data Set Options

DATA=*SAS-data-set*

names the input data set. The model is solved for each observation read from the DATA= data set. If the DATA= option is not specified on the SOLVE statement, the data set specified by the DATA= option in the PROC MODEL statement is used.

ESTDATA=*SAS-data-set*

names a data set whose first observation provides values for some or all of the parameters and whose additional observations (if any) give the covariance matrix of the parameter estimates. The covariance matrix read from the ESTDATA= data set is used to generate multivariate normal pseudo-random shocks to the model parameters when the RANDOM= option requests Monte Carlo simulation.

OUT=*SAS-data-set*

outputs the predicted (solution) values, residual values, actual values, or equation errors from the solution to a data set. The residual values are the $actual - predicted$ values, which is the negative of RESID.*variable* as defined in the section "Equation Translations" on page 1272. Only the solution values are output by default.

OUTACTUAL
outputs the actual values of the solved variables read from the input data set to the OUT= data set. This option is applicable only if the OUT= option is specified.

OUTALL
specifies the OUTACTUAL, OUTERRORS, OUTLAGS, OUTPREDICT, and OUTRESID options.

OUTERRORS
writes the equation errors to the OUT= data set. These values are normally very close to 0 when a simultaneous solution is computed; they can be used to double-check the accuracy of the solution process. This option applies only if the OUT= option is specified.

OUTLAGS
writes the observations that are used to start the lags to the OUT= data set. This option applies only if the OUT= option is specified.

OUTOBJVALS
writes the objective function value to the OBJVALS variable in the OUT= data set. The objective function value is computed only when the OPTIMIZE solution method is specified. This value is close to 0 when an unbounded simultaneous solution is computed and can be greater than 0 when bounds are active in the solution. This option applies only if the OUT= option is specified.

OUTPREDICT
writes the solution values to the OUT= data set. This option applies only if the OUT= option is specified.

The OUTPREDICT option is the default unless one of the other output options is specified.

OUTRESID
writes the residual values that are computed as the $actual - predicted$ values and is not the same as the RESID.*variable* values. This option applies only if the OUT= option is specified.

OUTVIOLATIONS
writes the equation violations to the OUT= data set. The equation violations are computed only when the OPTIMIZE solution method is specified. The violations provide information about how much each equation contributes to the objective function value when bounds are active in the solution. This option applies only if the OUT= option is specified.

PARMSDATA=*SAS-data-set*
specifies a data set that contains the parameter estimates. See the section "Input Data Sets" on page 1221 for more details.

RESIDDATA=*SAS-data-set*
specifies a data set that contains the residuals to be used in the empirical distribution. This data set can be created using the OUT= option in the FIT statement.

SDATA=*SAS-data-set*
specifies a data set that provides the covariance matrix of the equation errors. The covariance matrix that is read from the SDATA= data set is used to generate multivariate normal pseudo-random shocks to the equations when the RANDOM= option requests Monte Carlo simulation.

TIME=*name*
specifies the name of the time variable. This variable must be in the data set.

TYPE=*name*
specifies the estimation type. The name that is specified in the TYPE= option is compared to the _TYPE_ variable in the ESTDATA= and SDATA= data sets to select observations to use in constructing the covariance matrices. When TYPE= is omitted, the last estimation type in the data set is used.

Solution Mode Options: Lag Processing

DYNAMIC
specifies a dynamic solution. In the dynamic solution mode, solved values are used by the lagging functions. DYNAMIC is the default.

NAHEAD=*n*
specifies a simulation of *n*-period-ahead dynamic forecasting. The NAHEAD= option is used to simulate the process of using the model to produce successive forecasts to a fixed forecast horizon, in which each forecast uses the historical data available at the time the forecast is made.

Note that NAHEAD=1 produces a static (one-step-ahead) solution. NAHEAD=2 produces a solution that uses one-step-ahead solutions for the first lag (LAG1 functions return static predicted values) and actual values for longer lags. NAHEAD=3 produces a solution that uses NAHEAD=2 solutions for the first lags, NAHEAD=1 solutions for the second lags, and actual values for longer lags. In general, NAHEAD=*n* solutions use NAHEAD=*n*–1 solutions for LAG1, NAHEAD=*n*–2 solutions for LAG2, and so forth.

START=*s*
specifies static solutions until the *s*th observation and then changes to dynamic solutions. If the START=*s* option is specified, the first observation in the range in which LAG*n* delivers solved predicted values is *s*+*n*, while LAG*n* returns actual values for earlier observations.

STATIC
specifies a static solution. In static solution mode, actual values of the solved variables from the input data set are used by the lagging functions.

Solution Mode Options: Use of Available Data

FORECAST
specifies that the actual value of a solved variable is used as the solution value (instead of the predicted value from the model equations) whenever nonmissing data are available in the input data set. That is, in FORECAST mode, PROC MODEL solves only for those variables that are missing in the input data set.

SIMULATE
specifies that PROC MODEL always solves for all solution variables as a function of the input values of the other variables, even when actual data for some of the solution variables are available in the input data set. SIMULATE is the default.

Solution Mode Options: Numerical Solution Method

JACOBI
computes a simultaneous solution using a Jacobi iteration.

NEWTON
computes a simultaneous solution by using Newton's method. When the NEWTON option is selected, the analytic derivatives of the equation errors with respect to the solution variables are computed, and memory-efficient sparse matrix techniques are used for factoring the Jacobian matrix.

The NEWTON option can be used to solve both normalized-form and general-form equations and can compute goal-seeking solutions. NEWTON is the default.

OPTIMIZE
computes a simultaneous solution by minimizing a norm of the equation errors with respect to the solution variables. The OPTIMIZE method obeys constraints on the solution variables that are imposed by the BOUNDS and RESTRICT statements.

SEIDEL
computes a simultaneous solution by using a Gauss-Seidel method.

SINGLE
ONEPASS
specifies a single-equation (nonsimultaneous) solution. The model is executed once to compute predicted values for the variables from the actual values of the other endogenous variables. The SINGLE option can be used only for normalized-form equations and cannot be used for goal-seeking solutions.

For more information about these options, see the section "Solution Modes" on page 1232.

Monte Carlo Simulation Options

COPULA=(*copula-options*)
specifies the copula to be used in the simulation. You can specify the following *copula-options*:

- CLAYTON(θ), where θ is the Clayton copula parameter
- FRANK(θ), where θ is the Frank copula parameter
- GUMBEL(θ), where θ is the Gumbel copula parameter
- NORMAL
- NORMALMIX(*n*, $p_1 \ldots p_n$, $v_1 \ldots v_n$), where p_i are the probabilities and v_i are the variances
- T(*df*) < ASYM >, where *df* is the degrees-of-freedom parameter

The normal (Gaussian) copula is the default. The copula applies to covariance of equation errors.

PSEUDO=DEFAULT | TWISTER
specifies which pseudo-number generator to use in generating draws for Monte Carlo simulation. The two pseudo-random number generators that are supported by the MODEL procedure are a default congruential generator that has period $2^{31} - 1$ and a Mersenne-Twister pseudo-random number generator that has an extraordinarily long period $2^{19937} - 1$.

QUASI=NONE|SOBOL|FAURE
specifies a pseudo- or quasi-random number generator. Two quasi-random number generators are supported by the MODEL procedure: the Sobol sequence (QUASI=SOBOL) and the Faure sequence (QUASI=FAURE). The default is QUASI=NONE, which is the pseudo-random number generator.

RANDOM=*n*
repeats the solution *n* times for each BY group, with different random perturbations of the equation errors if the SDATA= option is specified; with different random perturbations of the parameters if the ESTDATA= option is specified and the ESTDATA= data set contains a parameter covariance matrix; and with different values returned from the random number generator functions, if any are used in the model program. If RANDOM=0, the random number generator functions always return zero. See the section "Monte Carlo Simulation" on page 1235 for details. The default is RANDOM=0.

SEED=*n*
specifies an integer to use as the seed in generating pseudo-random numbers to shock the parameters and equations when the ESTDATA= or SDATA= option is specified. If *n* is negative or 0, the time of day from the computer's clock is used as the seed. The SEED= option is relevant only if the RANDOM= option is specified. The default is SEED=0.

WISHART=*df*
specifies that a Wishart distribution with degrees of freedom *df* be used in place of the normal error covariance matrix. This option is used to model the variance of the error covariance matrix when Monte Carlo simulation is selected.

Options for Controlling the Numerical Solution Process

The following options are useful when you have difficulty converging to the simultaneous solution.

CONVERGE=*value*
specifies the convergence criterion for the simultaneous solution. Convergence of the solution is judged by comparing the CONVERGE= value to the maximum over the equations of

$$\frac{|\epsilon_i|}{|y_i| + 1E-6}$$

if they are computable, otherwise

$$|\epsilon_i|$$

where ϵ_i represents the equation error and y_i represents the solution variable that corresponds to the *i*th equation for normalized-form equations. The default is CONVERGE=1E–8.

MAXITER=*n*
specifies the maximum number of iterations allowed for computing the simultaneous solution for any observation. The default is MAXITER=50.

MAXSUBITER=*n*
specifies the maximum number of damping subiterations that are performed in solving a nonlinear system when using the NEWTON solution method. Damping is disabled by setting MAXSUBITER=0. The default is MAXSUBITER=10.

Printing Options

INTGPRINT
prints between data points integration values for the DERT. variables and the auxiliary variables. If you specify the DETAILS option, the integrated derivative variables are printed as well.

ITPRINT
prints the solution approximation and equation errors at each iteration for each observation. This option can produce voluminous output.

PRINTALL
specifies the printing control options DETAILS, ITPRINT, SOLVEPRINT, STATS, and THEIL.

SOLVEPRINT
prints the solution values and residuals at each observation.

STATS
prints various summary statistics for the solution values.

THEIL
prints tables of Theil inequality coefficients and Theil relative change forecast error measures for the solution values. See the section "Summary Statistics" on page 1251 for more information.

Other Options

Other options that can be used on the SOLVE statement include the following that list and analyze the model: BLOCK, GRAPH, LIST, LISTCODE, LISTDEP, LISTDER, and XREF. The LTEBOUND= and MINTIMESTEP= options can be used to control the integration process. The following printing-control options are also available: DETAILS, FLOW, MAXERRORS=, NOPRINT, and TRACE. For complete descriptions of these options, see the PROC MODEL and FIT statement options described earlier in this chapter.

TEST Statement

TEST < *"name"*> *test1* < *, test2 . . .* > < *,/ options* > **;**

The TEST statement performs tests of nonlinear hypotheses on the model parameters.

Each TEST statement applies to the parameters estimated by one FIT statement. TEST statements that appear before or after the first FIT statement are associated with the first FIT statement. Subsequent TEST statements are associated with the FIT statement that precedes them. TEST statements that are separated from a FIT statement by an intervening RUN, SOLVE, or RESET statement are ignored. You can specify any number of TEST statements.

If you specify options on the TEST statement, a comma is required before the "/" character that separates the test expressions from the options, because the "/" character can also be used within test expressions to indicate division.

The label lengths for tests and estimate statements are 256 characters. If the labels exceed this length, the label is truncated to 256 characters with a note printed to the log.

Each test is written as an expression optionally followed by an equal sign (=) and a second expression:

< expression > < = expression >

Test expressions can be composed of parameter names, arithmetic operators, functions, and constants. Comparison operators (such as =) and logical operators (such as &) cannot be used in TEST statement expressions. Parameters named in test expressions must be among the parameters estimated by the associated FIT statement.

If you specify only one expression in a test, that expression is tested against zero. For example, the following two TEST statements are equivalent:

```
test a + b;

test a + b = 0;
```

When you specify multiple tests in the same TEST statement, a joint test is performed. For example, the following TEST statement tests the joint hypothesis that both A and B are equal to zero.

```
test a, b;
```

To perform separate tests rather than a joint test, use separate TEST statements. For example, the following TEST statements test the two separate hypotheses that A is equal to zero and that B is equal to zero.

```
test a;
test b;
```

You can use the following options in the TEST statement.

WALD
specifies that a Wald test be computed. By default, the Wald test is computed.

LM
RAO
LAGRANGE
specifies that a Lagrange multiplier test be computed.

LR
LIKE
specifies that a likelihood ratio test be computed.

ALL
requests all three types of tests.

OUT=*SAS-data-set*
specifies the name of an output *SAS data set* that contains the test results. The format of the OUT= data set that is produced by the TEST statement is similar to that of the OUTEST= data set produced by the FIT statement.

VAR Statement

VAR *variables* < *initial-values* > . . . **;**

The VAR statement declares model variables and optionally provides initial values for the lags of the variables. See the section "Lag Logic" on page 1277 for more information.

VARGROUP Statement

VARGROUP *label*=*variable*. . . **;**

The VARGROUP statement applies a group label to the specified list of variables in the model program. Variable groups are used to identify sets of related solve variables. The variable groups can be used by the ANALYZEDEP= option in a subsequent SOLVE statement to help specify and understand the role of groups of solve variables in a SOLVE step. If a variable appears in more than one VARGROUP statement, the label that is specified in the last VARGROUP statement is applied to that variable.

WEIGHT Statement

WEIGHT *variable* **;**

The WEIGHT statement specifies a variable to supply weighting values to use for each observation in estimating parameters.

If the weight of an observation is nonpositive, that observation is not used for the estimation. *variable* must be a numeric variable in the input data set.

An alternative weighting method is to use an assignment statement to give values to the special variable _WEIGHT_. The _WEIGHT_ variable must not depend on the parameters being estimated. If both weighting specifications are given, the weights are multiplied together.

Details: Estimation by the MODEL Procedure

Estimation Methods

Consider the general nonlinear model:

$$\begin{aligned} \boldsymbol{\epsilon}_t &= \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \\ \mathbf{z}_t &= Z(\mathbf{x}_t) \end{aligned}$$

where $\mathbf{q} \in R^g$ is a real vector valued function of $\mathbf{y}_t \in R^g$, $\mathbf{x}_t \in R^l$, $\boldsymbol{\theta} \in R^p$, where g is the number of equations, l is the number of exogenous variables (lagged endogenous variables are considered exogenous here), p is

the number of parameters, and t ranges from 1 to n. $\mathbf{z}_t \in R^k$ is a vector of instruments. $\boldsymbol{\epsilon}_t$ is an unobservable disturbance vector with the following properties:

$$\begin{aligned} E(\boldsymbol{\epsilon}_t) &= 0 \\ E(\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_t^{'}) &= \boldsymbol{\Sigma} \end{aligned}$$

All of the methods implemented in PROC MODEL aim to minimize an *objective function*. The following table summarizes the objective functions that define the estimators and the corresponding estimator of the covariance of the parameter estimates for each method.

Table 19.2 Summary of PROC MODEL Estimation Methods

Method	Instruments	Objective Function	Covariance of θ
OLS	no	$\mathbf{r}'\mathbf{r}/n$	$(\mathbf{X}'(\mathrm{diag}(\mathbf{S})^{-1}\otimes\mathbf{I})\mathbf{X})^{-1}$
ITOLS	no	$\mathbf{r}'(\mathrm{diag}(\mathbf{S})^{-1}\otimes\mathbf{I})\mathbf{r}/n$	$(\mathbf{X}'(\mathrm{diag}(\mathbf{S})^{-1}\otimes\mathbf{I})\mathbf{X})^{-1}$
SUR	no	$\mathbf{r}'(\mathbf{S}_{\mathrm{OLS}}^{-1}\otimes\mathbf{I})\mathbf{r}/n$	$(\mathbf{X}'(\mathbf{S}^{-1}\otimes\mathbf{I})\mathbf{X})^{-1}$
ITSUR	no	$\mathbf{r}'(\mathbf{S}^{-1}\otimes\mathbf{I})\mathbf{r}/n$	$(\mathbf{X}'(\mathbf{S}^{-1}\otimes\mathbf{I})\mathbf{X})^{-1}$
N2SLS	yes	$\mathbf{r}'(\mathbf{I}\otimes\mathbf{W})\mathbf{r}/n$	$(\mathbf{X}'(\mathrm{diag}(\mathbf{S})^{-1}\otimes\mathbf{W})\mathbf{X})^{-1}$
IT2SLS	yes	$\mathbf{r}'(\mathrm{diag}(\mathbf{S})^{-1}\otimes\mathbf{W})\mathbf{r}/n$	$(\mathbf{X}'(\mathrm{diag}(\mathbf{S})^{-1}\otimes\mathbf{W})\mathbf{X})^{-1}$
N3SLS	yes	$\mathbf{r}'(\mathbf{S}_{\mathrm{N2SLS}}^{-1}\otimes\mathbf{W})\mathbf{r}/n$	$(\mathbf{X}'(\mathbf{S}^{-1}\otimes\mathbf{W})\mathbf{X})^{-1}$
IT3SLS	yes	$\mathbf{r}'(\mathbf{S}^{-1}\otimes\mathbf{W})\mathbf{r}/n$	$(\mathbf{X}'(\mathbf{S}^{-1}\otimes\mathbf{W})\mathbf{X})^{-1}$
GMM	yes	$[n\mathbf{m}_n(\theta)]'\hat{\mathbf{V}}_{\mathrm{N2SLS}}^{-1}[n\mathbf{m}_n(\theta)]/n$	$[(\mathbf{YX})'\hat{\mathbf{V}}^{-1}(\mathbf{YX})]^{-1}$
ITGMM	yes	$[n\mathbf{m}_n(\theta)]'\hat{\mathbf{V}}^{-1}[n\mathbf{m}_n(\theta)]/n$	$[(\mathbf{YX})'\hat{\mathbf{V}}^{-1}(\mathbf{YX})]^{-1}$
FIML	no	$constant + \frac{n}{2}\ln(\det(\mathbf{S}))$ $-\sum_1^n \ln\lvert(\mathbf{J}_t)\rvert$	$[\hat{\mathbf{Z}}'(\mathbf{S}^{-1}\otimes\mathbf{I})\hat{\mathbf{Z}}]^{-1}$

The column labeled "Instruments" identifies the estimation methods that require instruments. The variables used in this table and the remainder of this chapter are defined as follows:

n = is the number of nonmissing observations.

g = is the number of equations.

k = is the number of instrumental variables.

$\mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_g \end{bmatrix}$ is the $ng \times 1$ vector of residuals for the g equations stacked together.

$\mathbf{r}_i = \begin{bmatrix} q_i(\mathbf{y}_1, \mathbf{x}_1, \boldsymbol{\theta}) \\ q_i(\mathbf{y}_2, \mathbf{x}_2, \boldsymbol{\theta}) \\ \vdots \\ q_i(\mathbf{y}_n, \mathbf{x}_n, \boldsymbol{\theta}) \end{bmatrix}$ is the $n \times 1$ column vector of residuals for the i th equation.

S is a $g \times g$ matrix that estimates $\boldsymbol{\Sigma}$, the covariances of the errors across equations (referred to as the **S** matrix).

X is an $ng \times p$ matrix of partial derivatives of the residual with respect to the parameters.

W is an $n \times n$ matrix, $\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'$.

$\mathbf{Z}$	is an $n \times k$ matrix of instruments.
$\mathbf{Y}$	is a $gk \times ng$ matrix of instruments. $\mathbf{Y} = \mathbf{I}_g \otimes \mathbf{Z}'$.
$\hat{\mathbf{Z}}$	$\hat{\mathbf{Z}} = (\hat{Z}_1, \hat{Z}_2, \ldots, \hat{Z}_p)$ is an $ng \times p$ matrix. $\hat{Z}_i$ is a $ng \times 1$ column vector obtained from stacking the columns of $$\mathbf{U}\frac{1}{n}\sum_{t=1}^{n}\left(\frac{\partial \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})'}{\partial y_t}\right)^{-1}\frac{\partial^2 \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})'}{\partial y_t \partial \theta_i} - \mathbf{Q}_i$$
$\mathbf{U}$	is an $n \times g$ matrix of residual errors. $\mathbf{U} = \boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \ldots, \boldsymbol{\epsilon}_n{}'$.
$\mathbf{Q}$	is the $n \times g$ matrix $\mathbf{q}(\mathbf{y}_1, \mathbf{x}_1, \boldsymbol{\theta}), \mathbf{q}(\mathbf{y}_2, \mathbf{x}_2, \boldsymbol{\theta}), \ldots, \mathbf{q}(\mathbf{y}_n, \mathbf{x}_n, \boldsymbol{\theta})$.
$\mathbf{Q}_i$	is an $n \times g$ matrix $\frac{\partial \mathbf{Q}}{\partial \theta_i}$.
$\mathbf{I}$	is an $n \times n$ identity matrix.
$\mathbf{J}_t$	is $\frac{\partial \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})}{\partial \mathbf{y}_t'}$, which is a $g \times g$ Jacobian matrix.
$\mathbf{m}_n$	is first moment of the crossproduct $\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \otimes \mathbf{z}_t$, $m_n = \frac{1}{n}\sum_{t=1}^{n} \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \otimes \mathbf{z}_t$
$\mathbf{z}_t$	is a k column vector of instruments for observation t. $\mathbf{z}_t'$ is also the tth row of $\mathbf{Z}$.
$\hat{\mathbf{V}}$	is the $gk \times gk$ matrix that represents the variance of the moment functions.
k	is the number of instrumental variables used.
constant	is the constant $\frac{ng}{2}(1 + \ln(2\pi))$.
$\otimes$	is the notation for a Kronecker product.

All vectors are column vectors unless otherwise noted. Other estimates of the covariance matrix for FIML are also available.

Dependent Regressors and Two-Stage Least Squares

Ordinary regression analysis is based on several assumptions. A key assumption is that the independent variables are in fact statistically independent of the unobserved error component of the model. If this assumption is not true (if the regressor varies systematically with the error), then ordinary regression produces inconsistent results. The parameter estimates are *biased*.

Regressors might fail to be independent variables because they are dependent variables in a larger simultaneous system. For this reason, the problem of dependent regressors is often called *simultaneous equation bias*. For example, consider the following two-equation system:

$$y_1 = a_1 + b_1 y_2 + c_1 x_1 + \epsilon_1$$

$$y_2 = a_2 + b_2 y_1 + c_2 x_2 + \epsilon_2$$

In the first equation, y_2 is a dependent, or *endogenous*, variable. As shown by the second equation, y_2 is a function of y_1, which by the first equation is a function of ϵ_1, and therefore y_2 depends on ϵ_1. Likewise, y_1 depends on ϵ_2 and is a dependent regressor in the second equation. This is an example of a *simultaneous equation* system; y_1 and y_2 are a function of all the variables in the system.

Using the ordinary least squares (OLS) estimation method to estimate these equations produces biased estimates. One solution to this problem is to replace y_1 and y_2 on the right-hand side of the equations with predicted values, thus changing the regression problem to the following:

$$y_1 = a_1 + b_1\hat{y}_2 + c_1x_1 + \epsilon_1$$

$$y_2 = a_2 + b_2\hat{y}_1 + c_2x_2 + \epsilon_2$$

This method requires estimating the predicted values $\hat{y}_1$ and $\hat{y}_2$ through a preliminary, or "first stage," *instrumental regression*. An instrumental regression is a regression of the dependent regressors on a set of *instrumental variables*, which can be any independent variables useful for predicting the dependent regressors. In this example, the equations are linear and the exogenous variables for the whole system are known. Thus, the best choice for instruments (of the variables in the model) are the variables x_1 and x_2.

This method is known as *two-stage least squares* or 2SLS, or more generally as the *instrumental variables method*. The 2SLS method for linear models is discussed in Pindyck and Rubinfeld (1981, pp. 191–192). For nonlinear models this situation is more complex, but the idea is the same. In nonlinear 2SLS, the derivatives of the model with respect to the parameters are replaced with predicted values. See the section "Choice of Instruments" on page 1202 for further discussion of the use of instrumental variables in nonlinear regression.

To perform nonlinear 2SLS estimation with PROC MODEL, specify the instrumental variables with an INSTRUMENTS statement and specify the 2SLS or N2SLS option in the FIT statement. The following statements show how to estimate the first equation in the preceding example with PROC MODEL:

```
proc model data=in;
   y1 = a1 + b1 * y2 + c1 * x1;
   fit y1 / 2sls;
   instruments x1 x2;
run;
```

The 2SLS or instrumental variables estimator can be computed by using a first-stage regression on the instrumental variables as described previously. However, PROC MODEL actually uses the equivalent but computationally more appropriate technique of projecting the regression problem into the linear space defined by the instruments. Thus, PROC MODEL does not produce any "first stage" results when you use 2SLS. If you specify the FSRSQ option in the FIT statement, PROC MODEL prints "First-Stage R^2" statistic for each parameter estimate.

Formally, the $\hat{\boldsymbol{\theta}}$ that minimizes

$$\hat{S}_n = \frac{1}{n}\left(\sum_{t=1}^{n}(\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \theta)\otimes\mathbf{z}_t)\right)'\left(\sum_{t=1}^{n} I\otimes\mathbf{z}_t\mathbf{z}_t'\right)^{-1}\left(\sum_{t=1}^{n}(\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})\otimes\mathbf{z}_t)\right)$$

is the N2SLS estimator of the parameters. The estimate of $\boldsymbol{\Sigma}$ at the final iteration is used in the covariance of the parameters given in Table 19.2. See Amemiya (1985, p. 250) for details on the properties of nonlinear two-stage least squares.

Seemingly Unrelated Regression

If the regression equations are not simultaneous (so there are no dependent regressors), *seemingly unrelated regression* (SUR) can be used to estimate systems of equations with correlated random errors. The large-sample efficiency of an estimation can be improved if these cross-equation correlations are taken into account. SUR is also known as *joint generalized least squares* or *Zellner regression*. Formally, the $\hat{\boldsymbol{\theta}}$ that minimizes

$$\hat{S}_n = \frac{1}{n}\sum_{t=1}^{n} \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})' \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})$$

is the SUR estimator of the parameters.

The SUR method requires an estimate of the cross-equation covariance matrix, $\boldsymbol{\Sigma}$. PROC MODEL first performs an OLS estimation, computes an estimate, $\hat{\boldsymbol{\Sigma}}$, from the OLS residuals, and then performs the SUR estimation based on $\hat{\boldsymbol{\Sigma}}$. The OLS results are not printed unless you specify the OLS option in addition to the SUR option.

You can specify the $\hat{\boldsymbol{\Sigma}}$ to use for SUR by storing the matrix in a SAS data set and naming that data set in the SDATA= option. You can also feed the $\hat{\boldsymbol{\Sigma}}$ computed from the SUR residuals back into the SUR estimation process by specifying the ITSUR option. You can print the estimated covariance matrix $\hat{\boldsymbol{\Sigma}}$ by using the COVS option in the FIT statement.

The SUR method requires estimation of the $\boldsymbol{\Sigma}$ matrix, and this increases the sampling variability of the estimator for small sample sizes. The efficiency gain that SUR has over OLS is a large sample property, and you must have a reasonable amount of data to realize this gain. For a more detailed discussion of SUR, see Pindyck and Rubinfeld (1981, pp. 331–333).

Three-Stage Least Squares Estimation

If the equation system is simultaneous, you can combine the 2SLS and SUR methods to take into account both dependent regressors and cross-equation correlation of the errors. This is called *three-stage least squares* (3SLS).

Formally, the $\hat{\boldsymbol{\theta}}$ that minimizes

$$\hat{S}_n = \frac{1}{n}\left(\sum_{t=1}^{n}(\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})\otimes\mathbf{z}_t)\right)' \left(\sum_{t=1}^{n}(\hat{\boldsymbol{\Sigma}}\otimes\mathbf{z}_t\mathbf{z}_t')\right)^{-1} \left(\sum_{t=1}^{n}(\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})\otimes\mathbf{z}_t)\right)$$

is the 3SLS estimator of the parameters. For more details on 3SLS, see Gallant (1987, p. 435).

Residuals from the 2SLS method are used to estimate the $\boldsymbol{\Sigma}$ matrix required for 3SLS. The results of the preliminary 2SLS step are not printed unless the 2SLS option is also specified.

To use the three-stage least squares method, specify an INSTRUMENTS statement and use the 3SLS or N3SLS option in either the PROC MODEL statement or a FIT statement.

Generalized Method of Moments (GMM)

For systems of equations with heteroscedastic errors, generalized method of moments (GMM) can be used to obtain efficient estimates of the parameters. See the section "Heteroscedasticity" on page 1171 for alternatives to GMM.

Consider the nonlinear model

$$\begin{aligned}\boldsymbol{\epsilon}_t &= \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \\ \mathbf{z}_t &= Z(\mathbf{x}_t)\end{aligned}$$

where $\mathbf{z}_t$ is a vector of instruments and $\boldsymbol{\epsilon}_t$ is an unobservable disturbance vector that can be serially correlated and nonstationary.

In general, the following orthogonality condition is desired:

$$E(\boldsymbol{\epsilon}_t \otimes \mathbf{z}_t) = 0$$

This condition states that the expected crossproducts of the unobservable disturbances, $\boldsymbol{\epsilon}_t$, and functions of the observable variables are set to 0. The first moment of the crossproducts is

$$\begin{aligned}\mathbf{m}_n &= \frac{1}{n}\sum_{t=1}^{n} \mathbf{m}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \\ \mathbf{m}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) &= \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \otimes \mathbf{z}_t\end{aligned}$$

where $\mathbf{m}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \in R^{gk}$.

The case where $gk > p$ is considered here, where p is the number of parameters.

Estimate the true parameter vector θ^0 by the value of $\hat{\theta}$ that minimizes

$$S(\theta, \mathbf{V}) = [n\mathbf{m}_n(\theta)]' \mathbf{V}^{-1} [n\mathbf{m}_n(\theta)]/n$$

where

$$\mathbf{V} = \text{Cov}\left([n\mathbf{m}_n(\theta^0)], [n\mathbf{m}_n(\theta^0)]'\right)$$

The parameter vector that minimizes this objective function is the GMM estimator. GMM estimation is requested in the FIT statement with the GMM option.

The variance of the moment functions, $\mathbf{V}$, can be expressed as

$$\begin{aligned}\mathbf{V} &= E\left(\sum_{t=1}^{n} \boldsymbol{\epsilon}_t \otimes \mathbf{z}_t\right)\left(\sum_{s=1}^{n} \boldsymbol{\epsilon}_s \otimes \mathbf{z}_s\right)' \\ &= \sum_{t=1}^{n}\sum_{s=1}^{n} E\left[(\boldsymbol{\epsilon}_t \otimes \mathbf{z}_t)(\boldsymbol{\epsilon}_s \otimes \mathbf{z}_s)'\right] \\ &= n\mathbf{S}_n^0\end{aligned}$$

where $\mathbf{S}_n^0$ is estimated as

$$\hat{\mathbf{S}}_n = \frac{1}{n}\sum_{t=1}^{n}\sum_{s=1}^{n} (\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \otimes \mathbf{z}_t)(\mathbf{q}(\mathbf{y}_s, \mathbf{x}_s, \boldsymbol{\theta}) \otimes \mathbf{z}_s)'$$

Note that $\hat{\mathbf{S}}_n$ is a $gk \times gk$ matrix. Because Var $(\hat{\mathbf{S}}_n)$ does not decrease with increasing n, you consider estimators of $\mathbf{S}_n^0$ of the form:

$$\begin{aligned}
\hat{\mathbf{S}}_n(l(n)) &= \sum_{\tau=-n+1}^{n-1} \hat{w}(\frac{\tau}{l(n)})\mathbf{D}\hat{\mathbf{S}}_{n,\tau}\mathbf{D} \\
\hat{\mathbf{S}}_{n,\tau} &= \begin{cases} \sum_{t=1+\tau}^{n} [\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}^{\#}) \otimes \mathbf{z}_t][\mathbf{q}(\mathbf{y}_{t-\tau}, \mathbf{x}_{t-\tau}, \boldsymbol{\theta}^{\#}) \otimes \mathbf{z}_{t-\tau}]' & \tau \geq 0 \\ (\hat{\mathbf{S}}_{n,-\tau})' & \tau < 0 \end{cases} \\
\hat{w}(\frac{\tau}{l(n)}) &= \begin{cases} w(\frac{\tau}{l(n)}) & l(n) > 0 \\ \delta_{\tau,0} & l(n) = 0 \end{cases}
\end{aligned}$$

where $l(n)$ is a scalar function that computes the bandwidth parameter, $w(\cdot)$ is a scalar valued kernel, and the Kronecker delta function, $\delta_{i,j}$, is 1 if $i = j$ and 0 otherwise. The diagonal matrix $\mathbf{D}$ is used for a small sample degrees of freedom correction (Gallant 1987). The initial $\theta^{\#}$ used for the estimation of $\hat{\mathbf{S}}_n$ is obtained from a 2SLS estimation of the system. The degrees of freedom correction is handled by the VARDEF= option as it is for the $\mathbf{S}$ matrix estimation.

The following kernels are supported by PROC MODEL. They are listed with their default bandwidth functions.

Bartlett: KERNEL=BART

$$\begin{aligned}
w(x) &= \begin{cases} 1 - |x| & |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} \\
l(n) &= \frac{1}{2} n^{1/3}
\end{aligned}$$

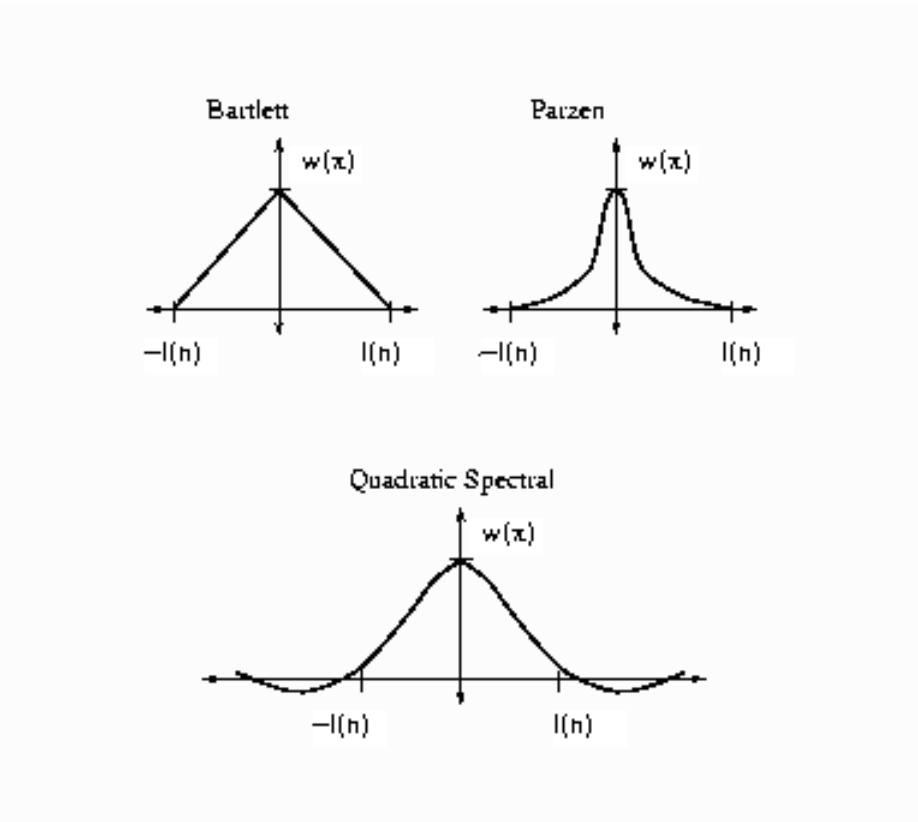
Parzen: KERNEL=PARZEN

$$\begin{aligned}
w(x) &= \begin{cases} 1 - 6|x|^2 + 6|x|^3 & 0 \leq |x| \leq \frac{1}{2} \\ 2(1 - |x|)^3 & \frac{1}{2} \leq |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} \\
l(n) &= n^{1/5}
\end{aligned}$$

Quadratic spectral: KERNEL=QS

$$\begin{aligned}
w(x) &= \frac{25}{12\pi^2 x^2} \left(\frac{\sin(6\pi x/5)}{6\pi x/5} - \cos(6\pi x/5) \right) \\
l(n) &= \frac{1}{2} n^{1/5}
\end{aligned}$$

Figure 19.23 Kernels for Smoothing



Details of the properties of these and other kernels are given in Andrews (1991). Kernels are selected with the KERNEL= option; KERNEL=PARZEN is the default. The general form of the KERNEL= option is

```
KERNEL=( PARZEN | QS | BART, c, e )
```

where the $e \geq 0$ and $c \geq 0$ are used to compute the bandwidth parameter as

$$l(n) = cn^e$$

The bias of the standard error estimates increases for large bandwidth parameters. A warning message is produced for bandwidth parameters greater than $n^{\frac{1}{3}}$. For a discussion of the computation of the optimal $l(n)$, refer to Andrews (1991).

The "Newey-West" kernel (Newey and West 1987) corresponds to the Bartlett kernel with bandwidth parameter $l(n) = L + 1$. That is, if the "lag length" for the Newey-West kernel is L, then the corresponding MODEL procedure syntax is KERNEL=(bart, L+1, 0).

Andrews and Monahan (1992) show that using prewhitening in combination with GMM can improve confidence interval coverage and reduce over rejection of t statistics at the cost of inflating the variance and MSE of the estimator. Prewhitening can be performed by using the %AR macros.

For the special case that the errors are not serially correlated—that is,

$$E(e_t \otimes \mathbf{z}_t)(e_s \otimes \mathbf{z}_s) = 0 \qquad t \neq s$$

the estimate for $\mathbf{S}_n^0$ reduces to

$$\hat{\mathbf{S}}_n = \frac{1}{n}\sum_{t=1}^{n}[\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})\otimes\mathbf{z}_t][\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})\otimes\mathbf{z}_t]'$$

The option KERNEL=(*kernel*,0,) is used to select this type of estimation when using GMM.

Covariance of GMM estimators

The covariance of GMM estimators, given a general weighting matrix $\mathbf{V}_{\mathrm{G}}^{-1}$, is

$$[(\mathbf{YX})'\mathbf{V}_{\mathrm{G}}^{-1}(\mathbf{YX})]^{-1}(\mathbf{YX})'\mathbf{V}_{\mathrm{G}}^{-1}\hat{\mathbf{V}}\mathbf{V}_{\mathrm{G}}^{-1}(\mathbf{YX})[(\mathbf{YX})'\mathbf{V}_{\mathrm{G}}^{-1}(\mathbf{YX})]^{-1}$$

By default or when GENGMMV is specified, this is the covariance of GMM estimators.

If the weighting matrix is the same as $\hat{\mathbf{V}}$, then the covariance of GMM estimators becomes

$$[(\mathbf{YX})'\hat{\mathbf{V}}^{-1}(\mathbf{YX})]^{-1}$$

If NOGENGMMV is specified, this is used as the covariance estimators.

Testing Overidentifying Restrictions

Let r be the number of unique instruments times the number of equations. The value r represents the number of orthogonality conditions imposed by the GMM method. Under the assumptions of the GMM method, $r - p$ linearly independent combinations of the orthogonality should be close to zero. The GMM estimates are computed by setting these combinations to zero. When r exceeds the number of parameters to be estimated, the OBJECTIVE*N, reported at the end of the estimation, is an asymptotically valid statistic to test the null hypothesis that the overidentifying restrictions of the model are valid. The OBJECTIVE*N is distributed as a chi-square with $r - p$ degrees of freedom (Hansen 1982, p. 1049). When the GMM method is selected, the value of the overidentifying restrictions test statistic, also known as Hansen's J test statistic, and its associated number of degrees of freedom are reported together with the probability under the null hypothesis.

Iterated Generalized Method of Moments (ITGMM)

Iterated generalized method of moments is similar to the iterated versions of 2SLS, SUR, and 3SLS. The variance matrix for GMM estimation is reestimated at each iteration with the parameters determined by the GMM estimation. The iteration terminates when the variance matrix for the equation errors change less than the CONVERGE= value. Iterated generalized method of moments is selected by the ITGMM option on the FIT statement. For some indication of the small sample properties of ITGMM, see Ferson and Foerster (1993).

Simulated Method of Moments (SMM)

The SMM method uses simulation techniques in model inference and estimation. It is appropriate for estimating models in which integrals appear in the objective function, and these integrals can be approximated by simulation. There might be various reasons for integrals to appear in an objective function (for example, transformation of a latent model into an observable model, missing data, random coefficients, heterogeneity, and so on).

This simulation method can be used with all the estimation methods except full information maximum likelihood (FIML) in PROC MODEL. SMM, also known as simulated generalized method of moments (SGMM), is the default estimation method because of its nice properties.

Estimation Details

A general nonlinear model can be described as

$$\boldsymbol{\epsilon}_t = \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})$$

where $\mathbf{q} \in R^g$ is a real vector valued function of $\mathbf{y}_t \in R^g$, $\mathbf{x}_t \in R^l$, $\boldsymbol{\theta} \in R^p$; g is the number of equations; l is the number of exogenous variables (lagged endogenous variables are considered exogenous here); p is the number of parameters; and t ranges from 1 to n. $\boldsymbol{\epsilon}_t$ is an unobservable disturbance vector with the following properties:

$$\begin{aligned} E(\boldsymbol{\epsilon}_t) &= 0 \\ E(\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_t^{'}) &= \boldsymbol{\Sigma} \end{aligned}$$

In many cases, it is not possible to write $\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})$ in a closed form. Instead $\mathbf{q}$ is expressed as an integral of a function $\mathbf{f}$; that is,

$$\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) = \int \mathbf{f}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}, \mathbf{u}_t) dP(\mathbf{u})$$

where $\mathbf{f} \in R^g$ is a real vector valued function of $\mathbf{y}_t \in R^g$, $\mathbf{x}_t \in R^l$, $\boldsymbol{\theta} \in R^p$, and $\mathbf{u}_t \in R^m$, m is the number of stochastic variables with a known distribution $P(\mathbf{u})$. Since the distribution of $\mathbf{u}$ is completely known, it is possible to simulate artificial draws from this distribution. Using such independent draws $\mathbf{u}_{ht}$, $h = 1, \ldots, H$, and the strong law of large numbers, $\mathbf{q}$ can be approximated by

$$\frac{1}{H} \sum_{h=1}^{H} \mathbf{f}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}, \mathbf{u}_{ht}).$$

Simulated Generalized Method of Moments (SGMM)

Generalized method of moments (GMM) is widely used to obtain efficient estimates for general model systems. When the moment conditions are not readily available in closed forms but can be approximated by simulation, simulated generalized method of moments (SGMM) can be used. The SGMM estimators have the nice property of being asymptotically consistent and normally distributed even if the number of draws H is fixed (see McFadden 1989; Pakes and Pollard 1989).

Consider the nonlinear model

$$\begin{aligned} \boldsymbol{\epsilon}_t &= \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) = \frac{1}{H} \sum_{h=1}^{H} \mathbf{f}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}, \mathbf{u}_{ht}) \\ \mathbf{z}_t &= Z(\mathbf{x}_t) \end{aligned}$$

where $\mathbf{z}_t \in R^k$ is a vector of k instruments and $\boldsymbol{\epsilon}_t$ is an unobservable disturbance vector that can be serially correlated and nonstationary. In the case of no instrumental variables, $\mathbf{z}_t$ is 1. $\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})$ is the vector of moment conditions, and it is approximated by simulation.

In general, theory suggests the following orthogonality condition

$$E(\boldsymbol{\epsilon}_t \otimes \mathbf{z}_t) = 0$$

which states that the expected crossproducts of the unobservable disturbances, $\boldsymbol{\epsilon}_t$, and functions of the observable variables are set to 0. The sample means of the crossproducts are

$$\begin{aligned} \mathbf{m}_n &= \frac{1}{n}\sum_{t=1}^{n}\mathbf{m}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \\ \mathbf{m}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) &= \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \otimes \mathbf{z}_t \end{aligned}$$

where $\mathbf{m}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \in R^{gk}$. The case where $gk > p$, where p is the number of parameters, is considered here. An estimate of the true parameter vector $\theta^{\mathbf{0}}$ is the value of $\hat{\theta}$ that minimizes

$$S(\theta, V) = [n\mathbf{m}_n(\theta)]'\mathbf{V}^{-1}[n\mathbf{m}_n(\theta)]/n$$

where

$$\mathbf{V} = \mathrm{Cov}\left(\mathbf{m}(\theta^{\mathbf{0}}), \mathbf{m}(\theta^{\mathbf{0}})'\right).$$

The steps for SGMM are as follows:

1. Start with a positive definite $\hat{\mathbf{V}}$ matrix. This $\hat{\mathbf{V}}$ matrix can be estimated from a consistent estimator of θ. If $\hat{\theta}$ is a consistent estimator, then $\mathbf{u}_t$ for $t = 1, \ldots, n$ can be simulated H' number of times. A consistent estimator of $\mathbf{V}$ is obtained as

$$\hat{\mathbf{V}} = \frac{1}{n}\sum_{t=1}^{n}[\frac{1}{H'}\sum_{h=1}^{H'}\mathbf{f}(\mathbf{y}_t, \mathbf{x}_t, \hat{\boldsymbol{\theta}}, \mathbf{u}_{ht}) \otimes \mathbf{z}_t][\frac{1}{H'}\sum_{h=1}^{H'}\mathbf{f}(\mathbf{y}_t, \mathbf{x}_t, \hat{\boldsymbol{\theta}}, \mathbf{u}_{ht}) \otimes \mathbf{z}_t]'$$

H' must be large so that this is an consistent estimator of $\mathbf{V}$.

2. Simulate H number of $\mathbf{u}_t$ for $t = 1, \ldots, n$. As shown by Gourieroux and Monfort (1993), the number of simulations H does not need to be very large. For $H = 10$, the SGMM estimator achieves 90% of the efficiency of the corresponding GMM estimator. Find $\hat{\theta}$ that minimizes the quadratic product of the moment conditions again with the weight matrix being $\hat{\mathbf{V}}^{-1}$.

$$\min_{\theta}[n\mathbf{m}_n(\theta)]'\hat{\mathbf{V}}^{-1}[n\mathbf{m}_n(\theta)]/n$$

3. The covariance matrix of $\sqrt{n}\theta$ is given as (Gourieroux and Monfort 1993)

$$\boldsymbol{\Sigma}_1^{-1}\mathbf{D}\hat{\mathbf{V}}^{-1}\mathbf{V}(\hat{\theta})\hat{\mathbf{V}}^{-1}\mathbf{D}'\boldsymbol{\Sigma}_1^{-1} + \frac{1}{H}\boldsymbol{\Sigma}_1^{-1}\mathbf{D}\hat{\mathbf{V}}^{-1}E[\mathbf{z}\otimes Var(\mathbf{f}|\mathbf{x})\otimes\mathbf{z}]\hat{\mathbf{V}}^{-1}\mathbf{D}'\boldsymbol{\Sigma}_1^{-1}$$

where $\boldsymbol{\Sigma}_1 = \mathbf{D}\hat{\mathbf{V}}^{-1}\mathbf{D}$, $\mathbf{D}$ is the matrix of partial derivatives of the residuals with respect to the parameters, $\mathbf{V}(\hat{\theta})$ is the covariance of moments from estimated parameters $\hat{\theta}$, and $Var(\mathbf{f}|\mathbf{x})$ is the covariance of moments for each observation from simulation. The first term is the variance-covariance matrix of the exact GMM estimator, and the second term accounts for the variation contributed by simulating the moments.

Implementation in PROC MODEL

In PROC MODEL, if the user specifies the GMM and NDRAW options in the FIT statement, PROC MODEL first fits the model by using N2SLS and computes $\hat{\mathbf{V}}$ by using the estimates from N2SLS and H' simulation. If NO2SLS is specified in the FIT statement, $\hat{\mathbf{V}}$ is read from VDATA= data set. If the user does not provide a $\hat{\mathbf{V}}$ matrix, the initial starting value of θ is used as the estimator for computing the $\hat{\mathbf{V}}$ matrix in step 1. If ITGMM option is specified instead of GMM, then PROC MODEL iterates from step 1 to step 3 until the $\mathbf{V}$ matrix converges.

The consistency of the parameter estimates is not affected by the variance correction shown in the second term in step 3. The correction on the variance of parameter estimates is not computed by default. To add the adjustment, use ADJSMMV option on the FIT statement. This correction is of the order of $\frac{1}{H}$ and is small even for moderate H.

The following example illustrates how to use SMM to estimate a simple regression model. Suppose the model is

$$y = a + bx + u, u \sim iid\ N(0, s^2).$$

First, consider the problem in a GMM context. The first two moments of y are easily derived:

$$\begin{aligned} E(y) &= a + bx \\ E(y^2) &= (a + bx)^2 + s^2 \end{aligned}$$

Rewrite the moment conditions in the form similar to the discussion above:

$$\begin{aligned} \epsilon_{1t} &= y_t - (a + bx_t) \\ \epsilon_{2t} &= y_t^2 - (a + bx_t)^2 - s^2 \end{aligned}$$

Then you can estimate this model by using GMM with following statements:

```
proc model data=a;
   parms a b s;
   instrument x;
   eq.m1 = y-(a+b*x);
   eq.m2 = y*y - (a+b*x)**2 - s*s;
   bound s > 0;
   fit m1 m2 / gmm;
run;
```

Now suppose you do not have the closed form for the moment conditions. Instead you can simulate the moment conditions by generating H number of simulated samples based on the parameters. Then the simulated moment conditions are

$$\begin{aligned} \epsilon_{1t} &= \frac{1}{H}\sum_{h=1}^{H}\{y_t - (a + bx_t + su_{t,h})\} \\ \epsilon_{2t} &= \frac{1}{H}\sum_{h=1}^{H}\{y_t^2 - (a + bx_t + su_{t,h})^2\} \end{aligned}$$

This model can be estimated by using SGMM with the following statements:

```
proc model data=_tmpdata;
   parms a b s;
   instrument x;
   ysim = (a+b*x) + s * rannor( 98711 );
   eq.m1 = y-ysim;
   eq.m2 = y*y - ysim*ysim;
   bound s > 0;
   fit m1 m2 / gmm ndraw=10;
run;
```

You can use the following MOMENT statement instead of specifying the two moment equations above:

```
moment ysim=(1, 2);
```

In cases where you require a large number of moment equations, using the MOMENT statement to specify them is more efficient.

Note that the NDRAW= option tells PROC MODEL that this is a simulation-based estimation. Thus, the random number function RANNOR returns random numbers in estimation process. During the simulation, 10 draws of $m1$ and $m2$ are generated for each observation, and the averages enter the objective functions just as the equations specified previously.

Other Estimation Methods

The simulation method can be used not only with GMM and ITGMM, but also with OLS, ITOLS, SUR, ITSUR, N2SLS, IT2SLS, N3SLS, and IT3SLS. These simulation-based methods are similar to the corresponding methods in PROC MODEL; the only difference is that the objective functions include the average of the H simulations.

Full Information Maximum Likelihood Estimation (FIML)

A different approach to the simultaneous equation bias problem is the full information maximum likelihood (FIML) estimation method (Amemiya 1977).

Compared to the instrumental variables methods (2SLS and 3SLS), the FIML method has these advantages and disadvantages:

- FIML does not require instrumental variables.
- FIML requires that the model include the full equation system, with as many equations as there are endogenous variables. With 2SLS or 3SLS, you can estimate some of the equations without specifying the complete system.
- FIML assumes that the equations errors have a multivariate normal distribution. If the errors are not normally distributed, the FIML method might produce poor results. 2SLS and 3SLS do not assume a specific distribution for the errors.
- The FIML method is computationally expensive.

The full information maximum likelihood estimators of θ and σ are the $\hat{\theta}$ and $\hat{\sigma}$ that minimize the negative log-likelihood function:

$$\begin{aligned}\mathbf{l}_n(\boldsymbol{\theta},\boldsymbol{\sigma}) = \quad & \tfrac{ng}{2} \quad \ln(2\pi) - \sum_{t=1}^{n} \ln\left(\left|\frac{\partial \mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})}{\partial \mathbf{y}_t'}\right|\right) + \frac{n}{2}\ln\left(|\boldsymbol{\Sigma}(\sigma)|\right) \\ + \quad & \frac{1}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}(\sigma)^{-1}\sum_{t=1}^{n}\mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})\mathbf{q}'(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})\right)\end{aligned}$$

The option FIML requests full information maximum likelihood estimation. If the errors are distributed normally, FIML produces efficient estimators of the parameters. If instrumental variables are not provided, the starting values for the estimation are obtained from a SUR estimation. If instrumental variables are provided, then the starting values are obtained from a 3SLS estimation. The log-likelihood value and the l_2 norm of the gradient of the negative log-likelihood function are shown in the estimation summary.

FIML Details

To compute the minimum of $\mathbf{l}_n(\boldsymbol{\theta},\boldsymbol{\sigma})$, this function is *concentrated* using the relation

$$\boldsymbol{\Sigma}(\theta) = \frac{1}{n}\sum_{t=1}^{n}\mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})\mathbf{q}'(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})$$

This results in the concentrated negative log-likelihood function discussed in Davidson and MacKinnon (1993):

$$\mathbf{l}_n(\boldsymbol{\theta}) = \frac{ng}{2}(1+\ln(2\pi)) - \sum_{t=1}^{n}\ln\left|\frac{\partial}{\partial \mathbf{y}_t'}\mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})\right| + \frac{n}{2}\ln|\boldsymbol{\Sigma}(\theta)|$$

The gradient of the negative log-likelihood function is

$$\frac{\partial}{\partial\theta_i}\mathbf{l}_n(\boldsymbol{\theta}) = \sum_{t=1}^{n}\nabla_i(t)$$

$$\begin{aligned}\nabla_i(t) \quad = \quad & -\mathrm{tr}\left(\left(\frac{\partial \mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})}{\partial \mathbf{y}_t'}\right)^{-1}\frac{\partial^2 \mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})}{\partial \mathbf{y}_t'\partial\theta_i}\right) \\ + \quad & \frac{1}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}(\theta)^{-1}\frac{\partial\boldsymbol{\Sigma}(\theta)}{\partial\theta_i}\right. \\ & \left.\left[I - \boldsymbol{\Sigma}(\theta)^{-1}\mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})\mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})'\right]\right) \\ + \quad & \mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta}')\boldsymbol{\Sigma}(\theta)^{-1}\frac{\partial \mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})}{\partial\theta_i}\end{aligned}$$

where

$$\frac{\partial\boldsymbol{\Sigma}(\theta)}{\partial\theta_i} = \frac{2}{n}\sum_{t=1}^{n}\mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})\frac{\partial \mathbf{q}(\mathbf{y}_t,\mathbf{x}_t,\boldsymbol{\theta})'}{\partial\theta_i}$$

The estimator of the variance-covariance of $\hat{\theta}$ (COVB) for FIML can be selected with the COVBEST= option with the following arguments:

CROSS selects the crossproducts estimator of the covariance matrix (Gallant 1987, p. 473):

$$C = \left(\frac{1}{n} \sum_{t=1}^{n} \nabla(t) \nabla'(t) \right)^{-1}$$

where $\nabla(t) = [\nabla_1(t), \nabla_2(t), \ldots, \nabla_p(t)]'$. This is the default.

GLS selects the generalized least squares estimator of the covariance matrix. This is computed as (Dagenais 1978)

$$C = [\hat{\mathbf{Z}}'(\mathbf{\Sigma}(\theta)^{-1} \otimes I)\hat{\mathbf{Z}}]^{-1}$$

where $\hat{\mathbf{Z}} = (\hat{Z}_1, \hat{Z}_2, \ldots, \hat{Z}_p)$ is $ng \times p$ and each $\hat{Z}_i$ column vector is obtained from stacking the columns of

$$\mathbf{U} \frac{1}{n} \sum_{t=1}^{n} \left(\frac{\partial \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})'}{\partial y} \right)^{-1} \frac{\partial^2 \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})'}{\partial \mathbf{y}_n' \partial \theta_i} - Q_i$$

$\mathbf{U}$ is an $n \times g$ matrix of residuals and q_i is an $n \times g$ matrix $\frac{\partial \mathbf{Q}}{\partial \theta_i}$.

FDA selects the inverse of concentrated likelihood Hessian as an estimator of the covariance matrix. The Hessian is computed numerically, so for a large problem this is computationally expensive.

The HESSIAN= option controls which approximation to the Hessian is used in the minimization procedure. Alternate approximations are used to improve convergence and execution time. The choices are as follows:

CROSS The crossproducts approximation is used.

GLS The generalized least squares approximation is used (default).

FDA The Hessian is computed numerically by finite differences.

HESSIAN=GLS has better convergence properties in general, but COVBEST=CROSS produces the most pessimistic standard error bounds. When the HESSIAN= option is used, the default estimator of the variance-covariance of $\hat{\theta}$ is the inverse of the Hessian selected.

Multivariate t Distribution Estimation

The multivariate *t* distribution is specified by using the ERRORMODEL statement with the T option. Other method specifications (FIML and OLS, for example) are ignored when the ERRORMODEL statement is used for a distribution other than normal.

The probability density function for the multivariate *t* distribution is

$$P_q = \frac{\Gamma(\frac{df+m}{2})}{(\pi * df)^{\frac{m}{2}} * \Gamma(\frac{df}{2}) \, |\mathbf{\Sigma}(\sigma)|^{\frac{1}{2}}} * \left(1 + \frac{\mathbf{q}'(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) \mathbf{\Sigma}(\sigma)^{-1} \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})}{df} \right)^{-\frac{df+m}{2}}$$

where m is the number of equations and df is the degrees of freedom.

The maximum likelihood estimators of θ and σ are the $\hat{\theta}$ and $\hat{\sigma}$ that minimize the negative log-likelihood function:

$$\begin{aligned}\mathbf{l}_n(\boldsymbol{\theta},\boldsymbol{\sigma}) &= -\sum_{t=1}^{n}\ln\left(\frac{\Gamma(\frac{df+m}{2})}{(\pi * df)^{\frac{m}{2}} * \Gamma(\frac{df}{2})} * \left(1+\frac{q_t'\boldsymbol{\Sigma}^{-1}q_t}{df}\right)^{-\frac{df+m}{2}}\right) \\ &\quad +\frac{n}{2} * \ln(|\boldsymbol{\Sigma}|) - \sum_{t=1}^{n}\ln\left(\left|\frac{\partial q_t}{\partial y_t'}\right|\right)\end{aligned}$$

The ERRORMODEL statement is used to request the *t* distribution maximum likelihood estimation. An OLS estimation is done to obtain initial parameter estimates and MSE.*var* estimates. Use NOOLS to turn off this initial estimation. If the errors are distributed normally, *t* distribution estimation produces results similar to FIML.

The multivariate model has a single shared degrees-of-freedom parameter, which is estimated. The degrees-of-freedom parameter can also be set to a fixed value. The log-likelihood value and the l_2 norm of the gradient of the negative log-likelihood function are shown in the estimation summary.

t Distribution Details

Since a variance term is explicitly specified by using the ERRORMODEL statement, $\boldsymbol{\Sigma}(\theta)$ is estimated as a correlation matrix and $\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})$ is normalized by the variance. The gradient of the negative log-likelihood function with respect to the degrees of freedom is

$$\begin{aligned}\frac{\partial l_n}{\partial df} &= \frac{nm}{2\,df} - \frac{n}{2}\frac{\Gamma'(\frac{df+m}{2})}{\Gamma(\frac{df+m}{2})} + \frac{n}{2}\frac{\Gamma'(\frac{df}{2})}{\Gamma(\frac{df}{2})} + \\ &\quad 0.5\log(1+\frac{\mathbf{q}'\boldsymbol{\Sigma}^{-1}\mathbf{q}}{df}) - \frac{0.5(df+m)}{(1+\frac{\mathbf{q}'\boldsymbol{\Sigma}^{-1}\mathbf{q}}{df})}\frac{\mathbf{q}'\boldsymbol{\Sigma}^{-1}\mathbf{q}}{df^2}\end{aligned}$$

The gradient of the negative log-likelihood function with respect to the parameters is

$$\frac{\partial l_n}{\partial \theta_i} = \frac{0.5(df+m)}{(1+\mathbf{q}'\boldsymbol{\Sigma}^{-1}\mathbf{q}/df)}\left[\frac{(2\,\mathbf{q}'\boldsymbol{\Sigma}^{-1}\frac{\partial \mathbf{q}}{\partial \theta_i})}{df} + \mathbf{q}'\boldsymbol{\Sigma}^{-1}\frac{\partial \boldsymbol{\Sigma}}{\partial \theta_i}\boldsymbol{\Sigma}^{-1}\mathbf{q}\right] - \frac{n}{2}\mathrm{trace}(\boldsymbol{\Sigma}^{-1}\frac{\partial \boldsymbol{\Sigma}}{\partial \theta_i})$$

where

$$\frac{\partial \boldsymbol{\Sigma}(\theta)}{\partial \theta_i} = \frac{2}{n}\sum_{t=1}^{n}\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})\frac{\partial \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta})'}{\partial \theta_i}$$

and

$$\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \boldsymbol{\theta}) = \frac{\epsilon(\theta)}{\sqrt{h(\theta)}} \in R^{m\times n}$$

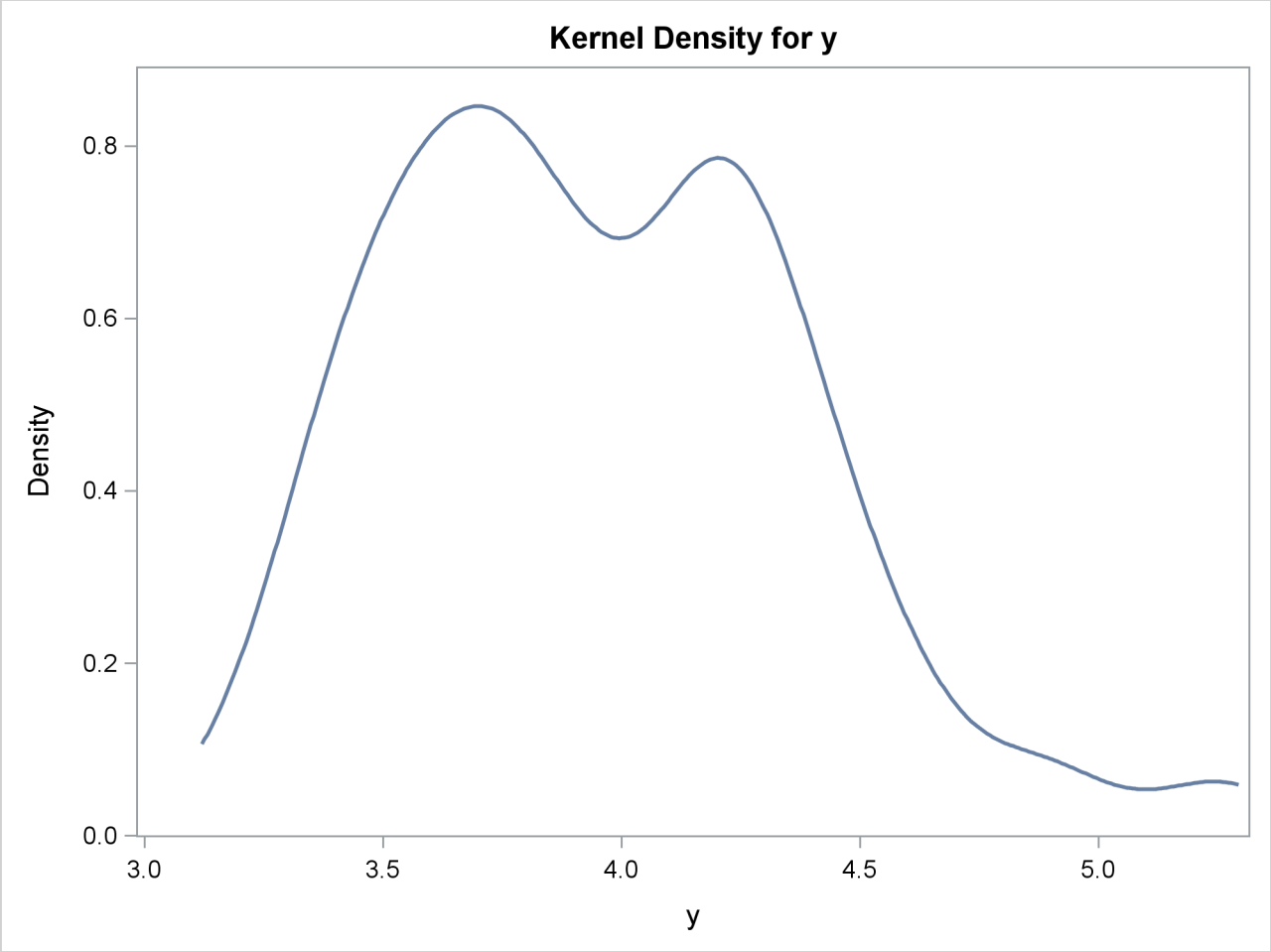
The estimator of the variance-covariance of $\hat{\theta}$ (COVB) for the *t* distribution is the inverse of the likelihood Hessian. The gradient is computed analytically, and the Hessian is computed numerically.

Empirical Distribution Estimation and Simulation

The following SAS statements fit a model that uses least squares as the likelihood function, but represent the distribution of the residuals with an empirical cumulative distribution function (CDF). The plot of the empirical probability distribution is shown in Figure 19.24.

```
data t;  /* Sum of two normals  */
   format date monyy.;
   do t = 0 to 9.9 by 0.1;
      date = intnx( 'month', '1jun90'd, (t*10)-1 );
      y =  0.1 * (rannor(123)-10) +
            .5 * (rannor(123)+10);
      output;
   end;
run;

ods select Model.Liklhood.ResidSummary
           Model.Liklhood.ParameterEstimates;

proc model data=t time=t itprint;
   dependent y;
   parm a 5;

   y = a;
   obj = resid.y * resid.y;
   errormodel y ~ general( obj )
   cdf=(empirical=(tails=( normal percent=10)));

   fit y / outsn=s out=r;
   id  date;

   solve y / data=t(where=(date='1aug98'd))
             residdata=r sdata=s
             random=200 seed=6789 out=monte ;
run;


proc kde data=monte;
   univar y / plots=density;
run;
```

Figure 19.24 Empirical PDF Plot



For simulation, if the CDF for the model is not built in to the procedure, you can use the CDF=EMPIRICAL() option. This uses the sorted residual data to create an empirical CDF. For computing the inverse CDF, the program needs to know how to handle the tails. For continuous data, the tail distribution is generally poorly determined. To counter this, the PERCENT= option specifies the percentage of the observations to use in constructing each tail. The default for the PERCENT= option is 10.

A normal distribution or a *t* distribution is used to extrapolate the tails to infinity. The standard errors for this extrapolation are obtained from the data so that the empirical CDF is continuous.

Properties of the Estimates

All of the methods are consistent. Small sample properties might not be good for nonlinear models. The tests and standard errors reported are based on the convergence of the distribution of the estimates to a normal distribution in large samples.

These nonlinear estimation methods reduce to the corresponding linear systems regression methods if the model is linear. If this is the case, PROC MODEL produces the same estimates as PROC SYSLIN.

Except for GMM, the estimation methods assume that the equation errors for each observation are identically and independently distributed with a 0 mean vector and positive definite covariance matrix $\mathbf{\Sigma}$ consistently

estimated by $\mathbf{S}$. For FIML, the errors need to be normally distributed. There are no other assumptions concerning the distribution of the errors for the other estimation methods.

The consistency of the parameter estimates relies on the assumption that the $\mathbf{S}$ matrix is a consistent estimate of $\mathbf{\Sigma}$. These standard error estimates are asymptotically valid, but for nonlinear models they might not be reliable for small samples.

The $\mathbf{S}$ matrix used for the calculation of the covariance of the parameter estimates is the best estimate available for the estimation method selected. For $\mathbf{S}$-iterated methods, this is the most recent estimation of $\mathbf{\Sigma}$. For OLS and 2SLS, an estimate of the $\mathbf{S}$ matrix is computed from OLS or 2SLS residuals and used for the calculation of the covariance matrix. For a complete list of the $\mathbf{S}$ matrix used for the calculation of the covariance of the parameter estimates, see Table 19.2.

Missing Values

An observation is excluded from the estimation if any variable used for FIT tasks is missing, if the weight for the observation is not greater than 0 when weights are used, or if a DELETE statement is executed by the model program. Variables used for FIT tasks include the equation errors for each equation, the instruments, if any, and the derivatives of the equation errors with respect to the parameters estimated. Note that variables can become missing as a result of computational errors or calculations with missing values.

The number of usable observations can change when different parameter values are used; some parameter values can be invalid and cause execution errors for some observations. PROC MODEL keeps track of the number of usable and missing observations at each pass through the data, and if the number of missing observations counted during a pass exceeds the number that was obtained using the previous parameter vector, the pass is terminated and the new parameter vector is considered infeasible. PROC MODEL never takes a step that produces more missing observations than the current estimate does.

The values used to compute the Durbin-Watson, R^2, and other statistics of fit are from the observations used in calculating the objective function and do not include any observation for which any needed variable was missing (residuals, derivatives, and instruments).

Details on the Covariance of Equation Errors

There are several $\mathbf{S}$ matrices that can be involved in the various estimation methods and in forming the estimate of the covariance of parameter estimates. These $\mathbf{S}$ matrices are estimates of $\mathbf{\Sigma}$, the true covariance of the equation errors. Apart from the choice of instrumental or noninstrumental methods, many of the methods provided by PROC MODEL differ in the way the various $\mathbf{S}$ matrices are formed and used.

All of the estimation methods result in a final estimate of $\mathbf{\Sigma}$, which is included in the output if the COVS option is specified. The final $\mathbf{S}$ matrix of each method provides the initial $\mathbf{S}$ matrix for any subsequent estimation.

This estimate of the covariance of equation errors is defined as

$$\mathbf{S} = \mathbf{D}(\mathbf{R}'\mathbf{R})\mathbf{D}$$

where $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_g)$ is composed of the equation residuals computed from the current parameter estimates in an $n \times g$ matrix and $\mathbf{D}$ is a diagonal matrix that depends on the VARDEF= option.

For VARDEF=N, the diagonal elements of $\mathbf{D}$ are $1/\sqrt{n}$, where n is the number of nonmissing observations. For VARDEF=WGT, n is replaced with the sum of the weights. For VARDEF=WDF, n is replaced with the

sum of the weights minus the model degrees of freedom. For the default VARDEF=DF, the *i*th diagonal element of **D** is $1/\sqrt{n - df_i}$, where df_i is the degrees of freedom (number of parameters) for the *i*th equation. Binkley and Nelson (1984) show the importance of using a degrees-of-freedom correction in estimating $\mathbf{\Sigma}$. Their results indicate that the DF method produces more accurate confidence intervals for N3SLS parameter estimates in the linear case than the alternative approach they tested. VARDEF=N is always used for the computation of the FIML estimates.

For the fixed **S** methods, the OUTSUSED= option writes the **S** matrix used in the estimation to a data set. This **S** matrix is either the estimate of the covariance of equation errors matrix from the preceding estimation, or a prior $\mathbf{\Sigma}$ estimate read in from a data set when the SDATA= option is specified. For the diagonal **S** methods, all of the off-diagonal elements of the **S** matrix are set to 0 for the estimation of the parameters and for the OUTSUSED= data set, but the output data set produced by the OUTS= option contains the off-diagonal elements. For the OLS and N2SLS methods, there is no previous estimate of the covariance of equation errors matrix, and the option OUTSUSED= saves an identity matrix unless a prior $\mathbf{\Sigma}$ estimate is supplied by the SDATA= option. For FIML, the OUTSUSED= data set contains the **S** matrix computed with VARDEF=N. The OUTS= data set contains the **S** matrix computed with the selected VARDEF= option. Both versions of the **S** matrix appear in the printed output for FIML.

If the COVS option is used, the method is not **S**-iterated, **S** is not an identity, and the OUTSUSED= matrix is included in the printed output.

For the methods that iterate the covariance of equation errors matrix, the **S** matrix is iteratively re-estimated from the residuals produced by the current parameter estimates. This **S** matrix estimate iteratively replaces the previous estimate until both the parameter estimates and the estimate of the covariance of equation errors matrix converge. The final OUTS= matrix and OUTSUSED= matrix are thus identical for the **S**-iterated methods.

Nested Iterations

By default, for **S**-iterated methods, the **S** matrix is held constant until the parameters converge once. Then the **S** matrix is reestimated. One iteration of the parameter estimation algorithm is performed, and the **S** matrix is again reestimated. This latter process is repeated until convergence of both the parameters and the **S** matrix. Since the objective of the minimization depends on the **S** matrix, this has the effect of chasing a moving target.

When the NESTIT option is specified, iterations are performed to convergence for the structural parameters with a fixed **S** matrix. The **S** matrix is then reestimated, the parameter iterations are repeated to convergence, and so on until both the parameters and the **S** matrix converge. This has the effect of fixing the objective function for the inner parameter iterations. It is more reliable, but usually more expensive, to nest the iterations.

R-Square Statistic

For unrestricted linear models with an intercept successfully estimated by OLS, R^2 is always between 0 and 1. However, nonlinear models do not necessarily encompass the dependent mean as a special case and can produce negative R^2 statistics. Negative R^2 statistics can also be produced even for linear models when an estimation method other than OLS is used and no intercept term is in the model.

R^2 is defined for normalized equations as

$$R^2 = 1 - \frac{SSE}{SSA - \bar{y}^2 \times n}$$

where SSA is the sum of the squares of the actual y's and $\bar{y}$ are the actual means. R^2 cannot be computed for models in general form because of the need for an actual Y.

Minimization Methods

PROC MODEL currently supports two methods for minimizing the objective function. These methods are described in the following sections.

GAUSS

The Gauss-Newton parameter-change vector for a system with g equations, n nonmissing observations, and p unknown parameters is

$$\mathbf{\Delta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{r}$$

where $\mathbf{\Delta}$ is the change vector, $\mathbf{X}$ is the stacked $ng \times p$ Jacobian matrix of partial derivatives of the residuals with respect to the parameters, and $\mathbf{r}$ is an $ng \times 1$vector of the stacked residuals. The components of $\mathbf{X}$ and $\mathbf{r}$ are weighted by the $\mathbf{S}^{-1}$ matrix. When instrumental methods are used, $\mathbf{X}$ and $\mathbf{r}$ are the projections of the Jacobian matrix and residuals vector in the instruments space and not the Jacobian and residuals themselves. In the preceding formula, $\mathbf{S}$ and $\mathbf{W}$ are suppressed. If instrumental variables are used, then the change vector becomes:

$$\mathbf{\Delta} = (\mathbf{X}'(\mathbf{S}^{-1}\otimes\mathbf{W})\mathbf{X})^{-1}\mathbf{X}'(\mathbf{S}^{-1}\otimes\mathbf{W})\mathbf{r}$$

This vector is computed at the end of each iteration. The objective function is then computed at the changed parameter values at the start of the next iteration. If the objective function is not improved by the change, the $\mathbf{\Delta}$ vector is reduced by one-half and the objective function is reevaluated. The change vector will be halved up to MAXSUBITER= times until the objective function is improved. If the objective function cannot be improved after MAXSUBITER= times, the procedure switches to the MARQUARDT method described in the next section to further improve the objective function.

For FIML, the $\mathbf{X}'\mathbf{X}$ matrix is substituted with one of three choices for approximations to the Hessian. See the section "Full Information Maximum Likelihood Estimation (FIML)" on page 1142 in this chapter.

MARQUARDT

The Marquardt-Levenberg parameter change vector is

$$\mathbf{\Delta} = (\mathbf{X}'\mathbf{X} + \lambda\mathrm{diag}(\mathbf{X}'\mathbf{X}))^{-1}\mathbf{X}'\mathbf{r}$$

where $\mathbf{\Delta}$ is the change vector, and $\mathbf{X}$ and $\mathbf{r}$ are the same as for the Gauss-Newton method, described in the preceding section. Before the iterations start, λ is set to a small value (1E–6). At each iteration, the objective function is evaluated at the parameters changed by $\mathbf{\Delta}$. If the objective function is not improved, λ is increased to 10λ and the step is tried again. λ can be increased up to MAXSUBITER= times to a maximum of 1E15 (whichever comes first) until the objective function is improved. For the start of the next iteration, λ is reduced to max(λ/10,1E–10).

Convergence Criteria

There are a number of measures that could be used as convergence or stopping criteria. PROC MODEL computes five convergence measures labeled R, S, PPC, RPC, and OBJECT.

When an estimation technique that iterates estimates of $\mathbf{\Sigma}$ is used (that is, IT3SLS), two convergence criteria are used. The termination values can be specified with the CONVERGE=(p,s) option in the FIT statement. If the second value, s, is not specified, it defaults to p. The criterion labeled S (described later in the section) controls the convergence of the **S** matrix. When S is less than s, the **S** matrix has converged. The criterion labeled R is compared to the p-value to test convergence of the parameters.

The R convergence measure cannot be computed accurately in the special case of singular residuals (when all the residuals are close to 0) or in the case of a 0 objective value. When either the trace of the **S** matrix computed from the current residuals (trace(S)) or the objective value is less than the value of the SINGULAR= option, convergence is assumed.

The various convergence measures are explained in the following:

R is the primary convergence measure for the parameters. It measures the degree to which the residuals are orthogonal to the Jacobian columns, and it approaches 0 as the gradient of the objective function becomes small. R is defined as the square root of

$$\frac{(r'(\mathbf{S}^{-1}\otimes\mathbf{W})\mathbf{X}(\mathbf{X}'(\mathbf{S}^{-1}\otimes\mathbf{W})\mathbf{X})^{-1}\mathbf{X}'(\mathbf{S}^{-1}\otimes\mathbf{W})r)}{(r'(\mathbf{S}^{-1}\otimes\mathbf{W})r)}$$

where **X** is the Jacobian matrix and **r** is the residuals vector. R is similar to the relative offset orthogonality convergence criterion proposed by Bates and Watts (1981).

In the univariate case, the R measure has several equivalent interpretations:

- the cosine of the angle between the residuals vector and the column space of the Jacobian matrix. When this cosine is 0, the residuals are orthogonal to the partial derivatives of the predicted values with respect to the parameters, and the gradient of the objective function is 0.
- the square root of the R^2 for the current linear pseudo-model in the residuals
- a norm of the gradient of the objective function, where the normalizing matrix is proportional to the current estimate of the covariance of the parameter estimates. Thus, using R, convergence is judged when the gradient becomes small in this norm.
- the prospective relative change in the objective function value expected from the next GAUSS step, assuming that the current linearization of the model is a good local approximation.

In the multivariate case, R is somewhat more complicated but is designed to go to 0 as the gradient of the objective becomes small and can still be given the previous interpretations for the aggregation of the equations weighted by $\mathbf{S}^{-1}$.

PPC is the prospective parameter change measure. PPC measures the maximum relative change in the parameters implied by the parameter-change vector computed for the next iteration.

At the *k*th iteration, PPC is the maximum over the parameters

$$\frac{\left|\theta_i^{k+1} - \theta_i^k\right|}{\left|\theta_i^k\right| + 10^{-6}}$$

where θ_i^k is the current value of the *i*th parameter and θ_i^{k+1} is the prospective value of this parameter after adding the change vector computed for the next iteration. The parameter with the maximum prospective relative change is printed with the value of PPC, unless the PPC is nearly 0.

RPC is the retrospective parameter change measure. RPC measures the maximum relative change in the parameters from the previous iteration. At the *k*th iteration, RPC is the maximum over *i* of

$$\frac{\left|\theta_i^k - \theta_i^{k-1}\right|}{\left|\theta_i^{k-1}\right| + 10^{-6}}$$

where θ_i^k is the current value of the *i*th parameter and θ_i^{k-1} is the previous value of this parameter. The name of the parameter with the maximum retrospective relative change is printed with the value of RPC, unless the RPC is nearly 0.

OBJECT measures the relative change in the objective function value between iterations:

$$\frac{\left|O^k - O^{k-1}\right|}{\left|O^{k-1}\right| + 10^{-6}}$$

where O^{k-1} is the value of the objective function (O^k) from the previous iteration.

S measures the relative change in the **S** matrix. S is computed as the maximum over *i, j* of

$$\frac{\left|S_{ij}^k - S_{ij}^{k-1}\right|}{\left|S_{ij}^{k-1}\right| + 10^{-6}}$$

where S^{k-1} is the previous **S** matrix. The S measure is relevant only for estimation methods that iterate the **S** matrix.

An example of the convergence criteria output is shown in Figure 19.25.

Figure 19.25 Convergence Criteria Output

Final Convergence Criteria	
R	0.000737
PPC(b)	0.003943
RPC(b)	0.00968
Object	4.784E-6
Trace(S)	0.533325
Objective Value	0.522214

The Trace(S) is the trace (the sum of the diagonal elements) of the **S** matrix computed from the current residuals. This row is labeled MSE if there is only one equation.

Troubleshooting Convergence Problems

As with any nonlinear estimation routine, there is no guarantee that the estimation will be successful for a given model and data. If the equations are linear with respect to the parameters, the parameter estimates always converge in one iteration. The methods that iterate the **S** matrix must iterate further for the **S** matrix to converge. Nonlinear models might not necessarily converge.

Convergence can be expected only with fully identified parameters, adequate data, and starting values sufficiently close to solution estimates.

Convergence and the rate of convergence might depend primarily on the choice of starting values for the estimates. This does not mean that a great deal of effort should be invested in choosing starting values. First, try the default values. If the estimation fails with these starting values, examine the model and data and rerun the estimation using reasonable starting values. It is usually not necessary that the starting values be very good, just that they not be very bad; choose values that seem plausible for the model and data.

An Example of Requiring Starting Values

Suppose you want to regress a variable Y on a variable X, assuming that the variables are related by the following nonlinear equation:

$$y = a + bx^c + \epsilon$$

In this equation, Y is linearly related to a power transformation of X. The unknown parameters are a, b, and c. ϵ is an unobserved random error. The following SAS statements generate simulated data. In this simulation, $a = 10$, $b = 2$, and the use of the SQRT function corresponds to $c = .5$.

```
data test;
  do i = 1 to 20;
     x = 5 * ranuni(1234);
     y = 10 + 2 * sqrt(x) + .5 * rannor(1234);
     output;
     end;
run;
```

The following statements specify the model and give descriptive labels to the model parameters. Then the FIT statement attempts to estimate a, b, and c by using the default starting value 0.0001.

```
proc model data=test;
   y = a + b * x ** c;
   label a = "Intercept"
         b = "Coefficient of Transformed X"
         c = "Power Transformation Parameter";
   fit y;
run;
```

PROC MODEL prints model summary and estimation problem summary reports and then prints the output shown in Figure 19.26.

Figure 19.26 Diagnostics for Convergence Failure

The MODEL Procedure
OLS Estimation

ERROR: The parameter estimates failed to converge for OLS after 100 iterations using CONVERGE=0.001 as the convergence criteria.

The MODEL Procedure
OLS Estimation

	Iteration	N Obs	R	Objective	N Subit	a	b	c
OLS	100	20	0.9627	3.9678	2	137.3822	-126.533	-0.00213

Gauss Method Parameter Change Vector

a	b	c
-69369.08	69368.01	-1.16

By using the default starting values, PROC MODEL is unable to take even the first step in iterating to the solution. The change in the parameters that the Gauss-Newton method computes is very extreme and makes the objective values worse instead of better. Even when this step is shortened by a factor of a million, the objective function is still worse, and PROC MODEL is unable to estimate the model parameters.

The problem is caused by the starting value of C. Using the default starting value C=0.0001, the first iteration attempts to compute better values of A and B by what is, in effect, a linear regression of Y on the 10,000th root of X, which is almost the same as the constant 1. Thus the matrix that is inverted to compute the changes is nearly singular and affects the accuracy of the computed parameter changes.

This is also illustrated by the next part of the output, which displays collinearity diagnostics for the crossproducts matrix of the partial derivatives with respect to the parameters, shown in Figure 19.27.

Figure 19.27 Collinearity Diagnostics

Collinearity Diagnostics

			Proportion of Variation		
Number	Eigenvalue	Condition Number	a	b	c
1	2.376793	1.0000	0.0000	0.0000	0.0000
2	0.623207	1.9529	0.0000	0.0000	0.0000
3	1.68475E-12	1187758	1.0000	1.0000	1.0000

This output shows that the matrix is singular and that the partials of A, B, and C with respect to the residual are collinear at the point (0.0001, 0.0001, 0.0001) in the parameter space. See the section "Linear Dependencies" on page 1161 for a full explanation of the collinearity diagnostics.

The MODEL procedure next prints the note shown in Figure 19.28, which suggests that you try different starting values.

Figure 19.28 Estimation Failure Note

Note: The parameter estimation is abandoned. Check your model and data. If the model is correct and the input data are appropriate, try rerunning the parameter estimation using different starting values for the parameter estimates.
PROC MODEL continues as if the parameter estimates had converged.

PROC MODEL then produces the usual printout of results for the nonconverged parameter values. The estimation summary is shown in Figure 19.29. The heading includes the reminder "(Not Converged)".

Figure 19.29 Nonconverged Estimation Summary

The MODEL Procedure
OLS Estimation Summary (Not Converged)

Data Set Options	
DATA=	TEST

Minimization Summary	
Parameters Estimated	3
Method	Gauss
Iterations	100
Subiterations	239
Average Subiterations	2.39

Final Convergence Criteria	
R	0.962666
PPC(b)	548.2193
RPC(b)	540.3066
Object	2.654E-6
Trace(S)	4.667946
Objective Value	3.967754

Observations Processed	
Read	20
Solved	20

The nonconverged estimation results are shown in Figure 19.30.

Figure 19.30 Nonconverged Results

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors (Not Converged)

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
y	3	17	79.3551	4.6679	2.1605	-1.6812	-1.9966

Note that the R^2 statistic is negative. An $R^2 < 0$ results when the residual mean squared error for the model is larger than the variance of the dependent variable. Negative R^2 statistics might be produced when either

the parameter estimates fail to converge correctly, as in this case, or when the correctly estimated model fits the data very poorly.

Controlling Starting Values

To fit the preceding model you must specify a better starting value for C. Avoid starting values of C that are either very large or close to 0. For starting values of A and B, you can specify values, use the default, or have PROC MODEL fit starting values for them conditional on the starting value for C.

Starting values are specified with the START= option of the FIT statement or in a PARMS statement. In PROC MODEL, you have several options to specify starting values for the parameters to be estimated. When more than one option is specified, the options are implemented in the following order of precedence (from highest to lowest): the START= option, the PARMS statement initialization value, the ESTDATA= option, and the PARMSDATA= option. When no starting values for the parameter estimates are specified with BY group processing, the default start value 0.0001 is used for each by group. Again, when no starting values are specified, and a model with a FIT statement is stored by the OUTMODEL=*outmodel-filename* option in a previous step, the *outmodel-filename* can be invoked in a subsequent PROC MODEL step by using the MODEL=*outmodel-filename* option with multiple estimation methods in the second step. In such a case, the parameter estimates from the *outmodel-filename* are used directly as starting values for OLS, and OLS results from the second step provide starting values for the subsequent estimation method such as 2SLS or SUR, provided that NOOLS is not specified.

For example, the following statements estimate the model parameters by using the starting values A=0.0001, B=0.0001, and C=5.

```
proc model data=test;
   y = a + b * x ** c;
   label a = "Intercept"
         b = "Coefficient of Transformed X"
         c = "Power Transformation Parameter";
   fit y start=(c=5);
run;
```

Using these starting values, the estimates converge in 16 iterations. The results are shown in Figure 19.31. Note that since the START= option explicitly declares parameters, the parameter C is placed first in the table.

Figure 19.31 Converged Results

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
y	3	17	5.7359	0.3374	0.5809	0.8062	0.7834

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
c	0.327079	0.2892	1.13	0.2738	Power Transformation Parameter
a	8.384311	3.3775	2.48	0.0238	Intercept
b	3.505391	3.4858	1.01	0.3287	Coefficient of Transformed X

Using the STARTITER Option

PROC MODEL can compute starting values for some parameters conditional on starting values you specify for the other parameters. You supply starting values for some parameters and specify the STARTITER option on the FIT statement.

For example, the following statements set C to 1 and compute starting values for A and B by estimating these parameters conditional on the fixed value of C. With C=1, this is equivalent to computing A and B by linear regression on X. A PARMS statement is used to declare the parameters in alphabetical order. The ITPRINT option is used to print the parameter values at each iteration.

```
proc model data=test;
   parms a b c;
   y = a + b * x ** c;
   label a = "Intercept"
         b = "Coefficient of Transformed X"
         c = "Power Transformation Parameter";
   fit y start=(c=1) / startiter itprint;
run;
```

With better starting values, the estimates converge in only eight iterations. Counting the iteration required to compute the starting values for A and B, this is seven fewer than the 16 iterations required without the STARTITER option. The iteration history listing is shown in Figure 19.32.

Figure 19.32 ITPRINT Listing

The MODEL Procedure
OLS Estimation

	Iteration	N Obs	R	Objective	N Subit	a	b	c
GRID	0	20	0.9989	162.9	0	0.00010	0.00010	1.00000
GRID	1	20	0.0000	0.3464	0	10.96530	0.77007	1.00000

	Iteration	N Obs	R	Objective	N Subit	a	b	c
OLS	0	20	0.3873	0.3464	0	10.96530	0.77007	1.00000
OLS	1	20	0.3339	0.3282	2	10.75993	0.99433	0.83096
OLS	2	20	0.3244	0.3233	1	10.46894	1.31205	0.66810
OLS	3	20	0.3151	0.3197	1	10.11707	1.69149	0.54626
OLS	4	20	0.2764	0.3110	1	9.74691	2.08492	0.46615
OLS	5	20	0.2379	0.3040	0	9.06175	2.80546	0.36575
OLS	6	20	0.0612	0.2879	0	8.51825	3.36746	0.33201
OLS	7	20	0.0022	0.2868	0	8.39485	3.49449	0.32776
OLS	8	20	0.0001	0.2868	0	8.38467	3.50502	0.32711

NOTE: At OLS Iteration 8 CONVERGE=0.001 Criteria Met.

The results produced in this case are almost the same as the results shown in Figure 19.31, except that the PARMS statement causes the parameter estimates table to be ordered A, B, C instead of C, A, B. They are not exactly the same because the different starting values caused the iterations to converge at a slightly different place. This effect is controlled by changing the convergence criterion with the CONVERGE= option.

By default, the STARTITER option performs one iteration to find starting values for the parameters that are not given values. In this case, the model is linear in A and B, so only one iteration is needed. If A or B were nonlinear, you could specify more than one "starting values" iteration by specifying a number for the STARTITER= option.

Finding Starting Values by Grid Search

PROC MODEL can try various combinations of parameter values and use the combination that produces the smallest objective function value as starting values. (For OLS the objective function is the residual mean square.) This is known as a preliminary *grid search*. You can combine the STARTITER option with a grid search.

For example, the following statements try five different starting values for C: 1, 0.7, 0.5, 0.3, and 0. For each value of C, values for A and B are estimated. The combination of A, B, and C values that produce the smallest residual mean square is then used to start the iterative process.

```
proc model data=test;
   parms a b c;
   y = a + b * x ** c;
   label a = "Intercept"
         b = "Coefficient of Transformed X"
         c = "Power Transformation Parameter";
   fit y start=(c=1 .7 .5 .3 0) / startiter itprint;
run;
```

The iteration history listing is shown in Figure 19.33. Using the best starting values found by the grid search, the OLS estimation only requires two iterations. However, since the grid search required nine iterations, the total iterations in this case is 11.

Figure 19.33 ITPRINT Listing

The MODEL Procedure
OLS Estimation

	Iteration	N Obs	R	Objective	N Subit	a	b	c
GRID	0	20	0.9989	162.9	0	0.00010	0.00010	1.00000
GRID	1	20	0.0000	0.3464	0	10.96530	0.77007	1.00000
GRID	0	20	0.7587	0.7242	0	10.96530	0.77007	0.70000
GRID	1	20	0.0000	0.3073	0	10.41027	1.36141	0.70000
GRID	0	20	0.7079	0.5843	0	10.41027	1.36141	0.50000
GRID	1	20	0.0000	0.2915	0	9.69319	2.13103	0.50000
GRID	0	20	0.7747	0.7175	0	9.69319	2.13103	0.30000
GRID	1	20	0.0000	0.2869	0	8.04397	3.85767	0.30000
GRID	0	20	0.5518	2.1277	0	8.04397	3.85767	0.00000
GRID	1	20	0.0000	1.4799	0	8.04397	4.66255	0.00000

	Iteration	N Obs	R	Objective	N Subit	a	b	c
OLS	0	20	0.0189	0.2869	0	8.04397	3.85767	0.30000
OLS	1	20	0.0158	0.2869	0	8.35023	3.54145	0.32233
OLS	2	20	0.0006	0.2868	0	8.37468	3.51540	0.32622

NOTE: At OLS Iteration 2 CONVERGE=0.001 Criteria Met.

Because no initial values for A or B were provided in the PARAMETERS statement or were read in with a PARMSDATA= or ESTDATA= option, A and B were given the default value of 0.0001 for the first iteration. At the second grid point, C=5, the values of A and B obtained from the previous iterations are used for the initial iteration. If initial values are provided for parameters, the parameters start at those initial values at each grid point.

Guessing Starting Values from the Logic of the Model

Example 19.1, which uses a logistic growth curve model of the U.S. population, illustrates the need for reasonable starting values. This model can be written

$$pop = \frac{a}{1 + \exp(b - c(t - 1790))}$$

where t is time in years. The model is estimated by using decennial census data of the U.S. population in millions. If this simple but highly nonlinear model is estimated by using the default starting values, the estimation fails to converge.

To find reasonable starting values, first consider the meaning of a and c. Taking the limit as time increases, a is the limiting or maximum possible population. So, as a starting value for a, several times the most recent population known can be used—for example, one billion (1000 million).

Dividing the time derivative by the function to find the growth rate and taking the limit as t moves into the past, you can determine that c is the initial growth rate. You can examine the data and compute an estimate of the growth rate for the first few decades, or you can pick a number that sounds like a plausible population growth rate figure, such as 2%.

To find a starting value for b, let t equal the base year used, 1790, which causes c to drop out of the formula for that year, and then solve for the value of b that is consistent with the known population in 1790 and with the starting value of a. This yields $b = \ln(a/3.9 - 1)$ or about 5.5, where a is 1000 and 3.9 is roughly the population for 1790 given in the data. The estimates converge using these starting values.

Convergence Problems

When estimating nonlinear models, you might encounter some of the following convergence problems.

Unable to Improve

The optimization algorithm might be unable to find a step that improves the objective function. If this happens in the Gauss-Newton method, the step size is halved to find a change vector for which the objective improves. In the Marquardt method, λ is increased to find a change vector for which the objective improves. If, after MAXSUBITER= step-size halvings or increases in λ, the change vector still does not produce a better objective value, the iterations are stopped and an error message is printed.

Failure of the algorithm to improve the objective value can be caused by a CONVERGE= value that is too small. Look at the convergence measures reported at the point of failure. If the estimates appear to be approximately converged, you can accept the NOT CONVERGED results reported, or you can try rerunning the FIT task with a larger CONVERGE= value.

If the procedure fails to converge because it is unable to find a change vector that improves the objective value, check your model and data to ensure that all parameters are identified and data values are reasonably scaled. Then, rerun the model with different starting values. Also, consider using the Marquardt method if the Gauss-Newton method fails; the Gauss-Newton method can get into trouble if the Jacobian matrix is nearly

singular or ill-conditioned. Keep in mind that a nonlinear model may be well-identified and well-conditioned for parameter values close to the solution values but unidentified or numerically ill-conditioned for other parameter values. The choice of starting values can make a big difference.

Nonconvergence

The estimates might diverge into areas where the program overflows or the estimates might go into areas where function values are illegal or too badly scaled for accurate calculation. The estimation might also take steps that are too small or that make only marginal improvement in the objective function and thus fail to converge within the iteration limit.

When the estimates fail to converge, collinearity diagnostics for the Jacobian crossproducts matrix are printed if there are 20 or fewer parameters estimated. See the section "Linear Dependencies" on page 1161 for an explanation of these diagnostics.

Inadequate Convergence Criterion

If convergence is obtained, the resulting estimates approximate only a minimum point of the objective function. The statistical validity of the results is based on the exact minimization of the objective function, and for nonlinear models the quality of the results depends on the accuracy of the approximation of the minimum. This is controlled by the convergence criterion used.

There are many nonlinear functions for which the objective function is quite flat in a large region around the minimum point so that many quite different parameter vectors might satisfy a weak convergence criterion. By using different starting values, different convergence criteria, or different minimization methods, you can produce very different estimates for such models.

You can guard against this by running the estimation with different starting values and different convergence criteria and checking that the estimates produced are essentially the same. If they are not, use a smaller CONVERGE= value.

Local Minimum

You might have converged to a local minimum rather than a global one. This problem is difficult to detect because the procedure appears to have succeeded. You can guard against this by running the estimation with different starting values or with a different minimization technique. The START= option can be used to automatically perform a grid search to aid in the search for a global minimum.

Discontinuities

The computational methods assume that the model is a continuous and smooth function of the parameters. If this is not the case, the methods might not work.

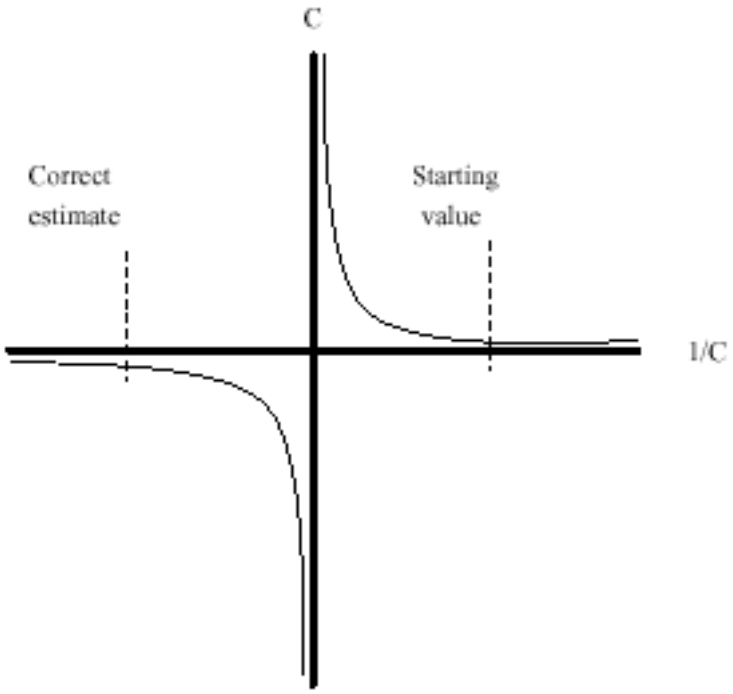
If the model equations or their derivatives contain discontinuities, the estimation usually succeeds, provided that the final parameter estimates lie in a continuous interval and that the iterations do not produce parameter values at points of discontinuity or parameter values that try to cross asymptotes.

One common case of discontinuities causing estimation failure is that of an asymptotic discontinuity between the final estimates and the initial values. For example, consider the following model, which is basically linear but is written with one parameter in reciprocal form:

```
y = a + b * x1 + x2 / c;
```

By placing the parameter C in the denominator, a singularity is introduced into the parameter space at C=0. This is not necessarily a problem, but if the correct estimate of C is negative while the starting value is positive (or vice versa), the asymptotic discontinuity at 0 will lie between the estimate and the starting value. This means that the iterations have to pass through the singularity to get to the correct estimates. The situation is shown in Figure 19.34.

Figure 19.34 Asymptotic Discontinuity



Because of the incorrect sign of the starting value, the C estimate goes off towards positive infinity in a vain effort to get past the asymptote and onto the correct arm of the hyperbola. As the computer is required to work with ever closer approximations to infinity, the numerical calculations break down and an "objective function was not improved" convergence failure message is printed. At this point, the iterations terminate with an extremely large positive value for C. When the sign of the starting value for C is changed, the estimates converge quickly to the correct values.

Linear Dependencies

In some cases, the Jacobian matrix might not be of full rank; parameters might not be fully identified for the current parameter values with the current data. When linear dependencies occur among the derivatives of the model, some parameters appear with a standard error of 0 and with the word BIASED printed in place of the *t* statistic. When this happens, collinearity diagnostics for the Jacobian crossproducts matrix are printed if the DETAILS option is specified and there are twenty or fewer parameters estimated. Collinearity diagnostics are also printed out automatically when a minimization method fails, or when the COLLIN option is specified.

For each parameter, the proportion of the variance of the estimate accounted for by each *principal component* is printed. The principal components are constructed from the eigenvalues and eigenvectors of the correlation

matrix (scaled covariance matrix). When collinearity exists, a principal component is associated with proportion of the variance of more than one parameter. The numbers reported are proportions so they remain between 0 and 1. If two or more parameters have large proportion values associated with the same principal component, then two problems can occur: the computation of the parameter estimates are slow or nonconvergent; and the parameter estimates have inflated variances (Belsley, Kuh, and Welsch 1980, pp. 105–117).

For example, the following cubic model is fit to a quadratic data set:

```
proc model data=test3;
   exogenous x1;
   parms b1 a1 c1 ;
   y1 = a1 * x1 + b1 * x1 * x1  + c1 * x1 * x1 *x1;
   fit y1 / collin;
run;
```

The collinearity diagnostics are shown in Figure 19.35.

Figure 19.35 Collinearity Diagnostics

The MODEL Procedure

Collinearity Diagnostics

			Proportion of Variation		
Number	Eigenvalue	Condition Number	b1	a1	c1
1	2.942920	1.0000	0.0001	0.0004	0.0002
2	0.056638	7.2084	0.0001	0.0357	0.0148
3	0.000442	81.5801	0.9999	0.9639	0.9850

Notice that the proportions associated with the smallest eigenvalue are almost 1. For this model, removing any of the parameters decreases the variances of the remaining parameters.

In many models, the collinearity might not be clear cut. Collinearity is not necessarily something you remove. A model might need to be reformulated to remove the redundant parameterization, or the limitations on the estimability of the model can be accepted. The GINV=G4 option can be helpful to avoid problems with convergence for models containing collinearities.

Collinearity diagnostics are also useful when an estimation does not converge. The diagnostics provide insight into the numerical problems and can suggest which parameters need better starting values. These diagnostics are based on the approach of Belsley, Kuh, and Welsch (1980).

Iteration History

The options ITPRINT, ITDETAILS, XPX, I, and ITALL specify a detailed listing of each iteration of the minimization process.

ITPRINT Option

The ITPRINT information is selected whenever any iteration information is requested.

The following information is displayed for each iteration:

N	is the number of usable observations.
Objective	is the corrected objective function value.
Trace(S)	is the trace of the **S** matrix.
subit	is the number of subiterations required to find a λ or a damping factor that reduces the objective function.
R	is the R convergence measure.

The estimates for the parameters at each iteration are also printed.

ITDETAILS Option

The additional values printed for the ITDETAILS option are:

Theta	is the angle in degrees between Δ, the parameter change vector, and the negative gradient of the objective function.
Phi	is the directional derivative of the objective function in the Δ direction scaled by the objective function.
Stepsize	is the value of the damping factor used to reduce Δ if the Gauss-Newton method is used.
Lambda	is the value of λ if the Marquardt method is used.
Rank(XPX)	is the rank of the $\mathbf{X}'\mathbf{X}$ matrix (output if the projected Jacobian crossproducts matrix is singular).

The definitions of PPC and R are explained in the section "Convergence Criteria" on page 1151. When the values of PPC are large, the parameter associated with the criteria is displayed in parentheses after the value.

XPX and I Options

The XPX and the I options select the printing of the augmented $\mathbf{X}'\mathbf{X}$ matrix and the augmented $\mathbf{X}'\mathbf{X}$ matrix after a *sweep* operation (Goodnight 1979) has been performed on it. An example of the output from the following statements is shown in Figure 19.36.

```
proc model data=test2;
   y1 = a1 * x2 * x2 - exp( d1*x1);
   y2 = a2 * x1 * x1 + b2 * exp( d2*x2);
   fit y1 y2 / itall XPX I ;
run;
```

Figure 19.36 XPX and I Options Output

The MODEL Procedure
OLS Estimation

Cross Products for System At OLS Iteration 0

	a1	d1	a2	b2	d2	Residual
a1	1839468	-33818.35	0.0	0.00	0.000000	3879959
d1	-33818	1276.45	0.0	0.00	0.000000	-76928
a2	0	0.00	42925.0	1275.15	0.154739	470686
b2	0	0.00	1275.2	50.01	0.003867	16055
d2	0	0.00	0.2	0.00	0.000064	2
Residual	3879959	-76928.14	470686.3	16055.07	2.329718	24576144

XPX Inverse for System At OLS Iteration 0

	a1	d1	a2	b2	d2	Residual
a1	0.000001	0.000028	0.000000	0.0000	0.00	2
d1	0.000028	0.001527	0.000000	0.0000	0.00	-9
a2	0.000000	0.000000	0.000097	-0.0025	-0.08	6
b2	0.000000	0.000000	-0.002455	0.0825	0.95	172
d2	0.000000	0.000000	-0.084915	0.9476	15746.71	11931
Residual	1.952150	-8.546875	5.823969	171.6234	11930.89	10819902

The first matrix, labeled "Cross Products," for OLS estimation is

$$\begin{bmatrix} \mathbf{X}'\mathbf{X} & \mathbf{X}'\mathbf{r} \\ \mathbf{r}'\mathbf{X} & \mathbf{r}'\mathbf{r} \end{bmatrix}$$

The column labeled Residual in the output is the vector $\mathbf{X}'\mathbf{r}$, which is the gradient of the objective function. The diagonal scalar value $\mathbf{r}'\mathbf{r}$ is the objective function uncorrected for degrees of freedom. The second matrix, labeled "XPX Inverse," is created through a sweep operation on the augmented $\mathbf{X}'\mathbf{X}$ matrix to get:

$$\begin{bmatrix} (\mathbf{X}'\mathbf{X})^{-1} & (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{r} \\ (\mathbf{X}'\mathbf{r})'(\mathbf{X}'\mathbf{X})^{-1} & \mathbf{r}'\mathbf{r} - (\mathbf{X}'\mathbf{r})'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{r} \end{bmatrix}$$

Note that the residual column is the change vector used to update the parameter estimates at each iteration. The corner scalar element is used to compute the R convergence criteria.

ITALL Option

The ITALL option, in addition to causing the output of all of the preceding options, outputs the **S** matrix, the inverse of the **S** matrix, the CROSS matrix, and the swept CROSS matrix. An example of a portion of the CROSS matrix for the preceding example is shown in Figure 19.37.

Figure 19.37 ITALL Option Crossproducts Matrix Output

The MODEL Procedure
OLS Estimation

Crossproducts Matrix At OLS Iteration 0					
	1	**@PRED.y1/@a1**	**@PRED.y1/@d1**	**@PRED.y2/@a2**	**@PRED.y2/@b2**
1	50.00	6409	-239.16	1275.0	50.00
@PRED.y1/@a1	6409.08	1839468	-33818.35	187766.1	6409.88
@PRED.y1/@d1	-239.16	-33818	1276.45	-7253.0	-239.19
@PRED.y2/@a2	1275.00	187766	-7253.00	42925.0	1275.15
@PRED.y2/@b2	50.00	6410	-239.19	1275.2	50.01
@PRED.y2/@d2	0.00	1	-0.03	0.2	0.00
RESID.y1	14699.97	3879959	-76928.14	420582.9	14701.77
RESID.y2	16052.76	4065028	-85083.68	470686.3	16055.07

Crossproducts Matrix At OLS Iteration 0			
	@PRED.y2/@d2	**RESID.y1**	**RESID.y2**
1	0.003803	14700	16053
@PRED.y1/@a1	0.813934	3879959	4065028
@PRED.y1/@d1	-0.026177	-76928	-85084
@PRED.y2/@a2	0.154739	420583	470686
@PRED.y2/@b2	0.003867	14702	16055
@PRED.y2/@d2	0.000064	2	2
RESID.y1	1.820356	11827102	12234106
RESID.y2	2.329718	12234106	12749042

Computer Resource Requirements

If you are estimating large systems, you need to be aware of how PROC MODEL uses computer resources (such as memory and the CPU) so they can be used most efficiently.

Saving Time with Large Data Sets

If your input data set has many observations, the FIT statement performs a large number of model program executions. A pass through the data is made at least once for each iteration and the model program is executed once for each observation in each pass. If you refine the starting estimates by using a smaller data set, the final estimation with the full data set might require fewer iterations.

For example, you could use

```
proc model;
   /* Model goes here */
   fit / data=a(obs=25);
   fit / data=a;
```

where OBS=25 selects the first 25 observations in A. The second FIT statement produces the final estimates using the full data set and starting values from the first run.

Fitting the Model in Sections to Save Space and Time

If you have a very large model (with several hundred parameters, for example), the procedure uses considerable space and time. You might be able to save resources by breaking the estimation process into several steps and estimating the parameters in subsets.

You can use the FIT statement to select for estimation only the parameters for selected equations. Do not break the estimation into too many small steps; the total computer time required is minimized by compromising between the number of FIT statements that are executed and the size of the crossproducts matrices that must be processed.

When the parameters are estimated for selected equations, the entire model program must be executed even though only a part of the model program might be needed to compute the residuals for the equations selected for estimation. If the model itself can be broken into sections for estimation (and later combined for simulation and forecasting), then more resources can be saved.

For example, to estimate the following four equation model in two steps, you could use

```
proc model data=a outmodel=part1;
   parms a0-a2 b0-b2 c0-c3 d0-d3;
   y1 = a0 + a1*y2 + a2*x1;
   y2 = b0 + b1*y1 + b2*x2;
   y3 = c0 + c1*y1 + c2*y4 + c3*x3;
   y4 = d0 + d1*y1 + d2*y3 + d3*x4;
   fit y1 y2;
   fit y3 y4;
   fit y1 y2 y3 y4;
run;
```

You should try estimating the model in pieces to save time only if there are more than 14 parameters; the preceding example takes more time, not less, and the difference in memory required is trivial.

Memory Requirements for Parameter Estimation

PROC MODEL is a large program, and it requires much memory. Memory is also required for the SAS System, various data areas, the model program and associated tables and data vectors, and a few crossproducts matrices. For most models, the memory required for PROC MODEL itself is much larger than that required for the model program, and the memory required for the model program is larger than that required for the crossproducts matrices.

The number of bytes needed for two crossproducts matrices, four **S** matrices, and three parameter covariance matrices is

$$8 \times (2 + k + m + g)^2 + 16 \times g^2 + 12 \times (p + 1)^2$$

plus lower-order terms, where m is the number of unique nonzero derivatives of each residual with respect to each parameter, g is the number of equations, k is the number of instruments, and p is the number of parameters. This formula is for the memory required for 3SLS. If you are using OLS, a reasonable estimate of the memory required for large problems (greater than 100 parameters) is to divide the value obtained from the formula in half.

Consider the following model program.

```
proc model data=test2 details;
   exogenous x1 x2;
   parms b1 100 a1 a2 b2 2.5 c2 55;
   y1 = a1 * y2 + b1 * x1 * x1;
   y2 = a2 * y1 + b2 * x2 * x2 + c2 / x2;
   fit y1 y2 / n3sls memoryuse;
   inst b1 b2 c2 x1 ;
run;
```

The DETAILS option prints the storage requirements information shown in Figure 19.38.

Figure 19.38 Storage Requirements Information

The MODEL Procedure

Storage Requirements for this Problem	
Order of XPX Matrix	6
Order of S Matrix	2
Order of Cross Matrix	13
Total Nonzero Derivatives	5
Distinct Variable Derivatives	5
Size of Cross matrix	728

The matrix $\mathbf{X}'\mathbf{X}$ augmented by the residual vector is called the XPX matrix in the output, and it has the size $m + 1$. The order of the **S** matrix, 2 for this example, is the value of g. The CROSS matrix is made up of the k unique instruments, a constant column that represents the intercept terms, followed by the m unique Jacobian variables plus a constant column that represents the parameters with constant derivatives, followed by the g residuals.

The size of two CROSS matrices in bytes is

$$8 \times (2 + k + m + g)^2 + 2 + k + m + g$$

Note that the CROSS matrix is symmetric, so only the diagonal and the upper triangular part of the matrix is stored. For examples of the CROSS and XPX matrices see the section "Iteration History" on page 1162.

The MEMORYUSE Option

The MEMORYUSE option in the FIT, SOLVE, MODEL, or RESET statement can be used to request a comprehensive memory usage summary.

Figure 19.39 shows an example of the output produced by the MEMORYUSE option.

Figure 19.39 MEMORYUSE Option Output for FIT Task

Memory Usage Summary (in bytes)	
Symbols	15652
Strings	2593
Lists	2412
Arrays	2208
Statements	2640
Opcodes	1840
Parsing	6180
Executable	12721
Block option	0
Cross reference	0
Flow analysis	336
Derivatives	28640
Data vector	368
Cross matrix	1480
X'X matrix	590
S matrix	144
GMM memory	0
Jacobian	0
Work vectors	702
Overhead	14222
----------	----------
Total	92728

Definitions of the memory components follow:

symbols	memory used to store information about variables in the model
strings	memory used to store the variable names and labels
lists	space used to hold lists of variables
arrays	memory used by ARRAY statements
statements	memory used for the list of programming statements in the model
opcodes	memory used to store the code compiled to evaluate the expression in the model program
parsing	memory used in parsing the SAS statements
executable	the compiled model program size
block option	memory used by the BLOCK option
cross ref.	memory used by the XREF option
flow analysis	memory used to compute the interdependencies of the variables
derivatives	memory used to compute and store the analytical derivatives
data vector	memory used for the program data vector
cross matrix	memory used for one or more copies of the CROSS matrix
$\mathbf{X}'\mathbf{X}$ matrix	memory used for one or more copies of the $\mathbf{X}'\mathbf{X}$ matrix
$\mathbf{S}$ matrix	memory used for the covariance matrix
GMM memory	additional memory used for the GMM and ITGMM methods
Jacobian	memory used for the Jacobian matrix for SOLVE and FIML
work vectors	memory used for miscellaneous work vectors
overhead	other miscellaneous memory

Testing for Normality

The NORMAL option in the FIT statement performs multivariate and univariate tests of normality.

The three multivariate tests provided are Mardia's skewness test and kurtosis test (Mardia 1970) and the Henze-Zirkler $T_{n,\beta}$ test (Henze and Zirkler 1990). The two univariate tests provided are the Shapiro-Wilk W test and the Kolmogorov-Smirnov test. (For details on the univariate tests, refer to "Goodness-of-Fit Tests" section in "The UNIVARIATE Procedure" chapter in the *Base SAS Procedures Guide*.) The null hypothesis for all these tests is that the residuals are normally distributed.

For a random sample $X_1, \ldots, X_n$, $X_i \in \mathrm{R}^d$, where d is the dimension of X_i and n is the number of observations, a measure of multivariate skewness is

$$b_{1,d} = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} [(X_i - \mu)' \mathbf{S}^{-1} (X_j - \mu)]^3$$

where $\mathbf{S}$ is the sample covariance matrix of $\mathbf{X}$. For weighted regression, both $\mathbf{S}$ and $(X_i - \mu)$ are computed by using the weights supplied by the WEIGHT statement or the _WEIGHT_ variable.

Mardia showed that under the null hypothesis $\frac{n}{6} b_{1,d}$ is asymptotically distributed as $\chi^2(d(d+1)(d+2)/6)$. For small samples, Mardia's skewness test statistic is calculated with a small sample correction formula, given by $\frac{nk}{6} b_{1,d}$ where the correction factor k is given by $k = (d+1)(n+1)(n+3)/n(((n+1)(d+1)) - 6)$. Mardia's skewness test statistic in PROC MODEL uses this small sample corrected formula.

A measure of multivariate kurtosis is given by

$$b_{2,d} = \frac{1}{n} \sum_{i=1}^{n} [(X_i - \mu)' \mathbf{S}^{-1} (X_i - \mu)]^2$$

Mardia showed that under the null hypothesis, $b_{2,d}$ is asymptotically normally distributed with mean $d(d+2)$ and variance $8d(d+2)/n$.

The Henze-Zirkler test is based on a nonnegative functional $D(.,.)$ that measures the distance between two distribution functions and has the property that

$$D(\mathrm{N}_d(0, I_d), Q) = 0$$

if and only if

$$Q = \mathrm{N}_d(0, I_d)$$

where $\mathrm{N}_d(\mu, \Sigma_d)$ is a d-dimensional normal distribution.

The distance measure $D(.,.)$ can be written as

$$D_\beta(P, Q) = \int_{\mathrm{R}^d} |\hat{P}(t) - \hat{Q}(t)|^2 \varphi_\beta(t) dt$$

where $\hat{P}(t)$ and $\hat{Q}(t)$ are the Fourier transforms of P and Q, and $\varphi_\beta(t)$ is a weight or a kernel function. The density of the normal distribution $\mathrm{N}_d(0, \beta^2 I_d)$ is used as $\varphi_\beta(t)$

$$\varphi_\beta(t) = (2\pi\beta^2)^{\frac{-d}{2}} \exp(\frac{-|t|^2}{2\beta^2}), t \in \mathrm{R}^d$$

where $|t| = (t't)^{0.5}$.

The parameter β depends on n as

$$\beta_d(n) = \frac{1}{\sqrt{2}}(\frac{2d+1}{4})^{1/(d+4)} n^{1/(d+4)}$$

The test statistic computed is called $T_\beta(d)$ and is approximately distributed as a lognormal. The lognormal distribution is used to compute the null hypothesis probability.

$$\begin{aligned} T_\beta(d) = \quad & \frac{1}{n} \sum_{j=1}^{n}\sum_{k=1}^{n} \exp(-\frac{\beta^2}{2}|Y_j - Y_k|^2) \\ - \quad & 2(1+\beta^2)^{-d/2} \sum_{j=1}^{n} \exp(-\frac{\beta^2}{2(1+\beta^2)}|Y_j|^2) + n(1+2\beta^2)^{-d/2} \end{aligned}$$

where

$$|Y_j - Y_k|^2 = (X_j - X_k)'\mathbf{S}^{-1}(X_j - X_k)$$

$$|Y_j|^2 = (X_j - \bar{X})'\mathbf{S}^{-1}(X_j - \bar{X})$$

Monte Carlo simulations suggest that $T_\beta(d)$ has good power against distributions with heavy tails.

The Shapiro-Wilk W test is computed only when the number of observations (n) is less than 2000 while computation of the Kolmogorov-Smirnov test statistic requires at least 2000 observations.

The following is an example of the output produced by the NORMAL option.

```
proc model data=test2;
   y1 = a1 * x2 * x2 - exp( d1*x1);
   y2 = a2 * x1 * x1 + b2 * exp( d2*x2);
   fit y1 y2 / normal ;
run;
```

Figure 19.40 Normality Test Output

The MODEL Procedure

Normality Test			
Equation	**Test Statistic**	**Value**	**Prob**
y1	Shapiro-Wilk W	0.34	<.0001
y2	Shapiro-Wilk W	0.82	<.0001
System	Mardia Skewness	286.4	<.0001
	Mardia Kurtosis	31.28	<.0001
	Henze-Zirkler T	6.65	<.0001

Heteroscedasticity

One of the key assumptions of regression is that the variance of the errors is constant across observations. If the errors have constant variance, the errors are called *homoscedastic*. Typically, residuals are plotted to assess this assumption. Standard estimation methods are inefficient when the errors are *heteroscedastic* or have nonconstant variance.

Heteroscedasticity Tests

The MODEL procedure provides two tests for heteroscedasticity of the errors: White's test and the modified Breusch-Pagan test.

Both White's test and the Breusch-Pagan are based on the residuals of the fitted model. For systems of equations, these tests are computed separately for the residuals of each equation.

The residuals of an estimation are used to investigate the heteroscedasticity of the true disturbances.

The WHITE option tests the null hypothesis

$$H_0 : \sigma_i^2 = \sigma^2 \quad \text{for all } i$$

White's test is general because it makes no assumptions about the form of the heteroscedasticity (White 1980). Because of its generality, White's test might identify specification errors other than heteroscedasticity (Thursby 1982). Thus, White's test might be significant when the errors are homoscedastic but the model is misspecified in other ways.

White's test is equivalent to obtaining the error sum of squares for the regression of squared residuals on a constant and all the unique variables in $\mathbf{J}\otimes\mathbf{J}$, where the matrix $\mathbf{J}$ is composed of the partial derivatives of the equation residual with respect to the estimated parameters. White's test statistic W is computed as follows:

$$W = nR^2$$

where R^2 is the correlation coefficient obtained from the above regression. The statistic is asymptotically distributed as chi-squared with P–1 degrees of freedom, where P is the number of regressors in the regression, including the constant and n is the total number of observations. In the example given below, the regressors are constant, income, income*income, income*income*income, and income*income*income*income. income*income occurs twice and one is dropped. Hence, P=5 with degrees of freedom, P–1=4.

Note that White's test in the MODEL procedure is different than White's test in the REG procedure requested by the SPEC option. The SPEC option produces the test from Theorem 2 on page 823 of White (1980). The WHITE option, on the other hand, produces the statistic discussed in Greene (1993).

The null hypothesis for the modified Breusch-Pagan test is homoscedasticity. The alternate hypothesis is that the error variance varies with a set of regressors, which are listed in the BREUSCH= option.

Define the matrix $\mathbf{Z}$ to be composed of the values of the variables listed in the BREUSCH= option, such that $z_{i,j}$ is the value of the jth variable in the BREUSCH= option for the ith observation. The null hypothesis of the Breusch-Pagan test is

$$\sigma_i^2 = \sigma^2(\alpha_0 + \boldsymbol{\alpha}' z_i) \qquad H_0 : \boldsymbol{\alpha} = \mathbf{0}$$

where σ_i^2 is the error variance for the ith observation and α_0 and $\boldsymbol{\alpha}$ are regression coefficients.

The test statistic for the Breusch-Pagan test is

$$bp = \frac{1}{v}(\mathbf{u} - \bar{u}\mathbf{i})'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'(\mathbf{u} - \bar{u}\mathbf{i})$$

where $\mathbf{u} = (e_1^2, e_2^2, \ldots, e_n^2)$, $\mathbf{i}$ is a $n \times 1$ vector of ones, and

$$v = \frac{1}{n}\sum_{i=1}^{n}(e_i^2 - \frac{\mathbf{e}'\mathbf{e}}{n})^2$$

This is a modified version of the Breusch-Pagan test, which is less sensitive to the assumption of normality than the original test (Greene 1993, p. 395).

The statements in the following example produce the output in Figure 19.41:

```
proc model data=schools;
   parms const inc inc2;

   exp = const + inc * income + inc2 * income * income;
   incsq = income * income;

   fit exp / white breusch=(1 income incsq);
run;
```

Figure 19.41 Output for Heteroscedasticity Tests

The MODEL Procedure

Heteroscedasticity Test					
Equation	Test	Statistic	DF	Pr > ChiSq	Variables
exp	White's Test	21.16	4	0.0003	Cross of all vars
	Breusch-Pagan	15.83	2	0.0004	1, income, incsq

Correcting for Heteroscedasticity

There are two methods for improving the efficiency of the parameter estimation in the presence of heteroscedastic errors. If the error variance relationships are known, weighted regression can be used or an error model can be estimated. For details about error model estimation, see the section "Error Covariance Structure Specification" on page 1181. If the error variance relationship is unknown, GMM estimation can be used.

Weighted Regression

The WEIGHT statement can be used to correct for the heteroscedasticity. Consider the following model, which has a heteroscedastic error term:

$$y_t = 250(e^{-0.2t} - e^{-0.8t}) + \sqrt{(9/t)}\epsilon_t$$

The data for this model is generated with the following SAS statements.

```
data test;
   do t=1 to 25;
      y = 250 * (exp( -0.2 * t ) - exp( -0.8 * t )) +
          sqrt( 9 / t ) * rannor(1);
      output;
   end;
run;
```

If this model is estimated with OLS, as shown in the following statements, the estimates shown in Figure 19.42 are obtained for the parameters.

```
proc model data=test;
   parms b1 0.1 b2 0.9;
   y = 250 * ( exp( -b1 * t ) - exp( -b2 * t ) );
   fit y;
run;
```

Figure 19.42 Unweighted OLS Estimates

The MODEL Procedure

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
b1	0.200977	0.00101	198.60	<.0001
b2	0.826236	0.00853	96.82	<.0001

If both sides of the model equation are multiplied by $\sqrt{t}$, the model has a homoscedastic error term. This multiplication or weighting is done through the WEIGHT statement. The WEIGHT statement variable operates on the squared residuals as

$$\boldsymbol{\epsilon}_t' \boldsymbol{\epsilon}_t = \text{weight} \times \mathbf{q}_t' \mathbf{q}_t$$

so that the WEIGHT statement variable represents the square of the model multiplier. The following PROC MODEL statements corrects the heteroscedasticity with a WEIGHT statement:

```
proc model data=test;
   parms b1 0.1 b2 0.9;
   y = 250 * ( exp( -b1 * t ) - exp( -b2 * t ) );
   fit y;
   weight t;
run;
```

Note that the WEIGHT statement follows the FIT statement. The weighted estimates are shown in Figure 19.43.

Figure 19.43 Weighted OLS Estimates

The MODEL Procedure

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
b1	0.200503	0.000844	237.53	<.0001
b2	0.816701	0.0139	58.71	<.0001

The weighted OLS estimates are identical to the output produced by the following PROC MODEL example:

```
proc model data=test;
   parms b1 0.1 b2 0.9;
   y = 250 * ( exp( -b1 * t ) - exp( -b2 * t ) );
  _weight_ = t;
   fit y;
run;
```

If the WEIGHT statement is used in conjunction with the _WEIGHT_ variable, the two values are multiplied together to obtain the weight used.

The WEIGHT statement and the _WEIGHT_ variable operate on all the residuals in a system of equations. If a subset of the equations needs to be weighted, the residuals for each equation can be modified through the RESID. variable for each equation. The following example demonstrates the use of the RESID. variable to make a homoscedastic error term:

```
proc model data=test;
   parms b1 0.1 b2 0.9;
   y = 250 * ( exp( -b1 * t ) - exp( -b2 * t ) );
   resid.y = resid.y * sqrt(t);
   fit y;
run;
```

These statements produce estimates of the parameters and standard errors that are identical to the weighted OLS estimates. The reassignment of the RESID.Y variable must be done after Y is assigned; otherwise it would have no effect. Also, note that the residual (RESID.Y) is multiplied by $\sqrt{t}$. Here the multiplier is acting on the residual before it is squared.

GMM Estimation

If the form of the heteroscedasticity is unknown, generalized method of moments estimation (GMM) can be used. The following PROC MODEL statements use GMM to estimate the example model used in the preceding section:

```
proc model data=test;
   parms b1 0.1 b2 0.9;
   y = 250 * ( exp( -b1 * t ) - exp( -b2 * t ) );
   fit y / gmm;
   instruments b1 b2;
run;
```

GMM is an instrumental method, so instrument variables must be provided.

GMM estimation generates estimates for the parameters shown in Figure 19.44.

Figure 19.44 GMM Estimation for Heteroscedasticity

The MODEL Procedure

Nonlinear GMM Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
b1	0.200487	0.000800	250.69	<.0001
b2	0.822148	0.0148	55.39	<.0001

Heteroscedasticity-Consistent Covariance Matrix Estimation

Homoscedasticity is required for ordinary least squares regression estimates to be efficient. A nonconstant error variance, heteroscedasticity, causes the OLS estimates to be inefficient, and the usual OLS covariance matrix, $\hat{\mathbf{\Sigma}}$, is generally invalid:

$$\hat{\mathbf{\Sigma}} = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}$$

When the variance of the errors of a classical linear model

$$Y = \mathbf{X}\beta + \epsilon$$

is not constant across observations (heteroscedastic), so that $\sigma_i^2 \neq \sigma_j^2$ for some $j > 1$, the OLS estimator

$$\hat{\beta}_{OLS} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'Y$$

is unbiased but it is inefficient. Models that take into account the changing variance can make more efficient use of the data. When the variances, σ_t^2, are known, generalized least squares (GLS) can be used and the estimator

$$\hat{\beta}_{GLS} = (\mathbf{X}'\mathbf{\Omega}\mathbf{X})^{-1}\mathbf{X}'\mathbf{\Omega}^{-1}Y$$

where

$$\mathbf{\Omega} = \begin{bmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_T^2 \end{bmatrix}$$

is unbiased and efficient. However, GLS is unavailable when the variances, σ_t^2, are unknown.

To solve this problem White (1980) proposed a heteroscedastic consistent-covariance matrix estimator (HCCME)

$$\hat{\mathbf{\Sigma}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\hat{\mathbf{\Omega}}\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$$

that is consistent as well as unbiased, where

$$\hat{\mathbf{\Omega}}_0 = \begin{bmatrix} \epsilon_1^2 & 0 & 0 & 0 \\ 0 & \epsilon_2^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \epsilon_T^2 \end{bmatrix}$$

and $\epsilon_t = Y_t - \mathbf{X}_t \hat{\beta_{OLS}}$.

This estimator is considered somewhat unreliable in finite samples. Therefore, Davidson and MacKinnon (1993) propose three different modifications to estimating $\hat{\mathbf{\Omega}}$. The first solution is to simply multiply ϵ_t^2 by $\frac{n}{n-df}$, where n is the number of observations and df is the number of explanatory variables, so that

$$\hat{\mathbf{\Omega}}_1 = \begin{bmatrix} \frac{n}{n-df}\,\epsilon_1^2 & 0 & 0 & 0 \\ 0 & \frac{n}{n-df}\,\epsilon_2^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \frac{n}{n-df}\,\epsilon_n^2 \end{bmatrix}$$

The second solution is to define

$$\hat{\mathbf{\Omega}}_2 = \begin{bmatrix} \frac{\epsilon_1^2}{1-\hat{h}_1} & 0 & 0 & 0 \\ 0 & \frac{\epsilon_2^2}{1-\hat{h}_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \frac{\epsilon_n^2}{1-\hat{h}_n} \end{bmatrix}$$

where $\hat{h}_t = \mathbf{X}_t(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}_t'$.

The third solution, called the "jackknife," is to define

$$\hat{\mathbf{\Omega}}_3 = \begin{bmatrix} \frac{\epsilon_1^2}{(1-\hat{h}_1)^2} & 0 & 0 & 0 \\ 0 & \frac{\epsilon_2^2}{(1-\hat{h}_2)^2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \frac{\epsilon_n^2}{(1-\hat{h}_T)^2} \end{bmatrix}$$

MacKinnon and White (1985) investigated these three modified HCCMEs, including the original HCCME, based on finite-sample performance of pseudo-t statistics. The original HCCME performed the worst. The first modification performed better. The second modification performed even better than the first, and the third modification performed the best. They concluded that the original HCCME should never be used in finite sample estimation, and that the second and third modifications should be used over the first modification if the diagonals of $\hat{\mathbf{\Omega}}$ are available.

Seemingly Unrelated Regression HCCME

Extending the discussion to systems of g equations, the HCCME for SUR estimation is

$$(\tilde{\mathbf{X}}'\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}'\hat{\mathbf{\Omega}}\tilde{\mathbf{X}}(\tilde{\mathbf{X}}'\tilde{\mathbf{X}})^{-1}$$

where $\tilde{\mathbf{X}}$ is a $n\ g \times k$ matrix with the first g rows representing the first observation, the next g rows representing the second observation, and so on. $\hat{\mathbf{\Omega}}$ is now a $n\ g \times n\ g$ block diagonal matrix with typical block $g \times g$

$$\hat{\mathbf{\Omega}}_i = \begin{bmatrix} \psi_{1,i}\,\psi_{1,i} & \psi_{1,i}\,\psi_{2,i} & \cdots & \psi_{1,i}\,\psi_{g,i} \\ \psi_{2,i}\,\psi_{1,i} & \psi_{2,i}\,\psi_{2,i} & \cdots & \psi_{2,i}\,\psi_{g,i} \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{g,i}\,\psi_{1,i} & \psi_{g,i}\,\psi_{2,i} & \cdots & \psi_{g,i}\,\psi_{g,i} \end{bmatrix}$$

where

$$\psi_{j,i} = \epsilon_{j,i} \quad HC_0$$

or

$$\psi_{j,i} = \sqrt{\frac{n}{n-df}}\epsilon_{j,i} \quad HC_1$$

or

$$\psi_{j,i} = \epsilon_{j,i}/\sqrt{1-\hat{h}_i} \quad HC_2$$

or

$$\psi_{j,i} = \epsilon_{j,i}/(1-\hat{h}_i) \quad HC_3$$

Two- and Three-Stage Least Squares HCCME

For two- and three-stage least squares, the HCCME for a g equation system is

$$\mathrm{Cov}\,F(\hat{\mathbf{\Omega}})\mathrm{Cov}$$

where

$$\mathrm{Cov} = \left(\frac{1}{n}\mathbf{X}'(\mathbf{I}\otimes\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}')\mathbf{X}\right)^{-1}$$

is the normal covariance matrix without the $\mathbf{S}$ matrix and

$$F(\mathbf{\Omega}) = \frac{1}{n}\sum_{i}^{g}\sum_{j}^{g}\mathbf{X}_i'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\hat{\mathbf{\Omega}}_{ij}\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}_j$$

where $\mathbf{X}_j$ is a $n \times p$ matrix with the jth equations regressors in the appropriate columns and zeros everywhere else.

$$\hat{\mathbf{\Omega}}_{ij} = \begin{bmatrix} \psi_{i,1}\psi_{j,1} & 0 & 0 & 0 \\ 0 & \psi_{i,2}\psi_{j,2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \psi_{i,n}\psi_{j,n} \end{bmatrix}$$

For 2SLS $\hat{\mathbf{\Omega}}_{ij} = 0$ when $i \neq j$. The ϵ_t used in $\hat{\mathbf{\Omega}}$ is computed by using the parameter estimates obtained from the instrumental variables estimation.

The leverage value for the ith equation used in the HCCME=2 and HCCME=3 methods is computed as conditional on the first stage as

$$h_{ti} = \mathbf{Z}_t(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{X}_i(\mathbf{X}'(\mathbf{I} \otimes \mathbf{Z}(\mathbf{Z}' * \mathbf{Z})^{-1}\mathbf{Z}')\mathbf{X})^{-1}\mathbf{X}_i'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}_t'$$

for 2SLS and

$$h_{ti} = \mathbf{Z}_t(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{X}_i(\mathbf{X}'(\mathbf{S}^{-1} \otimes \mathbf{Z}(\mathbf{Z}' * \mathbf{Z})^{-1}\mathbf{Z}')\mathbf{X})^{-1}\mathbf{X}_i'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}_t'/\mathbf{S}_{ii}$$

for 3SLS.

Testing for Autocorrelation

The GODFREY= option in the FIT statement produces the Godfrey Lagrange multiplier test for serially correlated residuals for each equation (Godfrey 1978a, b). n is the maximum autoregressive order, and specifies that Godfrey's tests be computed for lags 1 through n. The default number of lags is four.

The tests are performed separately for each equation estimated by the FIT statement. When a nonlinear model is estimated, the test is computed by using a linearized model.

The following is an example of the output produced by the GODFREY=3 option:

Figure 19.45 Autocorrelation Test Output

Godfrey Test Output

The MODEL Procedure

Godfrey's Serial Correlation Test			
Equation	Alternative	LM	Pr > LM
y	1	6.63	0.0100
	2	6.89	0.0319
	3	6.96	0.0732

The three variations of the test reported by the GODFREY=3 option are designed to have power against different alternative hypothesis. Thus, if the residuals in fact have only first-order autocorrelation, the lag 1 test has the most power for rejecting the null hypothesis of uncorrelated residuals. If the residuals have second- but not higher-order autocorrelation, the lag 2 test might be more likely to reject; the same is true for third-order autocorrelation and the lag 3 test.

The null hypothesis of Godfrey's tests is that the equation residuals are white noise. However, if the equation includes autoregressive error model of order p (AR(p),) then the lag i test, when considered in terms of the structural error, is for the null hypothesis that the structural errors are from an AR(p) process versus the alternative hypothesis that the errors are from an AR($p + i$) process.

The alternative ARMA(p, i) process is locally equivalent to the alternative AR($p + i$) process with respect to the null model AR(p). Thus, the GODFREY= option results are also a test of AR(p) errors against the alternative hypothesis of ARMA(p, i) errors. See Godfrey (1978a, b) for more detailed information.

Transformation of Error Terms

In PROC MODEL you can control the form of the error term. By default, the error term is assumed to be additive. This section demonstrates how to specify nonadditive error terms and discusses the effects of these transformations.

Models with Nonadditive Errors

The estimation methods used by PROC MODEL assume that the error terms of the equations are independently and identically distributed with zero means and finite variances. Furthermore, the methods assume that the RESID.*name* equation variable for normalized form equations or the EQ.*name* equation variable for general form equations contains an estimate of the error term of the true stochastic model whose parameters are being estimated. Details on RESID.*name* and EQ.*name* equation variables are in the section "Equation Translations" on page 1272.

To illustrate these points, consider the common loglinear model

$$y = \alpha x^{\beta} \tag{1}$$

$$\ln y = a + b\ln(x) \tag{2}$$

where a =log(α) and b =β. Equation (2) is called the *log form* of the equation in contrast to equation (1), which is called the *level form* of the equation. Using the SYSLIN procedure, you can estimate equation (2) by specifying

```
proc syslin data=in;
   model logy=logx;
run;
```

where LOGY and LOGX are the logs of Y and X computed in a preceding DATA step. The resulting values for INTERCEPT and LOGX correspond to a and b in equation (2).

Using the MODEL procedure, you can try to estimate the parameters in the level form (and avoid the DATA step) by specifying

```
proc model data=in;
   parms alpha beta;
   y = alpha * x ** beta;
   fit y;
run;
```

where ALPHA and BETA are the parameters in equation (1).

Unfortunately, at least one of the preceding is wrong; an ambiguity results because equations (1) and (2) contain no explicit error term. The SYSLIN and MODEL procedures both deal with additive errors; the residual used (the estimate of the error term in the equation) is the difference between the predicted and actual values (of LOGY for PROC SYSLIN and of Y for PROC MODEL in this example). If you perform the regressions discussed previously, PROC SYSLIN estimates equation (3) while PROC MODEL estimates equation (4).

$$\ln y = a + b\ln(x) + \epsilon \tag{3}$$

$$y = \alpha x^{\beta} + \xi \tag{4}$$

These are different statistical models. Equation (3) is the log form of equation (5)

$$y = \alpha x^{\beta} \mu \qquad (5)$$

where μ=e$^{\epsilon}$. Equation (4), on the other hand, cannot be linearized because the error term ξ (different from μ) is additive in the level form.

You must decide whether your model is equation (4) or (5). If the model is equation (4), you should use PROC MODEL. If you linearize equation (1) without considering the error term and apply SYSLIN to MODEL LOGY=LOGX, the results will be wrong. On the other hand, if your model is equation (5) (in practice it usually is), and you want to use PROC MODEL to estimate the parameters in the *level* form, you must do something to account for the multiplicative error.

PROC MODEL estimates parameters by minimizing an objective function. The objective function is computed using either the RESID.-prefixed equation variable or the EQ.-prefixed equation variable. You must make sure that these prefixed equation variables are assigned an appropriate error term. If the model has additive errors that satisfy the assumptions, nothing needs to be done. In the case of equation (5), the error is nonadditive and the equation is in normalized form, so you must alter the value of RESID.Y.

The following assigns a valid estimate of μ to RESID.Y:

```
y = alpha * x ** beta;
resid.y = actual.y / pred.y;
```

However, μ=e$^{\epsilon}$, and therefore μ, cannot have a mean of zero, and you cannot consistently estimate α and β by minimizing the sum of squares of an estimate of μ. Instead, you use $\epsilon = \ln\mu$.

```
proc model data=in;
   parms alpha beta;
   y = alpha * x ** beta;
   resid.y = log( actual.y / pred.y );
   fit y;
run;
```

If the model was expressed in general form, this transformation becomes

```
proc model data=in;
   parms alpha beta;
   EQ.trans = log( y / (alpha * x ** beta));
   fit trans;
run;
```

Both examples produce estimates of α and β of the level form that match the estimates of a and b of the log form. That is, ALPHA=exp(INTERCEPT) and BETA=LOGX, where INTERCEPT and LOGX are the PROC SYSLIN parameter estimates from the MODEL LOGY=LOGX. The standard error reported for ALPHA is different from that for the INTERCEPT in the log form.

The preceding example is not intended to suggest that loglinear models should be estimated in level form but, rather, to make the following points:

- Nonlinear transformations of equations involve the error term of the equation, and this should be taken into account when transforming models.

- The RESID.-prefixed and the EQ.-prefixed equation variables for models estimated by the MODEL procedure must represent additive errors with zero means.
- You can use assignments to RESID.-prefixed and EQ.-prefixed equation variables to transform error terms.
- Some models do not have additive errors or zero means, and many such models can be estimated using the MODEL procedure. The preceding approach applies not only to multiplicative models but to any model that can be manipulated to isolate the error term.

Predicted Values of Transformed Models

Nonadditive or transformed errors affect the distribution of the predicted values, as well as the estimates. For the preceding loglinear example, the MODEL procedure produces consistent parameter estimates. However, the predicted values for Y computed by PROC MODEL are not unbiased estimates of the expected values of Y, although they do estimate the conditional median Y values.

In general, the predicted values produced for a model with nonadditive errors are not unbiased estimates of the conditional means of the endogenous value. If the model can be transformed to a model with additive errors by using a *monotonic* transformation, the predicted values estimate the conditional medians of the endogenous variable.

For transformed models in which the biasing factor is known, you can use programming statements to correct for the bias in the predicted values as estimates of the endogenous means. In the preceding log-linear case, the predicted values are biased by the factor $\exp(\sigma^2/2)$. You can produce approximately unbiased predicted values in this case by writing the model as

```
proc model data=in;
   parms alpha beta;
   y=alpha * x ** beta;
   resid.y = log( actual.y / pred.y );
   fit y;
run;
```

See Miller (1984) for a discussion of bias factors for predicted values of transformed models.

Note that models with transformed errors are not appropriate for Monte Carlo simulation that uses the SDATA= option. PROC MODEL computes the OUTS= matrix from the transformed RESID.-prefixed equation variables, while it uses the SDATA= matrix to generate multivariate normal errors, which are added to the predicted values. This method of computing errors is inconsistent when the equation variables have been transformed.

Error Covariance Structure Specification

One of the key assumptions of regression is that the variance of the errors is constant across observations. Correcting for heteroscedasticity improves the efficiency of the estimates.

Consider the following general form for models:

$$
\begin{aligned}
\mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \theta) &= \boldsymbol{\varepsilon}_t \\
\boldsymbol{\varepsilon}_t &= H_t * \boldsymbol{\epsilon}_t \\
H_t &= \begin{bmatrix} \sqrt{h_{t,1}} & 0 & \dots & 0 \\ 0 & \sqrt{h_{t,2}} & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & \sqrt{h_{t,g}} \end{bmatrix} \\
\mathbf{h}_t &= \mathbf{g}(\mathbf{y}_t, \mathbf{x}_t, \phi)
\end{aligned}
$$

where $\boldsymbol{\epsilon}_t \sim N(0, \Sigma)$.

For models that are homoscedastic,

$$h_t = 1$$

If you have a model that is heteroscedastic with known form, you can improve the efficiency of the estimates by performing a weighted regression. The weight variable, using this notation, would be $1/\sqrt{h_t}$.

If the errors for a model are heteroscedastic and the functional form of the variance is known, the model for the variance can be estimated along with the regression function.

To specify a functional form for the variance, assign the function to an H.*var* variable where *var* is the equation variable. For example, if you want to estimate the scale parameter for the variance of a simple regression model

$$y = a * x + b$$

you can specify

```
proc model data=s;
   y = a * x + b;
   h.y = sigma**2;
fit y;
```

Consider the same model with the following functional form for the variance:

$$h_t = \sigma^2 * x^{2*\alpha}$$

This would be written as

```
proc model data=s;
   y = a * x + b;
   h.y = sigma**2 * x**(2*alpha);
fit y;
```

There are three ways to model the variance in the MODEL procedure: feasible generalized least squares, generalized method of moments, and full information maximum likelihood.

Feasible GLS

A simple approach to estimating a variance function is to estimate the mean parameters θ by using some auxiliary method, such as OLS, and then use the residuals of that estimation to estimate the parameters ϕ of the variance function. This scheme is called *feasible GLS*. It is possible to use the residuals from an auxiliary method for the purpose of estimating ϕ because in many cases the residuals consistently estimate the error terms.

For all estimation methods except GMM and FIML, using the H.var syntax specifies that feasible GLS is used in the estimation. For feasible GLS, the mean function is estimated by the usual method. The variance function is then estimated using pseudo-likelihood (PL) function of the generated residuals. The objective function for the PL estimation is

$$p_n(\sigma, \theta) = \sum_{i=1}^{n} \left(\frac{(y_i - f(x_i, \hat{\beta}))^2}{\sigma^2 h(z_i, \theta)} + \log[\sigma^2 h(z_i, \theta)] \right)$$

Once the variance function has been estimated, the mean function is reestimated by using the variance function as weights. If an S-iterated method is selected, this process is repeated until convergence (iterated feasible GLS).

Note that feasible GLS does not yield consistent estimates when one of the following is true:

- The variance is unbounded.
- There is too much serial dependence in the errors (the dependence does not fade with time).
- There is a combination of serial dependence and lag dependent variables.

The first two cases are unusual, but the third is much more common. Whether iterated feasible GLS avoids consistency problems with the last case is an unanswered research question. For more information see: Davidson and MacKinnon (1993, pp. 298–301); Gallant (1987, pp. 124–125); Amemiya (1985, pp. 202–203).

One limitation is that parameters cannot be shared between the mean equation and the variance equation. This implies that certain GARCH models, cross-equation restrictions of parameters, or testing of combinations of parameters in the mean and variance component are not allowed.

Generalized Method of Moments

In GMM, normally the first moment of the mean function is used in the objective function.

$$\begin{aligned} \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \theta) &= \epsilon_t \\ \mathbf{E}(\epsilon_t) &= 0 \end{aligned}$$

To add the second moment conditions to the estimation, add the equation

$$\mathbf{E}(\varepsilon_t * \varepsilon_t - h_t) = 0$$

to the model. For example, if you want to estimate σ for linear example above, you can write

```
proc model data=s;
   y = a * x + b;
   eq.two = resid.y**2 - sigma**2;
fit y two/ gmm;
instruments x;
run;
```

This is a popular way to estimate a continuous-time interest rate processes (see Chan et al. 1992). The H.var syntax automatically generates this system of equations.

To further take advantage of the information obtained about the variance, the moment equations can be modified to

$$\begin{aligned} \mathbf{E}(\varepsilon_t/\sqrt{h_t}) &= 0 \\ \mathbf{E}(\varepsilon_t * \varepsilon_t - h_t) &= 0 \end{aligned}$$

For the above example, this can be written as

```
proc model data=s;
   y = a * x + b;
   eq.two = resid.y**2 - sigma**2;
   resid.y = resid.y / sigma;
fit y two/ gmm;
instruments x;
run;
```

Note that, if the error model is misspecified in this form of the GMM model, the parameter estimates might be inconsistent.

Full Information Maximum Likelihood

For FIML estimation of variance functions, the concentrated likelihood below is used as the objective function. That is, the mean function is coupled with the variance function and the system is solved simultaneously.

$$\begin{aligned} l_n(\phi) &= \frac{ng}{2}(1+\ln(2\pi)) - \sum_{t=1}^{n} \ln\left(\left|\frac{\partial \mathbf{q}(\mathbf{y}_t, \mathbf{x}_t, \theta)}{\partial \mathbf{y}_t}\right|\right) \\ &\quad + \frac{1}{2}\sum_{t=1}^{n}\sum_{i=1}^{g}\left(\ln(h_{t,i}) + \mathbf{q}_i(\mathbf{y}_t, \mathbf{x}_t, \theta)^2/h_{t,i}\right) \end{aligned}$$

where g is the number of equations in the system.

The HESSIAN=GLS option is not available for FIML estimation that involves variance functions. The matrix used when HESSIAN=CROSS is specified is a crossproducts matrix that has been enhanced by the dual quasi-Newton approximation.

Examples

You can specify a GARCH(1,1) model as follows:

```
proc model data=modloc.usd_jpy;

             /* Mean model --------*/
   jpyret = intercept ;

             /* Variance model ----------------*/
   h.jpyret = arch0
            + arch1 * xlag( resid.jpyret ** 2, mse.jpyret  )
            + garch1 * xlag(h.jpyret, mse.jpyret) ;

   bounds arch0 arch1 garch1 >= 0;

fit jpyret / method=marquardt fiml;
run;
```

Note that the BOUNDS statement is used to ensure that the parameters are positive, a requirement for GARCH models.

EGARCH models are used because there are no restrictions on the parameters. You can specify a EGARCH(1,1) model as follows:

```
proc model data=sasuser.usd_dem ;

          /* Mean model ----------*/
  demret = intercept ;

             /* Variance model ----------------*/
  if ( _OBS_ =1 )  then
    h.demret = exp( earch0 + egarch1 * log(mse.demret)  );
  else
    h.demret = exp( earch0 + earch1 * zlag( g)
                          + egarch1 * log(zlag(h.demret)));
  g = - theta * nresid.demret + abs( nresid.demret ) - sqrt(2/3.1415);

fit demret / method=marquardt fiml  maxiter=100 converge=1.0e-6;
run;
```

Ordinary Differential Equations

Ordinary differential equations (ODEs) are also called *initial value problems* because a time zero value for each first-order differential equation is needed. The following is an example of a first-order system of ODEs:

$$
\begin{aligned}
y' &= -0.1y + 2.5z^2 \\
z' &= -z \\
y_0 &= 0 \\
z_0 &= 1
\end{aligned}
$$

Note that you must provide an initial value for each ODE.

As a reminder, any n-order differential equation can be modeled as a system of first-order differential equations. For example, consider the differential equations

$$\begin{aligned} y'' &= by' + cy \\ y_0 &= 0 \\ y'_0 &= 1 \end{aligned}$$

which can be written as the system of differential equations

$$\begin{aligned} y' &= z \\ z' &= by' + cy \\ y_0 &= 0 \\ z_0 &= 1 \end{aligned}$$

This differential system can be simulated as follows:

```
data t;
   time=0; output;
   time=1; output;
   time=2; output;
run;

proc model data=t ;
   dependent y 0 z 1;
   parm b -2 c -4;

   dert.y = z;
   dert.z = b * dert.y + c * y;

   solve y z / dynamic solveprint;
run;
```

The preceding statements produce the output shown in Figure 19.46. These statements produce additional output, which is not shown.

Figure 19.46 Simulation Results for Differential System

The MODEL Procedure
Simultaneous Simulation

Observation	1	Missing	2	CC	-1.000000
		Iterations	0		

Solution Values	
y	z
0.000000	1.000000

Figure 19.46 *continued*

Observation	2	Iterations	0	CC	0.000000	ERROR.y	0.000000

Solution Values	
y	z
0.2096398	-.2687053

Observation	3	Iterations	0	CC	9.464802	ERROR.y	-0.234405

Solution Values	
y	z
-.0247649	-.1035929

The differential variables are distinguished by the derivative with respect to time (DERT.) prefix. Once you define the DERT. variable, you can use it on the right-hand side of another equation. The differential equations must be expressed in normal form; implicit differential equations are not allowed, and other terms on the left-hand side are not allowed.

The TIME variable is the *implied with respect to* variable for all DERT. variables. The TIME variable is also the only variable that must be in the input data set.

You can provide initial values for the differential equations in the data set, in the declaration statement (as in the previous example), or in statements in the program. Using the previous example, you can specify the initial values as

```
proc model data=t ;
   dependent y z ;
   parm b -2 c -4;

   if ( time=0 ) then
      do;
         y=0;
         z=1;
      end;
   else
      do;
         dert.y = z;
         dert.z = b * dert.y + c * y;
      end;
   end;

   solve y z / dynamic solveprint;
run;
```

If you do not provide an initial value, 0 is used.

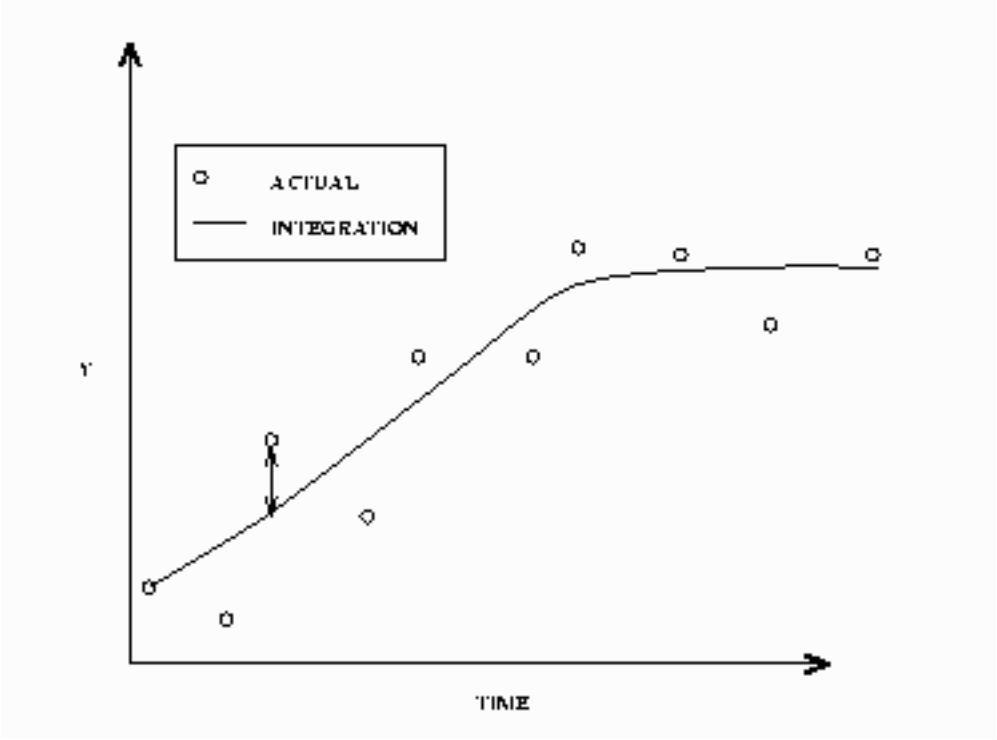
DYNAMIC and STATIC Simulation

Note that, in the previous example, the DYNAMIC option is specified in the SOLVE statement. The DYNAMIC and STATIC options work the same for differential equations as they do for dynamic systems. In the differential equation case, the DYNAMIC option makes the initial value needed at each observation the computed value from the previous iteration. For a static simulation, the data set must contain values for the integrated variables. For example, if DERT.Y and DERT.Z are the differential variables, you must include Y and Z in the input data set in order to do a static simulation of the model.

If the simulation is dynamic, the initial values for the differential equations are obtained from the data set, if they are available. If the variable is not in the data set, you can specify the initial value in a declaration statement. If you do not specify an initial value, the value of 0.0 is used.

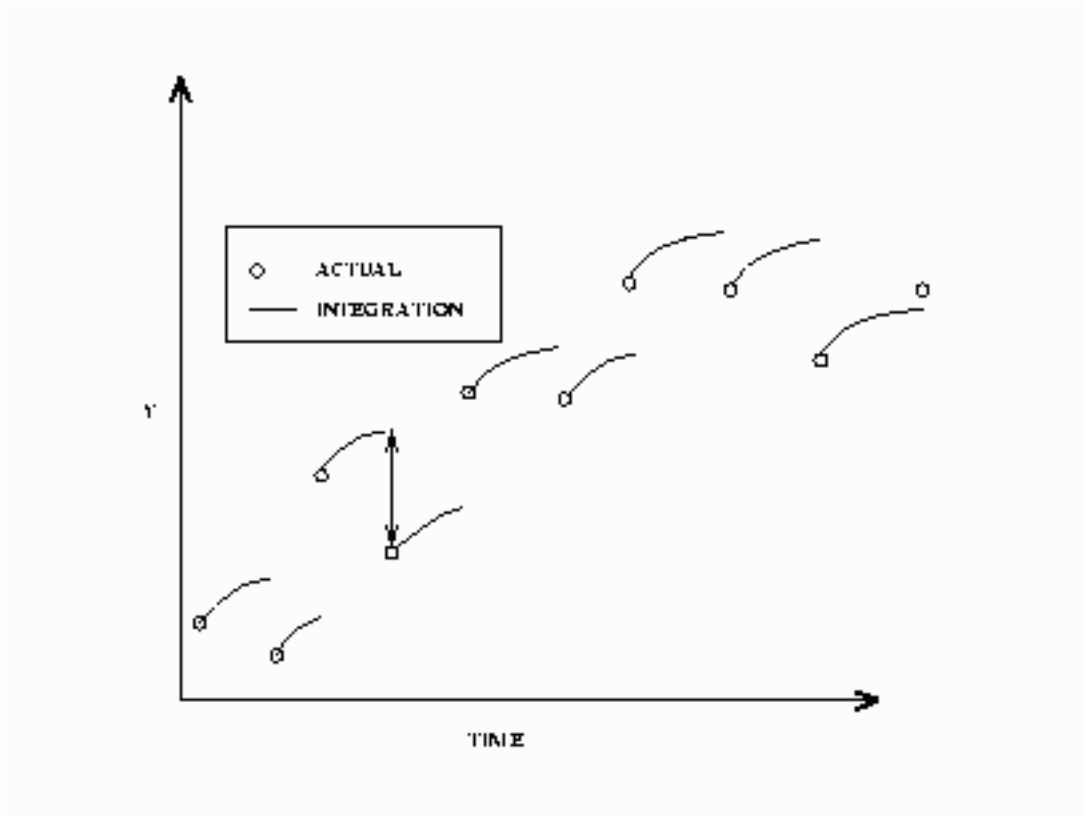
A dynamic solution is obtained by solving one initial value problem for all the data. A graph of a simple dynamic simulation is shown in Figure 19.47. If the time variable for the current observation is less than the time variable for the previous observation, the integration is restarted from this point. This allows for multiple samples in one data file.

Figure 19.47 Dynamic Solution



In a static solution, n–1 initial value problems are solved using the first n–1 data values as initial values. The equations are integrated using the *i* th data value as an initial value to the *i*+1 data value. Figure 19.48 displays a static simulation of noisy data from a simple differential equation. The static solution does not propagate errors in initial values as the dynamic solution does.

Figure 19.48 Static Solution



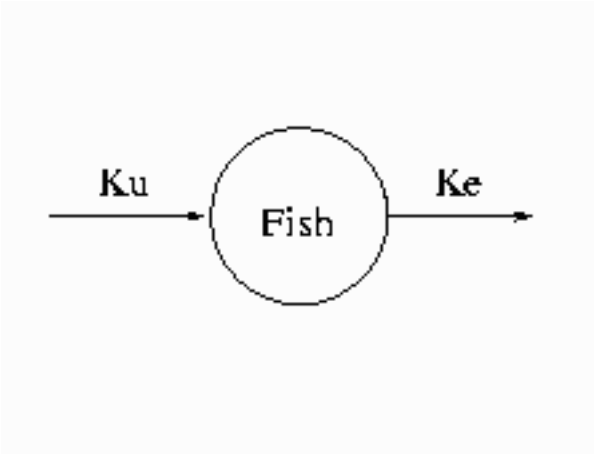
For estimation, the DYNAMIC and STATIC options in the FIT statement perform the same functions as they do in the SOLVE statement. Components of differential systems that have missing values or are not in the data set are simulated dynamically. For example, often in multiple compartment kinetic models, only one compartment is monitored. The differential equations that describe the unmonitored compartments are simulated dynamically.

For estimation, it is important to have accurate initial values for ODEs that are not in the data set. If an accurate initial value is not known, the initial value can be made an unknown parameter and estimated. This allows for errors in the initial values but increases the number of parameters to estimate by the number of equations.

Estimation of Differential Equations

Consider the kinetic model for the accumulation of mercury (Hg) in mosquito fish (Matis, Miller, and Allen 1991, p. 177). The model for this process is the one-compartment constant infusion model shown in Figure 19.49.

Figure 19.49 One-Compartment Constant Infusion Model



The differential equation that models this process is

$$\begin{aligned} \frac{dconc}{dt} &= k_u - k_e conc \\ conc_0 &= 0 \end{aligned}$$

The analytical solution to the model is

$$conc = (k_u/k_e)(1 - \exp(-k_e t))$$

The data for the model are

```
data fish;
   input day conc;
datalines;
0.0    0.0
1.0    0.15
2.0    0.2
3.0    0.26
4.0    0.32
6.0    0.33
;
```

To fit this model in differential form, use the following statements:

```
proc model data=fish;
   parm ku ke;

   dert.conc = ku - ke * conc;

   fit conc / time=day;
run;
```

The results from this estimation are shown in Figure 19.50.

Figure 19.50 Static Estimation Results for Fish Model

The MODEL Procedure

Nonlinear OLS Parameter Estimates				
Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
ku	0.180159	0.0312	5.78	0.0044
ke	0.524661	0.1181	4.44	0.0113

To perform a dynamic estimation of the differential equation, add the DYNAMIC option to the FIT statement.

```
proc model data=fish;
   parm ku .3 ke .3;

   dert.conc = ku - ke * conc;

   fit conc / time = day dynamic;
run;
```

The equation DERT.CONC is integrated from $conc(0) = 0$. The results from this estimation are shown in Figure 19.51.

Figure 19.51 Dynamic Estimation Results for Fish Model

The MODEL Procedure

Nonlinear OLS Parameter Estimates				
Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
ku	0.167109	0.0170	9.84	0.0006
ke	0.469033	0.0731	6.42	0.0030

To perform a dynamic estimation of the differential equation and estimate the initial value, use the following statements:

```
proc model data=fish;
   parm ku .3 ke .3 conc0 0;

   dert.conc = ku - ke * conc;

   fit conc initial=(conc = conc0) / time = day dynamic;
run;
```

The INITIAL= option in the FIT statement is used to associate the initial value of a differential equation with a parameter. The results from this estimation are shown in Figure 19.52.

Figure 19.52 Dynamic Estimation with Initial Value for Fish Model

The MODEL Procedure

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
ku	0.164408	0.0230	7.14	0.0057
ke	0.45949	0.0943	4.87	0.0165
conc0	0.003798	0.0174	0.22	0.8414

Finally, to estimate the fish model by using the analytical solution, use the following statements:

```
proc model data=fish;
   parm ku .3  ke .3;

   conc = (ku/ ke)*( 1 -exp(-ke * day));

   fit conc;
run;
```

The results from this estimation are shown in Figure 19.53.

Figure 19.53 Analytical Estimation Results for Fish Model

The MODEL Procedure

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
ku	0.167109	0.0170	9.84	0.0006
ke	0.469033	0.0731	6.42	0.0030

A comparison of the results among the four estimations reveals that the two dynamic estimations and the analytical estimation give nearly identical results (identical to the default precision). The two dynamic estimations are identical because the estimated initial value (0.00013071) is very close to the initial value used in the first dynamic estimation (0). Note also that the static model did not require an initial guess for the parameter values. Static estimation, in general, is more forgiving of bad initial values.

The form of the estimation that is preferred depends mostly on the model and data. If a very accurate initial value is known, then a dynamic estimation makes sense. If, additionally, the model can be written analytically, then the analytical estimation is computationally simpler. If only an approximate initial value is known and not modeled as an unknown parameter, the static estimation is less sensitive to errors in the initial value.

The form of the error in the model is also an important factor in choosing the form of the estimation. If the error term is additive and independent of previous error, then the dynamic mode is appropriate. If, on the other hand, the errors are cumulative, a static estimation is more appropriate. See the section "Monte Carlo Simulation" on page 1235 for an example.

Auxiliary Equations

Auxiliary equations can be used with differential equations. These are equations that need to be satisfied with the differential equations at each point between each data value. They are automatically added to the system, so you do not need to specify them in the SOLVE or FIT statement.

Consider the following example.

The Michaelis-Menten equations describe the kinetics of an enzyme-catalyzed reaction. The enzyme is E, and S is called the *substrate*. The enzyme first reacts with the substrate to form the enzyme-substrate complex ES, which then breaks down in a second step to form enzyme and products P.

The reaction rates are described by the following system of differential equations:

$$\frac{d[ES]}{dt} = k_1([E] - [ES])[S] - k_2[ES] - k_3[ES]$$
$$\frac{d[S]}{dt} = -k_1([E] - [ES])[S] + k_2[ES]$$
$$[E] = [E]_{tot} - [ES]$$

The first equation describes the rate of formation of ES from E + S. The rate of formation of ES from E + P is very small and can be ignored. The enzyme is in either the complexed or the uncomplexed form. So if the total ($[E]_{tot}$) concentration of enzyme and the amount bound to the substrate is known, $[E]$ can be obtained by conservation.

In this example, the conservation equation is an auxiliary equation and is coupled with the differential equations for integration.

Time Variable

You must provide a time variable in the data set. The name of the time variable defaults to TIME. You can use other variables as the time variable by specifying the TIME= option in the FIT or SOLVE statement. The time intervals need not be evenly spaced. If the time variable for the current observation is less than the time variable for the previous observation, the integration is restarted.

Differential Equations and Goal Seeking

Consider the following differential equation

$$y' = a*x$$

and the data set

```
data t2;
   y=0; time=0; output;
   y=2; time=1; output;
   y=3; time=2; output;
run;
```

The problem is to find values for X that satisfy the differential equation and the data in the data set. Problems of this kind are sometimes referred to as *goal-seeking problems* because they require you to search for values of X that satisfy the goal of Y.

This problem is solved with the following statements:

```
proc model data=t2;
   independent x 0;
   dependent y;
   parm a 5;
   dert.y = a * x;
   solve x / out=goaldata;
run;

proc print data=goaldata;
run;
```

The output from the PROC PRINT statement is shown in Figure 19.54.

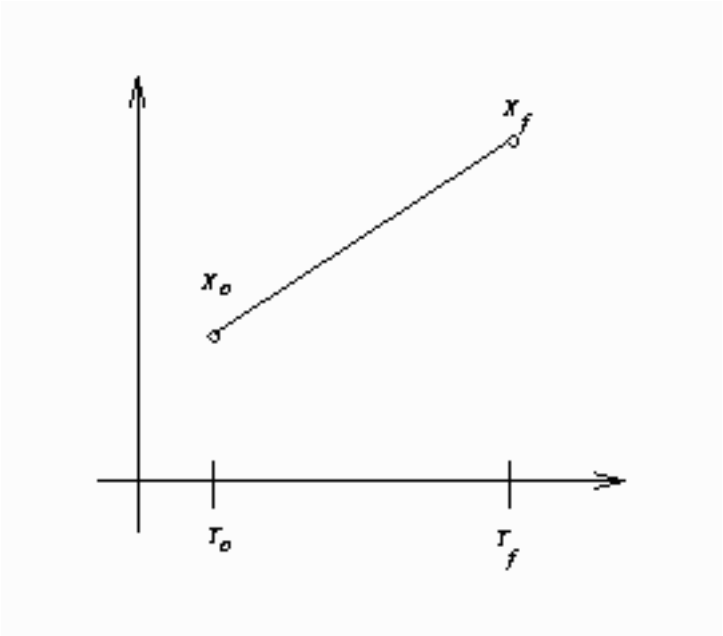
Figure 19.54 Dynamic Solution

Obs	_TYPE_	_MODE_	_ERRORS_	x	y	time
1	PREDICT	SIMULATE	0	0.0	0	0
2	PREDICT	SIMULATE	0	0.8	2	1
3	PREDICT	SIMULATE	0	-0.4	3	2

Note that an initial value of 0 is provided for the X variable because it is undetermined at TIME = 0.

In the preceding goal-seeking example, X is treated as a linear function between each set of data points (see Figure 19.55).

Figure 19.55 Form of X Used for Integration in Goal Seeking



If you integrate $y' = ax$ manually, you have

$$\begin{aligned} x(t) &= \frac{t_f - t}{t_f - t_o} x_o + \frac{t - t_o}{t_f - t_o} x_f \\ y_f - y_o &= \int_{t_o}^{t_f} ax(t)\, dt \\ &= a\frac{1}{t_f - t_o}(t(t_f x_o - t_o x_f) + \frac{1}{2}t^2(x_f - x_o))\Big|_{t_o}^{t_f} \end{aligned}$$

For observation 2, this reduces to

$$\begin{aligned} y_f - y_o &= \frac{1}{2} a * x_f \\ 2 &= 2.5 * x_f \end{aligned}$$

So $x = 0.8$ for this observation.

Goal seeking for the TIME variable is not allowed.

Restrictions and Bounds on Parameters

Using the BOUNDS and RESTRICT statements, PROC MODEL can compute optimal estimates subject to equality or inequality constraints on the parameter estimates.

Equality restrictions can be written as a vector function:

$$\mathbf{h}(\theta) = 0$$

Inequality restrictions are either active or inactive. When an inequality restriction is active, it is treated as an equality restriction. All inactive inequality restrictions can be written as a vector function:

$$F(\theta) \geq 0$$

Strict inequalities, such as $(f(\theta) > 0)$, are transformed into inequalities as $f(\theta) \times (1 - \epsilon) - \epsilon \geq 0$, where the tolerance ϵ is controlled by the EPSILON= option in the FIT statement and defaults to 10^{-8}. The ith inequality restriction becomes active if $F_i < 0$ and remains active until its Lagrange multiplier becomes negative. Lagrange multipliers are computed for all the nonredundant equality restrictions and all the active inequality restrictions.

For the following, assume the vector $\mathbf{h}(\theta)$ contains all the current active restrictions. The constraint matrix $\mathbf{A}$ is

$$\mathbf{A}(\hat{\theta}) = \frac{\partial \mathbf{h}(\hat{\theta})}{\partial \hat{\theta}}$$

The covariance matrix for the restricted parameter estimates is computed as

$$\mathbf{Z}(\mathbf{Z}'\mathbf{H}\mathbf{Z})^{-1}\mathbf{Z}'$$

where **H** is Hessian or approximation to the Hessian of the objective function ($(\mathbf{X}'(\mathrm{diag}(\mathbf{S})^{-1}\otimes\mathbf{I})\mathbf{X})$ for OLS), and **Z** is the last $(np - nc)$ columns of **Q**. **Q** is from an LQ factorization of the constraint matrix, nc is the number of active constraints, and np is the number of parameters. For more details on LQ factorization, see Gill, Murray, and Wright (1981). The covariance column in Table 19.2 summarizes the Hessian approximation used for each estimation method.

The covariance matrix for the Lagrange multipliers is computed as

$$(\mathbf{A}\mathbf{H}^{-1}\mathbf{A}')^{-1}$$

The p-value reported for a restriction is computed from a beta distribution rather than a t distribution because the numerator and the denominator of the t ratio for an estimated Lagrange multiplier are not independent.

The Lagrange multipliers for the active restrictions are printed with the parameter estimates. The Lagrange multiplier estimates are computed using the relationship

$$\mathbf{A}'\lambda = \mathrm{g}$$

where the dimensions of the constraint matrix **A** are the number of constraints by the number of parameters, λ is the vector of Lagrange multipliers, and g is the gradient of the objective function at the final estimates.

The final gradient includes the effects of the estimated **S** matrix. For example, for OLS the final gradient would be:

$$\mathrm{g} = \mathbf{X}'(\mathrm{diag}(\mathbf{S})^{-1}\otimes\mathbf{I})r$$

where r is the residual vector. Note that when nonlinear restrictions are imposed, the convergence measure R might have values greater than one for some iterations.

Tests on Parameters

In general, the hypothesis tested can be written as

$$H_0 : \mathbf{h}(\theta) = 0$$

where $\mathbf{h}(\theta)$ is a vector-valued function of the parameters θ given by the r expressions specified on the TEST statement.

Let $\hat{\mathbf{V}}$ be the estimate of the covariance matrix of $\hat{\theta}$. Let $\hat{\theta}$ be the unconstrained estimate of θ and $\tilde{\theta}$ be the constrained estimate of θ such that $h(\tilde{\theta}) = 0$. Let

$$\mathbf{A}(\theta) = \partial h(\theta)/\partial\theta \mid_{\hat{\theta}}$$

Let r be the dimension of $h(\theta)$ and n be the number of observations. Using this notation, the test statistics for the three kinds of tests are computed as follows.

The Wald test statistic is defined as

$$W = h'(\hat{\theta})\left(\mathbf{A}(\hat{\theta})\hat{\mathbf{V}}\mathbf{A}'(\hat{\theta})\right)^{-1}h(\hat{\theta})$$

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

$$R = \lambda' \mathbf{A}(\tilde{\theta})\tilde{\mathbf{V}}\mathbf{A}'(\tilde{\theta})\lambda$$

where λ is the vector of Lagrange multipliers from the computation of the restricted estimate $\tilde{\theta}$.

The Lagrange multiplier test statistic is equivalent to Rao's efficient score test statistic:

$$R = (\partial L(\tilde{\theta})/\partial\theta)'\tilde{\mathbf{V}}(\partial L(\tilde{\theta})/\partial\theta)$$

where L is the log-likelihood function for the estimation method used. For SUR, 3SLS, GMM, and iterated versions of these methods, the likelihood function is computed as

$$L = \text{Objective} \times Nobs/2$$

For OLS and 2SLS, the Lagrange multiplier test statistic is computed as:

$$R = [(\partial \hat{S}(\tilde{\theta})/\partial\theta)'\tilde{\mathbf{V}}(\partial \hat{S}(\tilde{\theta})/\partial\theta)]/\hat{S}(\tilde{\theta})$$

where $\hat{S}(\tilde{\theta})$ is the corresponding objective function value at the constrained estimate.

The likelihood ratio test statistic is

$$T = 2\left(L(\hat{\theta}) - L(\tilde{\theta})\right)$$

where $\tilde{\theta}$ represents the constrained estimate of θ and L is the concentrated log-likelihood value.

For OLS and 2SLS, the likelihood ratio test statistic is computed as:

$$T = (n - nparms) \times (\hat{S}(\tilde{\theta}) - \hat{S}(\hat{\theta}))/\hat{S}(\hat{\theta})$$

This test statistic is an approximation from

$$T = n \times log\left(1 + \frac{rF}{n - nparms}\right)$$

when the value of $rF/(n - nparms)$ is small (Greene 2004, p. 421).

The likelihood ratio test is not appropriate for models with nonstationary serially correlated errors (Gallant 1987, p. 139). The likelihood ratio test should not be used for dynamic systems, for systems with lagged dependent variables, or with the FIML estimation method unless certain conditions are met (see Gallant 1987, p. 479).

For each kind 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 reported for the tests are computed from the $\chi^2(r)$ distribution and are only asymptotically valid. When both RESTRICT and TEST statements are used in a PROC MODEL step, test statistics are computed by taking into account the constraints imposed by the RESTRICT statement.

Monte Carlo simulations suggest that the asymptotic distribution of the Wald test is a poorer approximation to its small sample distribution than the other two tests. However, the Wald test has the least computational cost, since it does not require computation of the constrained estimate $\tilde{\theta}$.

The following is an example of using the TEST statement to perform a likelihood ratio test for a compound hypothesis.

```
test a*exp(-k) = 1-k, d = 0 ,/ lr;
```

It is important to keep in mind that although individual t tests for each parameter are printed by default into the parameter estimates table, they are only asymptotically valid for nonlinear models. You should be cautious in drawing any inferences from these t tests for small samples.

Hausman Specification Test

Hausman's specification test, or m-statistic, can be used to test hypotheses in terms of bias or inconsistency of an estimator. This test was also proposed by Wu (1973). Hausman's m-statistic is as follows.

Given two estimators, $\hat{\beta}_0$ and $\hat{\beta}_1$, where under the null hypothesis both estimators are consistent but only $\hat{\beta}_0$ is asymptotically efficient and under the alternative hypothesis only $\hat{\beta}_1$ is consistent, the m-statistic is

$$m = \hat{q}'(\hat{\mathbf{V}}_1 - \hat{\mathbf{V}}_0)^{-}\hat{q}$$

where $\hat{\mathbf{V}}_1$ and $\hat{\mathbf{V}}_0$ represent consistent estimates of the asymptotic covariance matrices of $\hat{\beta}_1$ and $\hat{\beta}_0$ respectively, and

$$q = \hat{\beta}_1 - \hat{\beta}_0$$

The m-statistic is then distributed χ^2 with k degrees of freedom, where k is the rank of the matrix $(\hat{\mathbf{V}}_1 - \hat{\mathbf{V}}_0)$. A generalized inverse is used, as recommended by Hausman and Taylor (1982).

In the MODEL procedure, Hausman's m-statistic can be used to determine if it is necessary to use an instrumental variables method rather than a more efficient OLS estimation. Hausman's m-statistic can also be used to compare 2SLS with 3SLS for a class of estimators for which 3SLS is asymptotically efficient (similarly for OLS and SUR).

Hausman's m-statistic can also be used, in principle, to test the null hypothesis of normality when comparing 3SLS to FIML. Because of the poor performance of this form of the test, it is not offered in the MODEL procedure. See Fair (1984, pp. 246–247) for a discussion of why Hausman's test fails for common econometric models.

To perform a Hausman's specification test, specify the HAUSMAN option in the FIT statement. The selected estimation methods are compared using Hausman's m-statistic.

In the following example, Hausman's test is used to check the presence of measurement error. Under H_0 of no measurement error, OLS is efficient, while under H_1, 2SLS is consistent. In the following code, OLS and 2SLS are used to estimate the model, and Hausman's test is requested.

```
proc model data=one out=fiml2;
   endogenous y1 y2;

   y1 = py2 * y2 + px1 * x1 + interc;
   y2 = py1* y1 + pz1 * z1 + d2;

   fit y1 y2 / ols 2sls hausman;
   instruments x1 z1;
run;
```

The output specified by the HAUSMAN option produces the results shown in Figure 19.56.

Figure 19.56 Hausman's Specification Test Results

The MODEL Procedure

Hausman's Specification Test Results				
Efficient under H0	**Consistent under H1**	**DF**	**Statistic**	**Pr > ChiSq**
OLS	2SLS	6	13.86	0.0313

Figure 19.56 indicates that 2SLS is preferred over OLS at 5% level of significance. In this case, the null hypothesis of no measurement error is rejected. Hence, the instrumental variable estimator is required for this example due to the presence of measurement error.

Chow Tests

The Chow test is used to test for break points or structural changes in a model. The problem is posed as a partitioning of the data into two parts of size n_1 and n_2. The null hypothesis to be tested is

$$H_o : \beta_1 = \beta_2 = \beta$$

where β_1 is estimated by using the first part of the data and β_2 is estimated by using the second part.

The test is performed as follows (see Davidson and MacKinnon 1993, p. 380).

1. The p parameters of the model are estimated.
2. A second linear regression is performed on the residuals, $\hat{u}$, from the nonlinear estimation in step one.

   $$\hat{u} = \hat{\mathbf{X}}b + \text{residuals}$$

   where $\hat{\mathbf{X}}$ is Jacobian columns that are evaluated at the parameter estimates. If the estimation is an instrumental variables estimation with matrix of instruments $\mathbf{W}$, then the following regression is performed:

   $$\hat{u} = \mathbf{P}_{W^*}\hat{\mathbf{X}}b + \text{residuals}$$

   where $\mathbf{P}_{W^*}$ is the projection matrix.
3. The restricted SSE (RSSE) from this regression is obtained. An SSE for each subsample is then obtained by using the same linear regression.
4. The F statistic is then

   $$f = \frac{(RSSE - SSE_1 - SSE_2)/p}{(SSE_1 + SSE_2)/(n - 2p)}$$

   This test has p and $n - 2p$ degrees of freedom.

Chow's test is not applicable if $\min(n_1, n_2) < p$, since one of the two subsamples does not contain enough data to estimate β. In this instance, the *predictive Chow test* can be used. The predictive Chow test is defined as

$$f = \frac{(RSSE - SSE_1)\times(n_1 - p)}{SSE_1 * n_2}$$

where $n_1 > p$. This test can be derived from the Chow test by noting that the $SSE_2 = 0$ when $n_2 <= p$ and by adjusting the degrees of freedom appropriately.

You can select the Chow test and the predictive Chow test by specifying the CHOW=*arg* and the PCHOW=*arg* options in the FIT statement, where *arg* is either the number of observations in the first sample or a parenthesized list of first sample sizes. If the size of the one of the two groups in which the sample is partitioned is less than the number of parameters, then a predictive Chow test is automatically used. These tests statistics are not produced for GMM and FIML estimations.

The following is an example of the use of the Chow test.

```
data exp;
   x=0;
   do time=1 to 100;
      if time=50 then x=1;
      y = 35 * exp( 0.01 * time ) + rannor( 123 ) + x * 5;
      output;
   end;
run;

proc model data=exp;
   parm zo 35 b;
      dert.z = b * z;
      y=z;
   fit y init=(z=zo) / chow =(40 50 60) pchow=90;
run;
```

The data set introduces an artificial structural change into the model (the structural change effects the intercept parameter). The output from the requested Chow tests are shown in Figure 19.57.

Figure 19.57 Chow's Test Results

The MODEL Procedure

Structural Change Test					
Test	Break Point	Num DF	Den DF	F Value	Pr > F
Chow	40	2	96	12.95	<.0001
Chow	50	2	96	101.37	<.0001
Chow	60	2	96	26.43	<.0001
Predictive Chow	90	11	87	1.86	0.0566

Profile Likelihood Confidence Intervals

Wald-based and likelihood-ratio-based confidence intervals are available in the MODEL procedure for computing a confidence interval on an estimated parameter. A confidence interval on a parameter θ can be constructed by inverting a Wald-based or a likelihood-ratio-based test.

The approximate $100(1-\alpha)$ % Wald confidence interval for a parameter θ is

$$\hat{\theta} \pm z_{1-\alpha/2}\hat{\sigma}$$

where z_p is the $100p$th percentile of the standard normal distribution, $\hat{\theta}$ is the maximum likelihood estimate of θ, and $\hat{\sigma}$ is the standard error estimate of $\hat{\theta}$.

A likelihood-ratio-based confidence interval is derived from the χ^2 distribution of the generalized likelihood ratio test. The approximate $1-\alpha$ confidence interval for a parameter θ is

$$\theta : 2[l(\hat{\theta}) - l(\theta)] \leq q_{1,1-\alpha} = 2l^*$$

where $q_{1,1-\alpha}$ is the $(1-\alpha)$ quantile of the χ^2 with one degree of freedom, and $l(\theta)$ is the log likelihood as a function of one parameter. The endpoints of a confidence interval are the zeros of the function $l(\theta) - l^*$. Computing a likelihood-ratio-based confidence interval is an iterative process. This process must be performed twice for each parameter, so the computational cost is considerable. Using a modified form of the algorithm recommended by Venzon and Moolgavkar (1988), you can determine that the cost of each endpoint computation is approximately the cost of estimating the original system.

To request confidence intervals on estimated parameters, specify the PRL= option in the FIT statement. By default, the PRL option produces 95% likelihood ratio confidence limits. The coverage of the confidence interval is controlled by the ALPHA= option in the FIT statement.

The following is an example of the use of the confidence interval options.

```
data exp;
   do time = 1 to 20;
      y = 35 * exp( 0.01 * time ) + 5*rannor( 123 );
   output;
   end;
run;

proc model data=exp;
   parm zo 35 b;
      dert.z = b * z;
      y=z;
   fit y init=(z=zo) / prl=both;
   test zo = 40.475437 ,/ lr;
run;
```

The output from the requested confidence intervals and the TEST statement are shown in Figure 19.58

Figure 19.58 Confidence Interval Estimation

The MODEL Procedure

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
zo	36.58933	1.9471	18.79	<.0001
b	0.006497	0.00464	1.40	0.1780

Test Results

Test	Type	Statistic	Pr > ChiSq	Label
Test0	L.R.	3.81	0.0509	zo = 40.475437

Figure 19.58 *continued*

Parameter Wald 95% Confidence Intervals			
Parameter	**Value**	**Lower**	**Upper**
zo	36.5893	32.7730	40.4056
b	0.00650	-0.00259	0.0156

Parameter Likelihood Ratio 95% Confidence Intervals			
Parameter	**Value**	**Lower**	**Upper**
zo	36.5893	32.8381	40.4921
b	0.00650	-0.00264	0.0157

In this example the parameter value used in the likelihood ratio test, $zo = 40.475437$, is close to the upper bound computed for the likelihood ratio confidence interval, $zo \leq 40.4921$. This coincidence is not germane to the analysis however, since the likelihood ratio test is a test of the null hypothesis $H_0 : zo = 40.475437$ and the confidence interval can be viewed as a test of the null hypothesis $H_0 : 32.8381 \leq zo \leq 40.4921$.

Choice of Instruments

Several of the estimation methods supported by PROC MODEL are instrumental variables methods. There is no standard method for choosing instruments for nonlinear regression. Few econometric textbooks discuss the selection of instruments for nonlinear models. See Bowden and Turkington (1984, pp. 180–182) for more information.

The purpose of the instrumental projection is to purge the regressors of their correlation with the residual. For nonlinear systems, the regressors are the partials of the residuals with respect to the parameters.

Possible instrumental variables include the following:

- any variable in the model that is independent of the errors
- lags of variables in the system
- derivatives with respect to the parameters, if the derivatives are independent of the errors
- low-degree polynomials in the exogenous variables
- variables from the data set or functions of variables from the data set

Selected instruments must not have any of the following characteristics:

- depend on any variable endogenous with respect to the equations estimated
- depend on any of the parameters estimated
- be lags of endogenous variables if there is serial correlation of the errors

If the preceding rules are satisfied and there are enough observations to support the number of instruments used, the results should be consistent and the efficiency loss held to a minimum.

You need at least as many instruments as the maximum number of parameters in any equation, or some of the parameters cannot be estimated. Note that *number of instruments* means linearly independent instruments. If you add an instrument that is a linear combination of other instruments, it has no effect and does not increase the effective number of instruments.

You can, however, use too many instruments. In order to get the benefit of instrumental variables, you must have more observations than instruments. Thus, there is a trade-off; the instrumental variables technique completely eliminates the simultaneous equation bias only in large samples. In finite samples, the larger the excess of observations over instruments, the more the bias is reduced. Adding more instruments might improve the efficiency, but after some point efficiency declines as the excess of observations over instruments becomes smaller and the bias grows.

The instruments used in an estimation are printed out at the beginning of the estimation. For example, the following statements produce the instruments list shown in Figure 19.59.

```
proc model data=test2;
   exogenous x1 x2;
   parms b1 a1 a2 b2 2.5 c2 55;
   y1 = a1 * y2 + b1 * exp(x1);
   y2 = a2 * y1 + b2 * x2 * x2 + c2 / x2;
   fit y1 y2 / n2sls;
   inst b1 b2 c2 x1 ;
run;
```

Figure 19.59 Instruments Used Message

The MODEL Procedure

The 2 Equations to Estimate	
y1 =	F(b1, a1(y2))
y2 =	F(a2(y1), b2, c2)
Instruments	1 x1 @y1/@b1 @y2/@b2 @y2/@c2

This states that an intercept term, the exogenous variable X1, and the partial derivatives of the equations with respect to B1, B2, and C2, were used as instruments for the estimation.

Examples

Suppose that Y1 and Y2 are endogenous variables, that X1 and X2 are exogenous variables, and that A, B, C, D, E, F, and G are parameters. Consider the following model:

```
y1 = a + b * x1 + c * y2 + d * lag(y1);
y2 = e + f * x2 + g * y1;
fit y1 y2;
instruments exclude=(c g);
```

The INSTRUMENTS statement produces X1, X2, LAG(Y1), and an intercept as instruments.

In order to estimate the Y1 equation by itself, it is necessary to include X2 explicitly in the instruments since F, in this case, is not included in the following estimation:

```
y1 = a + b * x1 + c * y2 + d * lag(y1);
y2 = e + f * x2 + g * y1;
fit y1;
instruments x2 exclude=(c);
```

This produces the same instruments as before. You can list the parameter associated with the lagged variable as an instrument instead of using the EXCLUDE= option. Thus, the following is equivalent to the previous example:

```
y1 = a + b * x1 + c * y2 + d * lag(y1);
y2 = e + f * x2 + g * y1;
fit y1;
instruments x1 x2 d;
```

For an example of declaring instruments when estimating a model involving identities, consider Klein's Model I:

```
proc model data=klien;
   endogenous c p w i x wsum k y;
   exogenous  wp g t year;
   parms c0-c3 i0-i3 w0-w3;
   a: c = c0 + c1 * p + c2 * lag(p) + c3 * wsum;
   b: i = i0 + i1 * p + i2 * lag(p) + i3 * lag(k);
   c: w = w0 + w1 * x + w2 * lag(x) + w3 * year;
   x = c + i + g;
   y = c + i + g-t;
   p = x-w-t;
   k = lag(k) + i;
   wsum = w + wp;
run;
```

The three equations to estimate are identified by the labels A, B, and C. The parameters associated with the predetermined terms are C2, I2, I3, W2, and W3 (and the intercepts, which are automatically added to the instruments). In addition, the system includes five identities that contain the predetermined variables G, T, LAG(K), and WP. Thus, the INSTRUMENTS statement can be written as

```
lagk = lag(k);
instruments c2 i2 i3 w2 w3 g t wp lagk;
```

where LAGK is a program variable used to hold LAG(K). However, this is more complicated than it needs to be. Except for LAG(K), all the predetermined terms in the identities are exogenous variables, and LAG(K) is already included as the coefficient of I3. There are also more parameters for predetermined terms than for endogenous terms, so you might prefer to use the EXCLUDE= option. Thus, you can specify the same instruments list with the simpler statement

```
instruments _exog_ exclude=(c1 c3 i1 w1);
```

To illustrate the use of polynomial terms as instrumental variables, consider the following model:

```
y1 = a + b * exp( c * x1 ) + d * log( x2 ) + e * exp( f * y2 );
```

The parameters are A, B, C, D, E, and F, and the right-hand-side variables are X1, X2, and Y2. Assume that X1 and X2 are exogenous (independent of the error), while Y2 is endogenous. The equation for Y2 is not specified, but assume that it includes the variables X1, X3, and Y1, with X3 exogenous, so the exogenous variables of the full system are X1, X2, and X3. Using as instruments quadratic terms in the exogenous variables, the model is specified to PROC MODEL as follows:

```
proc model;
   parms a b c d e f;
   y1 = a + b * exp( c * x1 ) + d * log( x2 ) + e * exp( f * y2 );
   instruments inst1-inst9;
   inst1 = x1; inst2 = x2; inst3 = x3;
   inst4 = x1 * x1; inst5 = x1 * x2; inst6 = x1 * x3;
   inst7 = x2 * x2; inst8 = x2 * x3; inst9 = x3 * x3;
   fit y1 / 2sls;
run;
```

It is not clear what degree polynomial should be used. There is no way to know how good the approximation is for any degree chosen, although the first-stage R^2s might help the assessment.

First-Stage R-Squares

When the FSRSQ option is used on the FIT statement, the MODEL procedure prints a column of first-stage R^2 (FSRSQ) statistics along with the parameter estimates. The FSRSQ measures the fraction of the variation of the derivative column associated with the parameter that remains after projection through the instruments.

Ideally, the FSRSQ should be very close to 1.00 for exogenous derivatives. If the FSRSQ is small for an endogenous derivative, it is unclear whether this reflects a poor choice of instruments or a large influence of the errors in the endogenous right-hand-side variables. When the FSRSQ for one or more parameters is small, the standard errors of the parameter estimates are likely to be large.

Note that you can make all the FSRSQs larger (or 1.00) by including more instruments, because of the disadvantage discussed previously. The FSRSQ statistics reported are unadjusted R^2s and do not include a degrees-of-freedom correction.

Autoregressive Moving-Average Error Processes

Autoregressive moving-average error processes (ARMA errors) and other models that involve lags of error terms can be estimated by using FIT statements and simulated or forecast by using SOLVE statements. ARMA models for the error process are often used for models with autocorrelated residuals. The %AR macro can be used to specify models with autoregressive error processes. The %MA macro can be used to specify models with moving-average error processes.

Autoregressive Errors

A model with first-order autoregressive errors, AR(1), has the form

$$y_t = f(x_t, \theta) + \mu_t$$

$$\mu_t = \phi \mu_{t-1} + \epsilon_t$$

while an AR(2) error process has the form

$$\mu_t = \phi_1 \mu_{t-1} + \phi_2 \mu_{t-2} + \epsilon_t$$

and so forth for higher-order processes. Note that the ϵ_t's are independent and identically distributed and have an expected value of 0.

An example of a model with an AR(2) component is

$$y = \alpha + \beta x_1 + \mu_t$$

$$\mu_t = \phi_1 \mu_{t-1} + \phi_2 \mu_{t-2} + \epsilon_t$$

You would write this model as

```
proc model data=in;
   parms a b p1 p2;
   y = a + b * x1 + p1 * zlag1(y - (a + b * x1)) +
         p2 * zlag2(y - (a + b * x1));
   fit y;
run;
```

or equivalently using the %AR macro as

```
proc model data=in;
   parms a b;
   y = a + b * x1;
   %ar( y, 2 );
   fit y;
run;
```

Moving-Average Models

A model with first-order moving-average errors, MA(1), has the form

$$y_t = f(x_t) + \mu_t$$

$$\mu_t = \epsilon_t - \theta_1 \epsilon_{t-1}$$

where ϵ_t is identically and independently distributed with mean zero. An MA(2) error process has the form

$$\mu_t = \epsilon_t - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2}$$

and so forth for higher-order processes.

For example, you can write a simple linear regression model with MA(2) moving-average errors as

```
proc model data=inma2;
   parms a b ma1 ma2;
   y = a + b * x + ma1 * zlag1( resid.y ) +
       ma2 * zlag2( resid.y );
   fit;
run;
```

where MA1 and MA2 are the moving-average parameters.

Note that RESID.Y is automatically defined by PROC MODEL as

```
pred.y = a + b * x + ma1 * zlag1( resid.y ) +
       ma2 * zlag2( resid.y );
resid.y = pred.y - actual.y;
```

Note that RESID.Y is negative of ϵ_t.

The ZLAG function must be used for MA models to truncate the recursion of the lags. This ensures that the lagged errors start at zero in the lag-priming phase and do not propagate missing values when lag-priming period variables are missing, and it ensures that the future errors are zero rather than missing during simulation or forecasting. For details about the lag functions, see the section "Lag Logic" on page 1277.

This model written using the %MA macro is as follows:

```
proc model data=inma2;
   parms a b;
   y =  a + b * x;
   %ma(y, 2);
   fit;
run;
```

General Form for ARMA Models

The general ARMA(p,q) process has the following form

$$\mu_t = \phi_1\mu_{t-1} + \ldots + \phi_p\mu_{t-p} + \epsilon_t - \theta_1\epsilon_{t-1} - \ldots - \theta_q\epsilon_{t-q}$$

An ARMA(p,q) model can be specified as follows:

```
yhat =  ... compute structural predicted value here ... ;
yarma = ar1 * zlag1( y - yhat ) + ...  /* ar part */
                                + ar(p) * zlag(p)( y - yhat )
     + ma1 * zlag1( resid.y )   + ...  /* ma part */
                                + ma(q) * zlag(q)( resid.y );
y = yhat + yarma;
```

where ARi and MAj represent the autoregressive and moving-average parameters for the various lags. You can use any names you want for these variables, and there are many equivalent ways that the specification could be written.

Vector ARMA processes can also be estimated with PROC MODEL. For example, a two-variable AR(1) process for the errors of the two endogenous variables Y1 and Y2 can be specified as follows:

```
y1hat =  ... compute structural predicted value here ... ;

y1    = y1hat + ar1_1 * zlag1( y1 - y1hat )   /* ar part y1,y1 */
              + ar1_2 * zlag1( y2 - y2hat );  /* ar part y1,y2 */

y21hat =  ... compute structural predicted value here ... ;

y2     = y2hat + ar2_2 * zlag1( y2 - y2hat )   /* ar part y2,y2 */
               + ar2_1 * zlag1( y1 - y1hat );  /* ar part y2,y1 */
```

Convergence Problems with ARMA Models

ARMA models can be difficult to estimate. If the parameter estimates are not within the appropriate range, a moving-average model's residual terms grow exponentially. The calculated residuals for later observations can be very large or can overflow. This can happen either because improper starting values were used or because the iterations moved away from reasonable values.

Care should be used in choosing starting values for ARMA parameters. Starting values of 0.001 for ARMA parameters usually work if the model fits the data well and the problem is well-conditioned. Note that an MA model can often be approximated by a high-order AR model, and vice versa. This can result in high collinearity in mixed ARMA models, which in turn can cause serious ill-conditioning in the calculations and instability of the parameter estimates.

If you have convergence problems while estimating a model with ARMA error processes, try to estimate in steps. First, use a FIT statement to estimate only the structural parameters with the ARMA parameters held to zero (or to reasonable prior estimates if available). Next, use another FIT statement to estimate the ARMA parameters only, using the structural parameter values from the first run. Since the values of the structural parameters are likely to be close to their final estimates, the ARMA parameter estimates might now converge. Finally, use another FIT statement to produce simultaneous estimates of all the parameters. Since the initial values of the parameters are now likely to be quite close to their final joint estimates, the estimates should converge quickly if the model is appropriate for the data.

AR Initial Conditions

The initial lags of the error terms of AR(p) models can be modeled in different ways. The autoregressive error startup methods supported by SAS/ETS procedures are the following:

CLS	conditional least squares (ARIMA and MODEL procedures)
ULS	unconditional least squares (AUTOREG, ARIMA, and MODEL procedures)
ML	maximum likelihood (AUTOREG, ARIMA, and MODEL procedures)
YW	Yule-Walker (AUTOREG procedure only)
HL	Hildreth-Lu, which deletes the first p observations (MODEL procedure only)

See Chapter 8, "The AUTOREG Procedure," for an explanation and discussion of the merits of various AR(p) startup methods.

The CLS, ULS, ML, and HL initializations can be performed by PROC MODEL. For AR(1) errors, these initializations can be produced as shown in Table 19.3. These methods are equivalent in large samples.

Table 19.3 Initializations Performed by PROC MODEL: AR(1) ERRORS

Method	Formula
conditional least squares	Y=YHAT+AR1*ZLAG1(Y-YHAT);
unconditional least squares	Y=YHAT+AR1*ZLAG1(Y-YHAT); IF _OBS_=1 THEN RESID.Y=SQRT(1-AR1**2)*RESID.Y;
maximum likelihood	Y=YHAT+AR1*ZLAG1(Y-YHAT); W=(1-AR1**2)**(-1/(2*_NUSED_)); IF _OBS_=1 THEN W=W*SQRT(1-AR1**2); RESID.Y=W*RESID.Y;
Hildreth-Lu	Y=YHAT+AR1*LAG1(Y-YHAT);

MA Initial Conditions

The initial lags of the error terms of MA(q) models can also be modeled in different ways. The following moving-average error start-up paradigms are supported by the ARIMA and MODEL procedures:

ULS	unconditional least squares
CLS	conditional least squares
ML	maximum likelihood

The conditional least squares method of estimating moving-average error terms is not optimal because it ignores the start-up problem. This reduces the efficiency of the estimates, although they remain unbiased. The initial lagged residuals, extending before the start of the data, are assumed to be 0, their unconditional expected value. This introduces a difference between these residuals and the generalized least squares residuals for the moving-average covariance, which, unlike the autoregressive model, persists through the data set. Usually this difference converges quickly to 0, but for nearly noninvertible moving-average processes the convergence is quite slow. To minimize this problem, you should have plenty of data, and the moving-average parameter estimates should be well within the invertible range.

This problem can be corrected at the expense of writing a more complex program. Unconditional least squares estimates for the MA(1) process can be produced by specifying the model as follows:

```
yhat =  ... compute structural predicted value here ... ;
if _obs_ = 1 then do;
   h = sqrt( 1 + ma1 ** 2 );
   y = yhat;
   resid.y = ( y - yhat ) / h;
   end;
else do;
   g = ma1 / zlag1( h );
   h = sqrt( 1 + ma1 ** 2 - g ** 2 );
   y = yhat + g * zlag1( resid.y );
   resid.y = ( ( y - yhat) - g * zlag1( resid.y ) ) / h;
   end;
```

Moving-average errors can be difficult to estimate. You should consider using an AR(p) approximation to the moving-average process. A moving-average process can usually be well-approximated by an autoregressive process if the data have not been smoothed or differenced.

The %AR Macro

The SAS macro %AR generates programming statements for PROC MODEL for autoregressive models. The %AR macro is part of SAS/ETS software, and no special options need to be set to use the macro. The autoregressive process can be applied to the structural equation errors or to the endogenous series themselves.

The %AR macro can be used for the following types of autoregression:

- univariate autoregression
- unrestricted vector autoregression
- restricted vector autoregression

Univariate Autoregression

To model the error term of an equation as an autoregressive process, use the following statement after the equation:

```
%ar( varname, nlags )
```

For example, suppose that Y is a linear function of X1, X2, and an AR(2) error. You would write this model as follows:

```
proc model data=in;
   parms a b c;
   y = a + b * x1 + c * x2;
   %ar( y, 2 )
   fit y / list;
run;
```

The calls to %AR must come *after* all of the equations that the process applies to.

The preceding macro invocation, %AR(y,2), produces the statements shown in the LIST output in Figure 19.60.

Figure 19.60 LIST Option Output for an AR(2) Model

The MODEL Procedure

Listing of Compiled Program Code

Stmt	Line:Col	Statement as Parsed
1	2498:4	PRED.y = a + b * x1 + c * x2;
1	2498:4	RESID.y = PRED.y - ACTUAL.y;
1	2498:4	ERROR.y = PRED.y - y;
2	2499:14	_PRED__y = PRED.y;
3	2499:15	_OLD_PRED.y = PRED.y + y_l1 * ZLAG1(y - _PRED__y) + y_l2 * ZLAG2(y - _PRED__y);
3	2499:15	PRED.y = _OLD_PRED.y;
3	2499:15	RESID.y = PRED.y - ACTUAL.y;
3	2499:15	ERROR.y = PRED.y - y;

The _PRED__ prefixed variables are temporary program variables used so that the lags of the residuals are the correct residuals and not the ones redefined by this equation. Note that this is equivalent to the statements explicitly written in the section "General Form for ARMA Models" on page 1207.

You can also restrict the autoregressive parameters to zero at selected lags. For example, if you wanted autoregressive parameters at lags 1, 12, and 13, you can use the following statements:

```
proc model data=in;
   parms a b c;
   y = a + b * x1 + c * x2;
   %ar( y, 13, , 1 12 13 )
   fit y / list;
run;
```

These statements generate the output shown in Figure 19.61.

Figure 19.61 LIST Option Output for an AR Model with Lags at 1, 12, and 13

The MODEL Procedure

Listing of Compiled Program Code

Stmt	Line:Col	Statement as Parsed
1	2507:4	PRED.y = a + b * x1 + c * x2;
1	2507:4	RESID.y = PRED.y - ACTUAL.y;
1	2507:4	ERROR.y = PRED.y - y;
2	2508:14	_PRED__y = PRED.y;
3	2508:15	_OLD_PRED.y = PRED.y + y_l1 * ZLAG1(y - _PRED__y) + y_l12 * ZLAG12(y - _PRED__y) + y_l13 * ZLAG13(y - _PRED__y);
3	2508:15	PRED.y = _OLD_PRED.y;
3	2508:15	RESID.y = PRED.y - ACTUAL.y;
3	2508:15	ERROR.y = PRED.y - y;

There are variations on the conditional least squares method, depending on whether observations at the start of the series are used to "warm up" the AR process. By default, the %AR conditional least squares method uses all the observations and assumes zeros for the initial lags of autoregressive terms. By using the M= option, you can request that %AR use the unconditional least squares (ULS) or maximum-likelihood (ML) method instead. For example,

```
proc model data=in;
   y = a + b * x1 + c * x2;
   %ar( y, 2, m=uls )
   fit y;
run;
```

Discussions of these methods is provided in the section "AR Initial Conditions" on page 1208.

By using the M=CLS*n* option, you can request that the first *n* observations be used to compute estimates of the initial autoregressive lags. In this case, the analysis starts with observation *n* +1. For example:

```
proc model data=in;
   y = a + b * x1 + c * x2;
   %ar( y, 2, m=cls2 )
   fit y;
run;
```

You can use the %AR macro to apply an autoregressive model to the endogenous variable, instead of to the error term, by using the TYPE=V option. For example, if you want to add the five past lags of Y to the equation in the previous example, you could use %AR to generate the parameters and lags by using the following statements:

```
proc model data=in;
   parms a b c;
   y = a + b * x1 + c * x2;
   %ar( y, 5, type=v )
   fit y / list;
run;
```

The preceding statements generate the output shown in Figure 19.62.

Figure 19.62 LIST Option Output for an AR model of Y

The MODEL Procedure

Listing of Compiled Program Code

Stmt	Line:Col	Statement as Parsed
1	2530:4	PRED.y = a + b * x1 + c * x2;
1	2530:4	RESID.y = PRED.y - ACTUAL.y;
1	2530:4	ERROR.y = PRED.y - y;
2	2531:15	_OLD_PRED.y = PRED.y + y_l1 * ZLAG1(y) + y_l2 * ZLAG2(y) + y_l3 * ZLAG3(y) + y_l4 * ZLAG4(y) + y_l5 * ZLAG5(y);
2	2531:15	PRED.y = _OLD_PRED.y;
2	2531:15	RESID.y = PRED.y - ACTUAL.y;
2	2531:15	ERROR.y = PRED.y - y;

This model predicts Y as a linear combination of X1, X2, an intercept, and the values of Y in the most recent five periods.

Unrestricted Vector Autoregression

To model the error terms of a set of equations as a vector autoregressive process, use the following form of the %AR macro after the equations:

```
%ar( process_name, nlags, variable_list )
```

The *process_name* value is any name that you supply for %AR to use in making names for the autoregressive parameters. You can use the %AR macro to model several different AR processes for different sets of equations by using different process names for each set. The process name ensures that the variable names used are unique. Use a short *process_name* value for the process if parameter estimates are to be written to an output data set. The %AR macro tries to construct parameter names less than or equal to eight characters, but this is limited by the length of *process_name*, which is used as a prefix for the AR parameter names.

The *variable_list* value is the list of endogenous variables for the equations.

For example, suppose that errors for equations Y1, Y2, and Y3 are generated by a second-order vector autoregressive process. You can use the following statements:

```
proc model data=in;
   y1 = ... equation for y1 ...;
   y2 = ... equation for y2 ...;
   y3 = ... equation for y3 ...;
   %ar( name, 2, y1 y2 y3 )
   fit y1 y2 y3;
run;
```

which generate the following for Y1 and similar code for Y2 and Y3:

```
y1 = pred.y1 + name1_1_1*zlag1(y1-name_y1) +
     name1_1_2*zlag1(y2-name_y2) +
     name1_1_3*zlag1(y3-name_y3) +
     name2_1_1*zlag2(y1-name_y1) +
     name2_1_2*zlag2(y2-name_y2) +
     name2_1_3*zlag2(y3-name_y3) ;
```

Only the conditional least squares (M=CLS or M=CLS*n*) method can be used for vector processes.

You can also use the same form with restrictions that the coefficient matrix be 0 at selected lags. For example, the following statements apply a third-order vector process to the equation errors with all the coefficients at lag 2 restricted to 0 and with the coefficients at lags 1 and 3 unrestricted:

```
proc model data=in;
   y1 = ... equation for y1 ...;
   y2 = ... equation for y2 ...;
   y3 = ... equation for y3 ...;
   %ar( name, 3, y1 y2 y3, 1 3 )
   fit y1 y2 y3;
```

You can model the three series Y1–Y3 as a vector autoregressive process in the variables instead of in the errors by using the TYPE=V option. If you want to model Y1–Y3 as a function of past values of Y1–Y3 and some exogenous variables or constants, you can use %AR to generate the statements for the lag terms. Write an equation for each variable for the nonautoregressive part of the model, and then call %AR with the TYPE=V option. For example,

```
proc model data=in;
   parms a1-a3 b1-b3;
   y1 = a1 + b1 * x;
   y2 = a2 + b2 * x;
   y3 = a3 + b3 * x;
   %ar( name, 2, y1 y2 y3, type=v )
   fit y1 y2 y3;
run;
```

The nonautoregressive part of the model can be a function of exogenous variables, or it can be intercept parameters. If there are no exogenous components to the vector autoregression model, including no intercepts, then assign zero to each of the variables. There must be an assignment to each of the variables before %AR is called.

```
proc model data=in;
   y1=0;
   y2=0;
   y3=0;
   %ar( name, 2, y1 y2 y3, type=v )
   fit y1 y2 y3;
run;
```

This example models the vector Y=(Y1 Y2 Y3)′ as a linear function only of its value in the previous two periods and a white noise error vector. The model has 18=(3 × 3 + 3 × 3) parameters.

Syntax of the %AR Macro

There are two cases of the syntax of the %AR macro. When restrictions on a vector AR process are not needed, the syntax of the %AR macro has the general form

%AR *(* name *,* nlag *<,*endolist *<,* laglist *>> <,M=* method *> <,TYPE= V>)* **;**

where

name specifies a prefix for %AR to use in constructing names of variables needed to define the AR process. If the *endolist* is not specified, the endogenous list defaults to *name*, which must be the name of the equation to which the AR error process is to be applied. The *name* value cannot exceed 32 characters.

nlag is the order of the AR process.

endolist specifies the list of equations to which the AR process is to be applied. If more than one name is given, an unrestricted vector process is created with the structural residuals of all the equations included as regressors in each of the equations. If not specified, *endolist* defaults to *name*.

laglist specifies the list of lags at which the AR terms are to be added. The coefficients of the terms at lags not listed are set to 0. All of the listed lags must be less than or equal to *nlag*,

and there must be no duplicates. If not specified, the *laglist* defaults to all lags 1 through *nlag*.

M=*method* specifies the estimation method to implement. Valid values of M= are CLS (conditional least squares estimates), ULS (unconditional least squares estimates), and ML (maximum likelihood estimates). M=CLS is the default. Only M=CLS is allowed when more than one equation is specified. The ULS and ML methods are not supported for vector AR models by %AR.

TYPE=V specifies that the AR process is to be applied to the endogenous variables themselves instead of to the structural residuals of the equations.

Restricted Vector Autoregression

You can control which parameters are included in the process, restricting to 0 those parameters that you do not include. First, use %AR with the DEFER option to declare the variable list and define the dimension of the process. Then, use additional %AR calls to generate terms for selected equations with selected variables at selected lags. For example,

```
proc model data=d;
   y1 = ... equation for y1 ...;
   y2 = ... equation for y2 ...;
   y3 = ... equation for y3 ...;
   %ar( name, 2, y1 y2 y3, defer )
   %ar( name, y1, y1 y2 )
   %ar( name, y2 y3, , 1 )
   fit y1 y2 y3;
run;
```

The error equations produced are as follows:

```
y1 = pred.y1 + name1_1_1*zlag1(y1-name_y1) +
     name1_1_2*zlag1(y2-name_y2) + name2_1_1*zlag2(y1-name_y1) +
     name2_1_2*zlag2(y2-name_y2) ;
y2 = pred.y2 + name1_2_1*zlag1(y1-name_y1) +
     name1_2_2*zlag1(y2-name_y2) + name1_2_3*zlag1(y3-name_y3) ;
y3 = pred.y3 + name1_3_1*zlag1(y1-name_y1) +
     name1_3_2*zlag1(y2-name_y2) + name1_3_3*zlag1(y3-name_y3) ;
```

This model states that the errors for Y1 depend on the errors of both Y1 and Y2 (but not Y3) at both lags 1 and 2, and that the errors for Y2 and Y3 depend on the previous errors for all three variables, but only at lag 1.

%AR Macro Syntax for Restricted Vector AR

An alternative use of %AR is allowed to impose restrictions on a vector AR process by calling %AR several times to specify different AR terms and lags for different equations.

The first call has the general form

%AR(name, nlag, endolist *, DEFER)* **;**

where

name specifies a prefix for %AR to use in constructing names of variables needed to define the vector AR process.

nlag specifies the order of the AR process.

endolist specifies the list of equations to which the AR process is to be applied.

DEFER specifies that %AR is not to generate the AR process but is to wait for further information specified in later %AR calls for the same *name* value.

The subsequent calls have the general form

```
%AR( name, eqlist, varlist, laglist,TYPE= )
```

where

name is the same as in the first call.

eqlist specifies the list of equations to which the specifications in this %AR call are to be applied. Only names specified in the *endolist* value of the first call for the *name* value can appear in the list of equations in *eqlist*.

varlist specifies the list of equations whose lagged structural residuals are to be included as regressors in the equations in *eqlist*. Only names in the *endolist* of the first call for the *name* value can appear in *varlist*. If not specified, *varlist* defaults to *endolist*.

laglist specifies the list of lags at which the AR terms are to be added. The coefficients of the terms at lags not listed are set to 0. All of the listed lags must be less than or equal to the value of *nlag*, and there must be no duplicates. If not specified, *laglist* defaults to all lags 1 through *nlag*.

The %MA Macro

The SAS macro %MA generates programming statements for PROC MODEL for moving-average models. The %MA macro is part of SAS/ETS software, and no special options are needed to use the macro. The moving-average error process can be applied to the structural equation errors. The syntax of the %MA macro is the same as the %AR macro except there is no TYPE= argument.

When you are using the %MA and %AR macros combined, the %MA macro must follow the %AR macro. The following SAS/IML statements produce an ARMA(1, (1 3)) error process and save it in the data set MADAT2.

```
proc iml;
   phi = { 1 .2 };
   theta = { 1 .3 0 .5 };
   y = armasim( phi, theta, 0, .1, 200, 32565 );
   create madat2 from y[colname='y'];
   append from y;
quit;
```

The following PROC MODEL statements are used to estimate the parameters of this model by using maximum likelihood error structure:

```
title 'Maximum Likelihood ARMA(1, (1 3))';
proc model data=madat2;
   y=0;
   %ar( y, 1, ,  M=ml )
   %ma( y, 3, , 1 3,  M=ml )  /* %MA always after %AR */
   fit y;
run;
title;
```

The estimates of the parameters produced by this run are shown in Figure 19.63.

Figure 19.63 Estimates from an ARMA(1, (1 3)) Process

Maximum Likelihood ARMA(1, (1 3))

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
y	3	197	2.6383	0.0134	0.1157	-0.0067	-0.0169
RESID.y		197	1.9957	0.0101	0.1007		

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
y_l1	-0.10067	0.1187	-0.85	0.3973	AR(y) y lag1 parameter
y_m1	-0.1934	0.0939	-2.06	0.0408	MA(y) y lag1 parameter
y_m3	-0.59384	0.0601	-9.88	<.0001	MA(y) y lag3 parameter

Syntax of the %MA Macro

There are two cases of the syntax for the %MA macro. When restrictions on a vector MA process are not needed, the syntax of the %MA macro has the general form

%MA *(* name *,* nlag < *,* endolist < *,* laglist >> < *,M=* method > *)* **;**

where

name specifies a prefix for %MA to use in constructing names of variables needed to define the MA process and is the default *endolist*.

nlag is the order of the MA process.

endolist specifies the equations to which the MA process is to be applied. If more than one name is given, CLS estimation is used for the vector process.

laglist specifies the lags at which the MA terms are to be added. All of the listed lags must be less than or equal to *nlag*, and there must be no duplicates. If not specified, the *laglist* defaults to all lags 1 through *nlag*.

M=*method* specifies the estimation method to implement. Valid values of M= are CLS (conditional least squares estimates), ULS (unconditional least squares estimates), and ML (maximum

likelihood estimates). M=CLS is the default. Only M=CLS is allowed when more than one equation is specified in the *endolist*.

%MA Macro Syntax for Restricted Vector Moving-Average

An alternative use of %MA is allowed to impose restrictions on a vector MA process by calling %MA several times to specify different MA terms and lags for different equations.

The first call has the general form

%MA(name , nlag , endolist , *DEFER)* **;**

where

name specifies a prefix for %MA to use in constructing names of variables needed to define the vector MA process.

nlag specifies the order of the MA process.

endolist specifies the list of equations to which the MA process is to be applied.

DEFER specifies that %MA is not to generate the MA process but is to wait for further information specified in later %MA calls for the same *name* value.

The subsequent calls have the general form

```
%MA( name, eqlist, varlist, laglist )
```

where

name is the same as in the first call.

eqlist specifies the list of equations to which the specifications in this %MA call are to be applied.

varlist specifies the list of equations whose lagged structural residuals are to be included as regressors in the equations in *eqlist*.

laglist specifies the list of lags at which the MA terms are to be added.

Distributed Lag Models and the %PDL Macro

In the following example, the variable y is modeled as a linear function of x, the first lag of x, the second lag of x, and so forth:

$$y_t = a + b_0 x_t + b_1 x_{t-1} + b_2 x_{t-2} + b_3 x_{t-3} + \ldots + b_n x_{t-n}$$

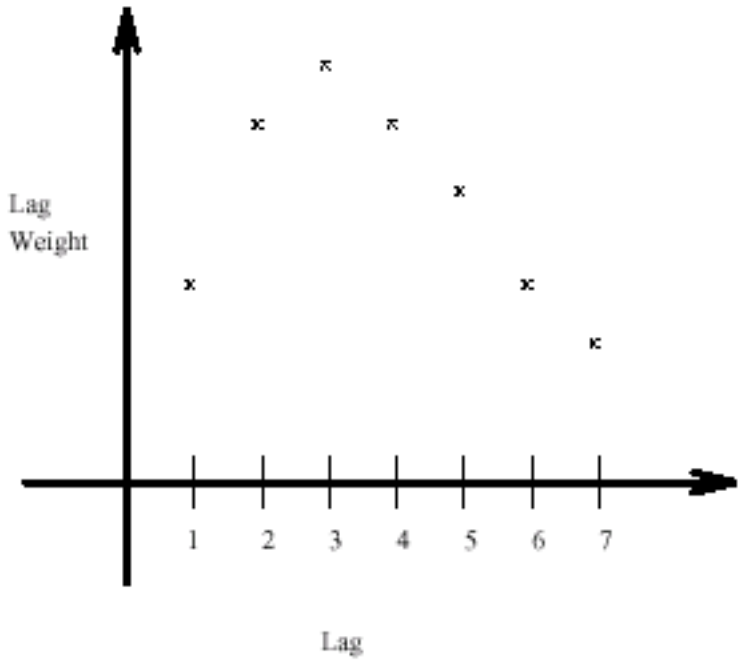
Models of this sort can introduce a great many parameters for the lags, and there may not be enough data to compute accurate independent estimates for them all. Often, the number of parameters is reduced by

assuming that the lag coefficients follow some pattern. One common assumption is that the lag coefficients follow a polynomial in the lag length

$$b_i = \sum_{j=0}^{d} \alpha_j (i)^j$$

where d is the degree of the polynomial used. Models of this kind are called *Almon lag models*, *polynomial distributed lag models*, or *PDLs* for short. For example, Figure 19.64 shows the lag distribution that can be modeled with a low-order polynomial. Endpoint restrictions can be imposed on a PDL to require that the lag coefficients be 0 at the 0th lag, or at the final lag, or at both.

Figure 19.64 Polynomial Distributed Lags



For linear single-equation models, SAS/ETS software includes the PDLREG procedure for estimating PDL models. See Chapter 21, "The PDLREG Procedure," for a more detailed discussion of polynomial distributed lags and an explanation of endpoint restrictions.

Polynomial and other distributed lag models can be estimated and simulated or forecast with PROC MODEL. For polynomial distributed lags, the %PDL macro can generate the needed programming statements automatically.

The %PDL Macro

The SAS macro %PDL generates the programming statements to compute the lag coefficients of polynomial distributed lag models and to apply them to the lags of variables or expressions.

To use the %PDL macro in a model program, you first call it to declare the lag distribution; later, you call it again to apply the PDL to a variable or expression. The first call generates a PARMS statement for the

polynomial parameters and assignment statements to compute the lag coefficients. The second call generates an expression that applies the lag coefficients to the lags of the specified variable or expression. A PDL can be declared only once, but it can be used any number of times (that is, the second call can be repeated).

The initial declaratory call has the general form

%PDL *(* pdlname, nlags, degree *, R=*code *, OUTEST=*dataset *)* **;**

where *pdlname* is a name (up to 32 characters) that you give to identify the PDL, *nlags* is the lag length, and *degree* is the degree of the polynomial for the distribution. The R=*code* is optional for endpoint restrictions. The value of *code* can be FIRST (for upper), LAST (for lower), or BOTH (for both upper and lower endpoints). See Chapter 21, "The PDLREG Procedure," for a discussion of endpoint restrictions. The option OUTEST=*dataset* creates a data set that contains the estimates of the parameters and their covariance matrix.

The later calls to apply the PDL have the general form

```
%PDL( pdlname, expression )
```

where *pdlname* is the name of the PDL and *expression* is the variable or expression to which the PDL is to be applied. The *pdlname* given must be the same as the name used to declare the PDL.

The following statements produce the output in Figure 19.65:

```
proc model data=in list;
   parms int pz;
   %pdl(xpdl,5,2);
   y = int + pz * z + %pdl(xpdl,x);
   %ar(y,2,M=ULS);
   id i;
fit y / out=model1 outresid converge=1e-6;
run;
```

Figure 19.65 %PDL Macro Estimates

The MODEL Procedure

Nonlinear OLS Estimates

Term	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
XPDL_L0	1.568788	0.0935	16.77	<.0001	PDL(XPDL,5,2) coefficient for lag0
XPDL_L1	0.564917	0.0328	17.22	<.0001	PDL(XPDL,5,2) coefficient for lag1
XPDL_L2	-0.05063	0.0593	-0.85	0.4155	PDL(XPDL,5,2) coefficient for lag2
XPDL_L3	-0.27785	0.0517	-5.37	0.0004	PDL(XPDL,5,2) coefficient for lag3
XPDL_L4	-0.11675	0.0368	-3.17	0.0113	PDL(XPDL,5,2) coefficient for lag4
XPDL_L5	0.43267	0.1362	3.18	0.0113	PDL(XPDL,5,2) coefficient for lag5

This second example models two variables, Y1 and Y2, and uses two PDLs:

```
proc model data=in;
   parms int1 int2;
   %pdl( logxpdl, 5, 3 )
   %pdl( zpdl, 6, 4 )
   y1 = int1 + %pdl( logxpdl, log(x) ) + %pdl( zpdl, z );
```

```
    y2 = int2 + %pdl( zpdl, z );
    fit y1 y2;
run;
```

A (5,3) PDL of the log of X is used in the equation for Y1. A (6,4) PDL of Z is used in the equations for both Y1 and Y2. Since the same ZPDL is used in both equations, the lag coefficients for Z are the same for the Y1 and Y2 equations, and the polynomial parameters for ZPDL are shared by the two equations. See Example 19.5 for a complete example and comparison with PDLREG.

Input Data Sets

DATA= Input Data Set

For FIT tasks, the DATA= option specifies which input data set to use in estimating parameters. Variables in the model program are looked up in the DATA= data set and, if found, their attributes (type, length, label, and format) are set to be the same as those in the DATA= data set (if not defined otherwise within PROC MODEL).

ESTDATA= Input Data Set

The ESTDATA= option specifies an input data set that contains an observation that gives values for some or all of the model parameters. The data set can also contain observations that gives the rows of a covariance matrix for the parameters.

Parameter values read from the ESTDATA= data set provide initial starting values for parameters estimated. Observations that provide covariance values, if any are present in the ESTDATA= data set, are ignored.

The ESTDATA= data set is usually created by the OUTEST= option in a previous FIT statement. You can also create an ESTDATA= data set with a SAS DATA step program. The data set must contain a numeric variable for each parameter to be given a value or covariance column. The name of the variable in the ESTDATA= data set must match the name of the parameter in the model. Parameters with names longer than 32 characters cannot be set from an ESTDATA= data set. The data set must also contain a character variable _NAME_ of length 32. _NAME_ has a blank value for the observation that gives values to the parameters. _NAME_ contains the name of a parameter for observations that define rows of the covariance matrix.

More than one set of parameter estimates and covariances can be stored in the ESTDATA= data set if the observations for the different estimates are identified by the variable _TYPE_. _TYPE_ must be a character variable of length 8. The TYPE= option is used to select for input the part of the ESTDATA= data set for which the _TYPE_ value matches the value of the TYPE= option.

In PROC MODEL, you have several options to specify starting values for the parameters to be estimated. When more than one option is specified, the options are implemented in the following order of precedence (from highest to lowest): the START= option, the PARMS statement initialization value, the ESTDATA= option, and the PARMSDATA= option. If no options are specified for the starting value, the default value of 0.0001 is used.

The following SAS statements generate the ESTDATA= data set shown in Figure 19.66. The second FIT statement uses the TYPE= option to select the estimates from the GMM estimation as starting values for the FIML estimation.

```
/* Generate test data */
data gmm2;
    do t=1 to 50;
       x1 = sqrt(t) ;
       x2 = rannor(10) * 10;
       y1 = -.002 * x2 * x2 - .05 / x2 - 0.001 * x1 * x1;
       y2 = 0.002* y1 + 2 * x2 * x2 + 50 / x2 + 5 * rannor(1);
       y1 = y1 + 5 * rannor(1);
       z1 = 1; z2 = x1 * x1; z3 = x2 * x2; z4 = 1.0/x2;
       output;
    end;
run;

proc model data=gmm2 ;
   exogenous x1 x2;
   parms a1 a2 b1 2.5 b2 c2 55 d1;
   inst b1 b2 c2 x1 x2;
   y1 = a1 * y2 + b1 * x1 * x1 + d1;
   y2 = a2 * y1 + b2 * x2 * x2 + c2 / x2 + d1;

   fit y1 y2 / 3sls gmm kernel=(qs,1,0.2) outest=gmmest;

   fit y1 y2 / fiml type=gmm estdata=gmmest;
run;

proc print data=gmmest;
run;
```

Figure 19.66 ESTDATA= Data Set

Obs	_NAME_	_TYPE_	_STATUS_	_NUSED_	a1	a2	b1	b2	c2	d1
1		3SLS	0 Converged	50	-.002229607	-1.25002	0.025827	1.99609	49.8119	-0.44533
2		GMM	0 Converged	50	-.001772196	-1.02345	0.014025	1.99726	49.8648	-0.87573

MISSING=PAIRWISE | DELETE

When missing values are encountered for any one of the equations in a system of equations, the default action is to drop that observation for all of the equations. The new MISSING=PAIRWISE option in the FIT statement provides a different method of handling missing values that avoids losing data for nonmissing equations for the observation. This is especially useful for SUR estimation on equations with unequal numbers of observations.

The option MISSING=PAIRWISE specifies that missing values are tracked on an equation-by-equation basis. The MISSING=DELETE option specifies that the entire observation is omitted from the analysis when any equation has a missing predicted or actual value for the equation. The default is MISSING=DELETE.

When you specify the MISSING=PAIRWISE option, the $\mathbf{S}$ matrix is computed as

$$\mathbf{S} = \mathbf{D}(\mathbf{R}'\mathbf{R})\mathbf{D}$$

where $\mathbf{D}$ is a diagonal matrix that depends on the VARDEF= option, the matrix $\mathbf{R}$ is $(\mathbf{r}_1, \ldots, \mathbf{r}_g)$, and $\mathbf{r}_i$ is the vector of residuals for the ith equation with r_{ij} replaced with zero when r_{ij} is missing.

For MISSING=PAIRWISE, the calculation of the diagonal element $d_{i,i}$ of **D** is based on n_i, the number of nonmissing observations for the *i*th equation, instead of on n. Similarly, for VARDEF=WGT or WDF, the calculation is based on the sum of the weights for the nonmissing observations for the *i*th equation instead of on the sum of the weights for all observations. See the description of the VARDEF= option for the definition of **D**.

The degrees-of-freedom correction for a shared parameter is computed by using the average number of observations used in its estimation.

The MISSING=PAIRWISE option is not valid for the GMM and FIML estimation methods.

For the instrumental variables estimation methods (2SLS, 3SLS), when an instrument is missing for an observation, that observation is dropped for all equations, regardless of the MISSING= option.

PARMSDATA= Input Data Set

The option PARMSDATA= reads values for all parameters whose names match the names of variables in the PARMSDATA= data set. Values for any or all of the parameters in the model can be reset by using the PARMSDATA= option. The PARMSDATA= option goes in the PROC MODEL statement, and the data set is read before any FIT or SOLVE statements are executed.

In PROC MODEL, you have several options to specify starting values for the parameters to be estimated. When more than one option is specified, the options are implemented in the following order of precedence (from highest to lowest): the START= option, the PARMS statement initialization value, the ESTDATA= option, and the PARMSDATA= option. If no options are specified for the starting value, the default value of 0.0001 is used. Together, the OUTPARMS= and PARMSDATA= options enable you to change part of a model and recompile the new model program without the need to reestimate equations that were not changed.

Suppose you have a large model with parameters estimated and you now want to replace one equation, Y, with a new specification. Although the model program must be recompiled with the new equation, you don't need to reestimate all the equations, just the one that changed.

Using the OUTPARMS= and PARMSDATA= options, you could do the following:

```
proc model model=oldmod outparms=temp; run;
proc model outmodel=newmod parmsdata=temp data=in;
   ...  include new model definition with changed y eq. here ...
   fit y;
run;
```

The model file NEWMOD then contains the new model and its estimated parameters plus the old models with their original parameter values.

SDATA= Input Data Set

The SDATA= option allows a cross-equation covariance matrix to be input from a data set. The **S** matrix read from the SDATA= data set, specified in the FIT statement, is used to define the objective function for the OLS, N2SLS, SUR, and N3SLS estimation methods and is used as the initial **S** for the methods that iterate the **S** matrix.

Most often, the SDATA= data set has been created by the OUTS= or OUTSUSED= option in a previous FIT statement. The OUTS= and OUTSUSED= data sets from a FIT statement can be read back in by a FIT statement in the same PROC MODEL step.

You can create an input SDATA= data set by using the DATA step. PROC MODEL expects to find a character variable _NAME_ in the SDATA= data set as well as variables for the equations in the estimation or solution. For each observation with a _NAME_ value that matches the name of an equation, PROC MODEL fills the corresponding row of the **S** matrix with the values of the names of equations found in the data set. If a row or column is omitted from the data set, a 1 is placed on the diagonal for the row or column. Missing values are ignored, and since the **S** matrix is symmetric, you can include only a triangular part of the **S** matrix in the SDATA= data set with the omitted part indicated by missing values. If the SDATA= data set contains multiple observations with the same _NAME_, the last values supplied for the _NAME_ are used. The structure of the expected data set is further described in the section "OUTS= Data Set" on page 1228.

Use the TYPE= option in the PROC MODEL or FIT statement to specify the type of estimation method used to produce the **S** matrix you want to input.

The following SAS statements are used to generate an **S** matrix from a GMM and a 3SLS estimation and to store that estimate in the data set GMMS:

```
proc model data=gmm2 ;
   exogenous x1 x2;
   parms a1 a2 b1 2.5 b2 c2 55 d1;
   inst b1 b2 c2 x1 x2;
   y1 = a1 * y2 + b1 * x1 * x1 + d1;
   y2 = a2 * y1 + b2 * x2 * x2 + c2 / x2 + d1;

   fit y1 y2 / 3sls gmm kernel=(qs,1,0.2)
               outest=gmmest outs=gmms;
run;

proc print data=gmms;
run;
```

The data set GMMS is shown in Figure 19.67.

Figure 19.67 SDATA= Data Set

Obs	_NAME_	_TYPE_	_NUSED_	y1	y2
1	y1	3SLS	50	27.1032	38.1599
2	y2	3SLS	50	38.1599	74.6253
3	y1	GMM	50	27.6248	32.2811
4	y2	GMM	50	32.2811	58.8387

VDATA= Input data set

The VDATA= option enables a variance matrix for GMM estimation to be input from a data set. When the VDATA= option is used in the PROC MODEL or FIT statement, the matrix that is input is used to define the objective function and is used as the initial **V** for the methods that iterate the **V** matrix.

Normally the VDATA= matrix is created from the OUTV= option in a previous FIT statement. Alternately an input VDATA= data set can be created by using the DATA step. Each row and column of the **V** matrix is associated with an equation and an instrument. The position of each element in the **V** matrix can then be indicated by an equation name and an instrument name for the row of the element and an equation name and

an instrument name for the column. Each observation in the VDATA= data set is an element in the **V** matrix. The row and column of the element are indicated by four variables (EQ_ROW, INST_ROW, EQ_COL, and INST_COL) that contain the equation name or instrument name. The variable name for an element is VALUE. Missing values are set to 0. Because the variance matrix is symmetric, only a triangular part of the matrix needs to be input.

The following SAS statements are used to generate a **V** matrix estimation from GMM and to store that estimate in the data set GMMV:

```
proc model data=gmm2;
   exogenous x1 x2;
   parms a1 a2 b2 b1 2.5 c2 55 d1;
   inst b1 b2 c2 x1 x2;
   y1 = a1 * y2 + b1 * x1 * x1 + d1;
   y2 = a2 * y1 + b2 * x2 * x2 + c2 / x2 + d1;

   fit y1 y2 / gmm outv=gmmv;
run;

proc print data=gmmv(obs=15);
run;
```

The data set GMM2 was generated by the example in the preceding ESTDATA= section. The **V** matrix stored in GMMV is selected for use in an additional GMM estimation by the following FIT statement:

```
fit y1 y2 / gmm vdata=gmmv;
run;
```

A partial listing of the GMMV data set is shown in Figure 19.68. There are a total of 78 observations in this data set. The **V** matrix is 12 by 12 for this example.

Figure 19.68 The First 15 Observations in the VDATA= Data Set

Obs	_TYPE_	EQ_ROW	EQ_COL	INST_ROW	INST_COL	VALUE
1	GMM	y1	y1	1	1	1555.78
2	GMM	y1	y1	x1	1	8565.80
3	GMM	y1	y1	x1	x1	49932.47
4	GMM	y1	y1	x2	1	8244.34
5	GMM	y1	y1	x2	x1	51324.21
6	GMM	y1	y1	x2	x2	159913.24
7	GMM	y1	y1	@PRED.y1/@b1	1	49933.61
8	GMM	y1	y1	@PRED.y1/@b1	x1	301270.02
9	GMM	y1	y1	@PRED.y1/@b1	x2	317277.10
10	GMM	y1	y1	@PRED.y1/@b1	@PRED.y1/@b1	1860095.90
11	GMM	y1	y1	@PRED.y2/@b2	1	163855.31
12	GMM	y1	y1	@PRED.y2/@b2	x1	900622.60
13	GMM	y1	y1	@PRED.y2/@b2	x2	1285421.56
14	GMM	y1	y1	@PRED.y2/@b2	@PRED.y1/@b1	5173744.58
15	GMM	y1	y1	@PRED.y2/@b2	@PRED.y2/@b2	30307640.16

Output Data Sets

OUT= Data Set

For normalized form equations, the OUT= data set specified in the FIT statement contains residuals, actuals, and predicted values of the dependent variables computed from the parameter estimates. For general form equations, actual values of the endogenous variables are copied for the residual and predicted values.

The variables in the data set are as follows:

- BY variables
- RANGE variable
- ID variables
- _ESTYPE_, a character variable of length 8 that identifies the estimation method: OLS, SUR, N2SLS, N3SLS, ITOLS, ITSUR, IT2SLS, IT3SLS, GMM, ITGMM, or FIML
- _TYPE_, a character variable of length 8 that identifies the type of observation: RESIDUAL, PREDICT, or ACTUAL
- _WEIGHT_, the weight of the observation in the estimation. The _WEIGHT_ value is 0 if the observation was not used. It is equal to the product of the _WEIGHT_ model program variable and the variable named in the WEIGHT statement, if any, or 1 if weights were not used.
- the WEIGHT statement variable if used
- the model variables. The dependent variables for the normalized form equations in the estimation contain residuals, actuals, or predicted values, depending on the _TYPE_ variable, whereas the model variables that are not associated with estimated equations always contain actual values from the input data set.
- any other variables named in the OUTVARS statement. These can be program variables computed by the model program, CONTROL variables, parameters, or special variables in the model program.

The following SAS statements are used to generate and print an OUT= data set:

```
proc model data=gmm2;
   exogenous x1 x2;
   parms a1 a2 b2 b1 2.5 c2 55 d1;
   inst b1 b2 c2 x1 x2;
   y1 = a1 * y2 + b1 * x1 * x1 + d1;
   y2 = a2 * y1 + b2 * x2 * x2 + c2 / x2 + d1;

   fit y1 y2 / 3sls gmm out=resid outall ;
run;

proc print data=resid(obs=20);
run;
```

The data set GMM2 was generated by the example in the preceding ESTDATA= section above. A partial listing of the RESID data set is shown in Figure 19.69.

Figure 19.69 The OUT= Data Set

Obs	_ESTYPE_	_TYPE_	_WEIGHT_	x1	x2	y1	y2
1	3SLS	ACTUAL	1	1.00000	-1.7339	-3.05812	-23.071
2	3SLS	PREDICT	1	1.00000	-1.7339	-0.36806	-19.351
3	3SLS	RESIDUAL	1	1.00000	-1.7339	-2.69006	-3.720
4	3SLS	ACTUAL	1	1.41421	-5.3046	0.59405	43.866
5	3SLS	PREDICT	1	1.41421	-5.3046	-0.49148	45.588
6	3SLS	RESIDUAL	1	1.41421	-5.3046	1.08553	-1.722
7	3SLS	ACTUAL	1	1.73205	-5.2826	3.17651	51.563
8	3SLS	PREDICT	1	1.73205	-5.2826	-0.48281	41.857
9	3SLS	RESIDUAL	1	1.73205	-5.2826	3.65933	9.707
10	3SLS	ACTUAL	1	2.00000	-0.6878	3.66208	-70.011
11	3SLS	PREDICT	1	2.00000	-0.6878	-0.18592	-76.502
12	3SLS	RESIDUAL	1	2.00000	-0.6878	3.84800	6.491
13	3SLS	ACTUAL	1	2.23607	-7.0797	0.29210	99.177
14	3SLS	PREDICT	1	2.23607	-7.0797	-0.53732	92.201
15	3SLS	RESIDUAL	1	2.23607	-7.0797	0.82942	6.976
16	3SLS	ACTUAL	1	2.44949	14.5284	1.86898	423.634
17	3SLS	PREDICT	1	2.44949	14.5284	-1.23490	421.969
18	3SLS	RESIDUAL	1	2.44949	14.5284	3.10388	1.665
19	3SLS	ACTUAL	1	2.64575	-0.6968	-1.03003	-72.214
20	3SLS	PREDICT	1	2.64575	-0.6968	-0.10353	-69.680

OUTEST= Data Set

The OUTEST= data set contains parameter estimates and, if requested, estimates of the covariance of the parameter estimates.

The variables in the data set are as follows:

- BY variables
- _NAME_, a character variable of length 32, blank for observations that contain parameter estimates or a parameter name for observations that contain covariances
- _TYPE_, a character variable of length 8 that identifies the estimation method: OLS, SUR, N2SLS, N3SLS, ITOLS, ITSUR, IT2SLS, IT3SLS, GMM, ITGMM, or FIML
- _STATUS_, variable that gives the convergence status of estimation. _STATUS_ = 0 when convergence criteria are met, = 1 when estimation converges with a note, = 2 when estimation converges with a warning, and = 3 when estimation fails to converge
- _NUSED_, the number of observations used in estimation
- the parameters estimated

If the COVOUT option is specified, an additional observation is written for each row of the estimate of the covariance matrix of parameter estimates, with the _NAME_ values that contain the parameter names for the rows. Parameter names longer than 32 characters are truncated.

OUTPARMS= Data Set

The option OUTPARMS= writes all the parameter estimates to an output data set. This output data set contains one observation and is similar to the OUTEST= data set, but it contains all the parameters, is not associated with any FIT task, and contains no covariances. The OUTPARMS= option is used in the PROC MODEL statement, and the data set is written at the end, after any FIT or SOLVE steps have been performed.

OUTS= Data Set

The OUTS= SAS data set contains the estimate of the covariance matrix of the residuals across equations. This matrix is formed from the residuals that are computed by using the parameter estimates.

The variables in the OUTS= data set are as follows:

- BY variables
- _NAME_, a character variable that contains the name of the equation
- _TYPE_, a character variable of length 8 that identifies the estimation method: OLS, SUR, N2SLS, N3SLS, ITOLS, ITSUR, IT2SLS, IT3SLS, GMM, ITGMM, or FIML
- variables with the names of the equations in the estimation

Each observation contains a row of the covariance matrix. The data set is suitable for use with the SDATA= option in a subsequent FIT or SOLVE statement. (See the section "Tests on Parameters" on page 1196 in this chapter for an example of the SDATA= option.)

OUTSUSED= Data Set

The OUTSUSED= SAS data set contains the covariance matrix of the residuals across equations that is used to define the objective function. The form of the OUTSUSED= data set is the same as that for the OUTS= data set.

Note that OUTSUSED= is the same as OUTS= for the estimation methods that iterate the **S** matrix (ITOLS, IT2SLS, ITSUR, and IT3SLS). If the SDATA= option is specified in the FIT statement, OUTSUSED= is the same as the SDATA= matrix read in for the methods that do not iterate the **S** matrix (OLS, SUR, N2SLS, and N3SLS).

OUTV= Data Set

The OUTV= data set contains the estimate of the variance matrix, **V**. This matrix is formed from the instruments and the residuals that are computed by using the final parameter estimates obtained from the estimation method chosen.

An estimate of **V** obtained from 2SLS is used in GMM estimation. Hence if you input the dataset obtained from the OUTV statement in 2SLS into the VDATA statement while fitting GMM, you get the same result by fitting GMM directly without specifying the VDATA option.

ODS Table Names

PROC MODEL assigns a name to each table it creates. You can use these names to reference the table when you use the Output Delivery System (ODS) to select tables and create output data sets. These names are listed in the following table.

Table 19.4 ODS Tables Produced in PROC MODEL

ODS Table Name	Description	Option
ODS Tables Created by the FIT Statement		
AugGMMCovariance	Crossproducts matrix	GMM ITALL
ChowTest	Structural change test	CHOW=
CollinDiagnostics	Collinearity diagnostics	
ConfInterval	Profile likelihood confidence intervals	PRL=
ConvCrit	Convergence criteria for estimation	default
ConvergenceStatus	Convergence status	default
CorrB	Correlations of parameters	COVB/CORRB
CorrResiduals	Correlations of residuals	CORRS/COVS
CovB	Covariance of parameters	COVB/CORRB
CovResiduals	Covariance of residuals	CORRS/COVS
Crossproducts	Crossproducts matrix	ITALL/ITPRINT
DatasetOptions	Data sets used	default
DetResidCov	Determinant of the residuals	DETAILS
DWTest	Durbin Watson test	DW=
Equations	Listing of equations to estimate	default
EstSummaryMiss	Model summary statistics for PAIRWISE	MISSING=
EstSummaryStats	Objective, objective * N	default
FirstLagrMultEst	First order Lagrange Multiplier estimates	GMM ITALL
GMMCovariance	Crossproducts matrix	GMM DETAILS
GMMTestStats	GMM test statistics	GMM
Godfrey	Godfrey's serial correlation test	GF=
HausmanTest	Hausman's test table	HAUSMAN
HeteroTest	Heteroscedasticity test tables	BREUSCH/PAGEN
InvXPXMat	X′X inverse for system	I
IterInfo	Iteration printing	ITALL/ITPRINT
LagLength	Model lag length	default
MinSummary	Number of parameters, estimation kind	default
ModSummary	Listing of all categorized variables	default
ModVars	Listing of model variables and parameters	default
NormalityTest	Normality test table	NORMAL
ObsSummary	Identifies observations with errors	default
ObsUsed	Observations read, used, and missing.	default
ParameterEstimates	Parameter estimates	default
ParmChange	Parameter change vector	ITALL
ResidSummary	Summary of the SSE, MSE for the equations	default

Table 19.4 (continued)

ODS Table Name	Description	Option
SecondLagrMultEst	Second order Lagrange Multiplier estimates	GMM ITALL
SizeInfo	Storage requirement for estimation	DETAILS
TermEstimates	Nonlinear OLS and ITOLS Estimates	OLS/ITOLS
TestResults	Test statement table	
WgtVar	The name of the weight variable	
XPXMat	X′X for system	XPX
YkVector	Marquardt iteration vector	GMM ITALL
ODS Tables Created by the SOLVE Statement		
BlockEqsAndVars	Dependency analysis block partitioning	ANALYZEDEPS=
DatasetOptions	Data sets used	default
DescriptiveStatistics	Descriptive statistics	STATS
FitStatistics	Fit statistics for simulation	STATS
LagLength	Model lag length	default
ModSummary	Listing of all categorized variables	default
ObsSummary	Simulation trace output	SOLVEPRINT
ObsUsed	Observations read, used, and missing.	default
SimulationSummary	Number of variables solved for	default
SolutionVarList	Solution variable lists	default
TheilRelStats	Theil relative change error statistics	THEIL
TheilStats	Theil forecast error statistics	THEIL
ErrorVec	Iteration Error vector	ITPRINT
ResidualValues	Iteration residual values	ITPRINT
PredictedValues	Iteration predicted values	ITPRINT
SolutionValues	Iteration solved for variable values	ITPRINT
ODS Tables Created by the FIT and SOLVE Statements		
AdjacencyMatrix	Adjacency graph	GRAPH
BlockAnalysis	Block analysis	BLOCK
BlockStructure	Block structure	BLOCK
CodeDependency	Variable cross reference	LISTDEP
CodeList	Listing of programs statements	LISTCODE
CrossReference	Cross-reference listing for program	
DepStructure	Dependency structure of the system	BLOCK
FirstDerivatives	First derivative table	LISTDER
IterIntg	Integration iteration output	INTGPRINT
MemUsage	Memory usage statistics	MEMORYUSE
MissingDependencies	Missing values by dependency	REPORTMISSINGS
MissingObservations	Missing values by observation	REPORTMISSINGS
MissingSymbols	Missing values by symbol	REPORTMISSINGS
ParmReadIn	Parameter estimates read in	ESTDATA=
ProgList	Listing of compiled program code	
RangeInfo	RANGE statement specification	

Table 19.4 (continued)

ODS Table Name	Description	Option
SortAdjacencyMatrix	Sorted adjacency graph	GRAPH
TransitiveClosure	Transitive closure graph	GRAPH

The AugGMMCovariance table is the **V** matrix augmented with the moment vector at iteration zero, produced when the ITALL option is used with the GMM option. If the **V** matrix to be used in GMM is read in by the VDATA option, then AugGMMCovariance would be the same matrix augmented with the moment vectors. The GMMCovariance ODS output is produced only when you read in a covariance matrix to be used in the GMM method. This table is produced by using DETAILS option with the GMM option.

ODS Graphics

Statistical procedures use ODS Graphics to create graphs as part of their output. ODS Graphics is described in detail in Chapter 21, "Statistical Graphics Using ODS" (*SAS/STAT User's Guide*).

Before you create graphs, ODS Graphics must be enabled (for example, with 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 the use of ODS for creating graphics with the MODEL procedure.

ODS Graph Names

PROC MODEL assigns a name to each graph it creates using ODS. You can use these names to reference the graphs when you use ODS. The names are listed in Table 19.5.

To request these graphs, ODS Graphics must be enabled.

Table 19.5 ODS Graphics Produced by PROC MODEL

ODS Graph Name	Plot Description
ACFPlot	Autocorrelation of residuals
ActualByPredicted	Predicted versus actual plot
BlockDependencyPlot	Simulation dependency analysis
CooksD	Cook's *D* plot
DiagnosticsPanel	Panel of all plots
IACFPlot	Inverse autocorrelation of residuals
QQPlot	Q-Q plot of residuals
PACFPlot	Partial autocorrelation of residuals
ResidualHistogram	Histogram of the residuals
StudentResidualPlot	Studentized residual plot

Details: Simulation by the MODEL Procedure

The *solution*, given the vector **k**, of the following nonlinear system of equations is the vector **u** that satisfies this equation:

$$\mathbf{q}(\mathbf{u}, \mathbf{k}, \boldsymbol{\theta}) = 0$$

A *simulation* is a set of solutions $\mathbf{u}_t$ for a specific sequence of vectors $\mathbf{k}_t$.

Model simulation can be performed to do the following:

- check how well the model predicts the actual values over the historical period
- investigate the sensitivity of the solution to changes in the input values or parameters
- examine the dynamic characteristics of the model
- check the stability of the simultaneous solution
- estimate the statistical distribution of the predicted values of the nonlinear model using Monte Carlo methods

By combining the various solution modes with different input data sets, model simulation can answer many different questions about the model. This section presents details of model simulation and solution.

Solution Modes

The following solution modes are commonly used:

- The *dynamic simultaneous forecast* mode is used for forecasting with the model. Collect the historical data on the model variables, the future assumptions of the exogenous variables, and any prior information on the future endogenous values, and combine them in a SAS data set. Use the FORECAST option in the SOLVE statement.
- The *dynamic simultaneous simulation* mode is often called *ex post simulation*, *historical simulation*, or *ex post forecasting*. Use the DYNAMIC option. This mode is the default.
- The *static simultaneous simulation* mode can be used to examine the within-period performance of the model without the complications of previous period errors. Use the STATIC option.
- The *NAHEAD*=n *dynamic simultaneous simulation* mode can be used to see how well *n*-period-ahead forecasting would have performed over the historical period. Use the NAHEAD=*n* option.

The different solution modes are explained in detail in the following sections.

Dynamic and Static Simulations

In model simulation, either solved values or actual values from the data set can be used to supply lagged values of an endogenous variable. A *dynamic* solution refers to a solution obtained by using only solved values for the lagged values. Dynamic mode is used both for forecasting and for simulating the dynamic properties of the model.

A *static* solution refers to a solution obtained by using the actual values when available for the lagged endogenous values. Static mode is used to simulate the behavior of the model without the complication of previous period errors. Dynamic simulation is the default.

If you want to use static values for lags only for the first *n* observations, and dynamic values thereafter, specify the START=*n* option. For example, if you want a dynamic simulation to start after observation twenty-four, specify START=24 on the SOLVE statement. If the model being simulated had a value lagged for four time periods, then this value would start using dynamic values when the simulation reached observation number 28.

n-Period-Ahead Forecasting

Suppose you want to regularly forecast 12 months ahead and produce a new forecast each month as more data becomes available. *n*-period-ahead forecasting allows you to test how well you would have done over time if you had been using your model to forecast one year ahead.

To see how well a model predicts *n* time periods in the future, perform an *n*-period-ahead forecast on real data and compare the forecast values with the actual values.

n-period-ahead forecasting refers to using dynamic values for the lagged endogenous variables only for lags *1* through *n–1*. For example, one-period-ahead forecasting, specified by the NAHEAD=1 option in the SOLVE statement, is the same as if a static solution had been requested. Specifying NAHEAD=2 produces a solution that uses dynamic values for lag one and static, actual, values for longer lags.

The following example is a two-year-ahead dynamic simulation. The output is shown in Figure 19.70.

```
data yearly;
   input year x1 x2 x3 y1 y2 y3;
   datalines;
84 4 9  0  7  4  5
85 5 6  1  1  27  4
86 3 8  2  5  8  2
87 2 10 3  0  10 10
88 4 7  6  20 60 40
89 5 4  8  40 40 40
90 3 2  10 50 60 60
91 2 5  11 40 50 60
;
run;

proc model data=yearly outmodel=yearlyModel;
   endogenous y1 y2 y3;
   exogenous  x1 x2 x3;

   y1 = 2 + 3*x1 - 2*x2 + 4*x3;
   y2 = 4 + lag2( y3 ) + 2*y1 + x1;
   y3 = lag3( y1 ) + y2 - x2;
```

```
   solve y1 y2 y3 / nahead=2 out=c;
run;

proc print data=c;
run;
```

Figure 19.70 NAHEAD Summary Report

The MODEL Procedure
Dynamic Simultaneous 2-Periods-Ahead Forecasting Simulation

Data Set Options	
DATA=	YEARLY
OUT=	C

Solution Summary	
Variables Solved	3
Simulation Lag Length	3
Solution Method	NEWTON
CONVERGE=	1E-8
Maximum CC	0
Maximum Iterations	1
Total Iterations	8
Average Iterations	1

Observations Processed	
Read	20
Lagged	12
Solved	8
First	5
Last	8

Variables Solved For	y1 y2 y3

The C data set is shown in Figure 19.71:

Figure 19.71 C Data Set

Obs	_TYPE_	_MODE_	_LAG_	_ERRORS_	y1	y2	y3	x1	x2	x3
1	PREDICT	SIMULATE	0	0	0	10	7	2	10	3
2	PREDICT	SIMULATE	1	0	24	58	52	4	7	6
3	PREDICT	SIMULATE	1	0	41	101	102	5	4	8
4	PREDICT	SIMULATE	1	0	47	141	139	3	2	10
5	PREDICT	SIMULATE	1	0	42	130	145	2	5	11

The preceding two-year-ahead simulation can be emulated without using the NAHEAD= option by the following PROC MODEL statements:

```
proc model data=yearly model=yearlyModel;
  range year = 87 to 88;
  solve y1 y2 y3 / dynamic solveprint;
run;

  range year = 88 to 89;
  solve y1 y2 y3 / dynamic solveprint;
run;

  range year = 89 to 90;
  solve y1 y2 y3 / dynamic solveprint;
run;

  range year = 90 to 91;
  solve y1 y2 y3 / dynamic solveprint;
```

The totals shown under "Observations Processed" in Figure 19.70 are equal to the sum of the four individual runs.

Simulation and Forecasting

You can perform a simulation of your model or use the model to produce forecasts. *Simulation* refers to the determination of the endogenous or dependent variables as a function of the input values of the other variables, even when actual data for some of the solution variables are available in the input data set. The simulation mode is useful for verifying the fit of the model parameters. Simulation is selected by the SIMULATE option in the SOLVE statement. Simulation mode is the default.

In forecast mode, PROC MODEL solves only for those endogenous variables that are missing in the data set. The actual value of an endogenous variable is used as the solution value whenever nonmissing data for it is available in the input data set. Forecasting is selected by the FORECAST option in the SOLVE statement.

For example, an econometric forecasting model can contain an equation to predict future tax rates, but tax rates are usually set in advance by law. Thus, for the first year or so of the forecast, the predicted tax rate should really be exogenous. Or, you might want to use a prior forecast of a certain variable from a short-run forecasting model to provide the predicted values for the earlier periods of a longer-range forecast of a long-run model. A common situation in forecasting is when historical data needed to fill the initial lags of a dynamic model are available for some of the variables but have not yet been obtained for others. In this case, the forecast must start in the past to supply the missing initial lags. Clearly, you should use the actual data that are available for the lags. In all the preceding cases, the forecast should be produced by running the model in the FORECAST mode; simulating the model over the future periods would not be appropriate.

Monte Carlo Simulation

The accuracy of the forecasts produced by PROC MODEL depends on four sources of error (Pindyck and Rubinfeld 1981, pp. 405–406):

- The system of equations contains an implicit random error term ϵ

  $$\mathbf{g}(\mathbf{y}, \mathbf{x}, \hat{\boldsymbol{\theta}}) = \boldsymbol{\epsilon}$$

  where $\mathbf{y}$, $\mathbf{x}$, $\mathbf{g}$, $\hat{\boldsymbol{\theta}}$, and $\boldsymbol{\epsilon}$ are vector valued.

- The estimated values of the parameters, $\hat{\boldsymbol{\theta}}$, are themselves random variables.
- The exogenous variables might have been forecast themselves and therefore might contain errors.
- The system of equations might be incorrectly specified; the model only approximates the process modeled.

The RANDOM= option is used to request Monte Carlo (or stochastic) simulations to generate confidence intervals for errors that arise from the first two sources. The Monte Carlo simulations can be performed with $\boldsymbol{\epsilon}$, $\boldsymbol{\theta}$, or both vectors represented as random variables. The SEED= option is used to control the random number generator for the simulations. SEED=0 forces the random number generator to use the system clock as its seed value.

In Monte Carlo simulations, repeated simulations are performed on the model for random perturbations of the parameters and the additive error term. The random perturbations follow a multivariate normal distribution with expected value of 0 and covariance described by a covariance matrix of the parameter estimates in the case of θ, or a covariance matrix of the equation residuals for the case of ϵ. PROC MODEL can generate both covariance matrices or you can provide them.

The ESTDATA= option specifies a data set that contains an estimate of the covariance matrix of the parameter estimates to use for computing perturbations of the parameters. The ESTDATA= data set is usually created by the FIT statement with the OUTEST= and OUTCOV options. When the ESTDATA= option is specified, the matrix read from the ESTDATA= data set is used to compute vectors of random shocks or perturbations for the parameters. These random perturbations are computed at the start of each repetition of the solution and added to the parameter values. The perturbed parameters are fixed throughout the solution range. If the covariance matrix of the parameter estimates is not provided, the parameters are not perturbed.

The SDATA= option specifies a data set that contains the covariance matrix of the residuals to use for computing perturbations of the equations. The SDATA= data set is usually created by the FIT statement with the OUTS= option. When SDATA= is specified, the matrix read from the SDATA= data set is used to compute vectors of random shocks or perturbations for the equations. These random perturbations are computed at each observation. The simultaneous solution satisfies the model equations plus the random shocks. That is, the solution is not a perturbation of a simultaneous solution of the structural equations; rather, it is a simultaneous solution of the stochastic equations by using the simulated errors. If the SDATA= option is not specified, the random shocks are not used.

The different random solutions are identified by the _REP_ variable in the OUT= data set. An unperturbed solution with _REP_=0 is also computed when the RANDOM= option is used. RANDOM=*n* produces *n* +1 solution observations for each input observation in the solution range. If the RANDOM= option is not specified, the SDATA= and ESTDATA= options are ignored, and no Monte Carlo simulation is performed.

PROC MODEL does not have an automatic way of modeling the exogenous variables as random variables for Monte Carlo simulation. If the exogenous variables have been forecast, the error bounds for these variables should be included in the error bounds generated for the endogenous variables. If the models for the exogenous variables are included in PROC MODEL, then the error bounds created from a Monte Carlo simulation contain the uncertainty due to the exogenous variables.

Alternatively, if the distribution of the exogenous variables is known, the built-in random number generator functions can be used to perturb these variables appropriately for the Monte Carlo simulation. For example, if you know the forecast of an exogenous variable, X, has a standard error of 5.2 and the error is normally distributed, then the following statements can be used to generate random values for X:

```
x_new = x + 5.2 * rannor(456);
```

During a Monte Carlo simulation, the random number generator functions produce one value at each observation. It is important to use a different seed value for all the random number generator functions in the model program; otherwise, the perturbations will be correlated. For the unperturbed solution, _REP_=0, the random number generator functions return 0.

PROC UNIVARIATE can be used to create confidence intervals for the simulation (see the Monte Carlo simulation example in the section "Getting Started: MODEL Procedure" on page 1070).

Multivariate *t* Distribution Simulation

To perform a Monte Carlo analysis of models that have residuals distributed as a multivariate *t*, use the ERRORMODEL statement with either the ~ t(*variance*, *df*) option or with the CDF=t(*variance*, *df*) option. The CDF= option specifies the distribution that is used for simulation so that the estimation can be done for one set of distributional assumptions and the simulation for another.

The following is an example of estimating and simulating a system of equations with *t* distributed errors by using the ERRORMODEL statement.

```
/* generate simulation data set */
data five;
   set xfrate end=last;
   if last then do;
      todate = date +5;
      do date = date to todate;
         output;
      end;
   end;
run;
```

The preceding DATA step generates the data set to request a five-days-ahead forecast. The following statements estimate and forecast the three forward-rate models of the following form.

$$
\begin{aligned}
rate_t &= rate_{t-1} + \mu * rate_{t-1} + \nu \\
\nu &= \sigma * rate_{t-1} * \epsilon \\
\epsilon &\sim \mathrm{N}(0,1)
\end{aligned}
$$

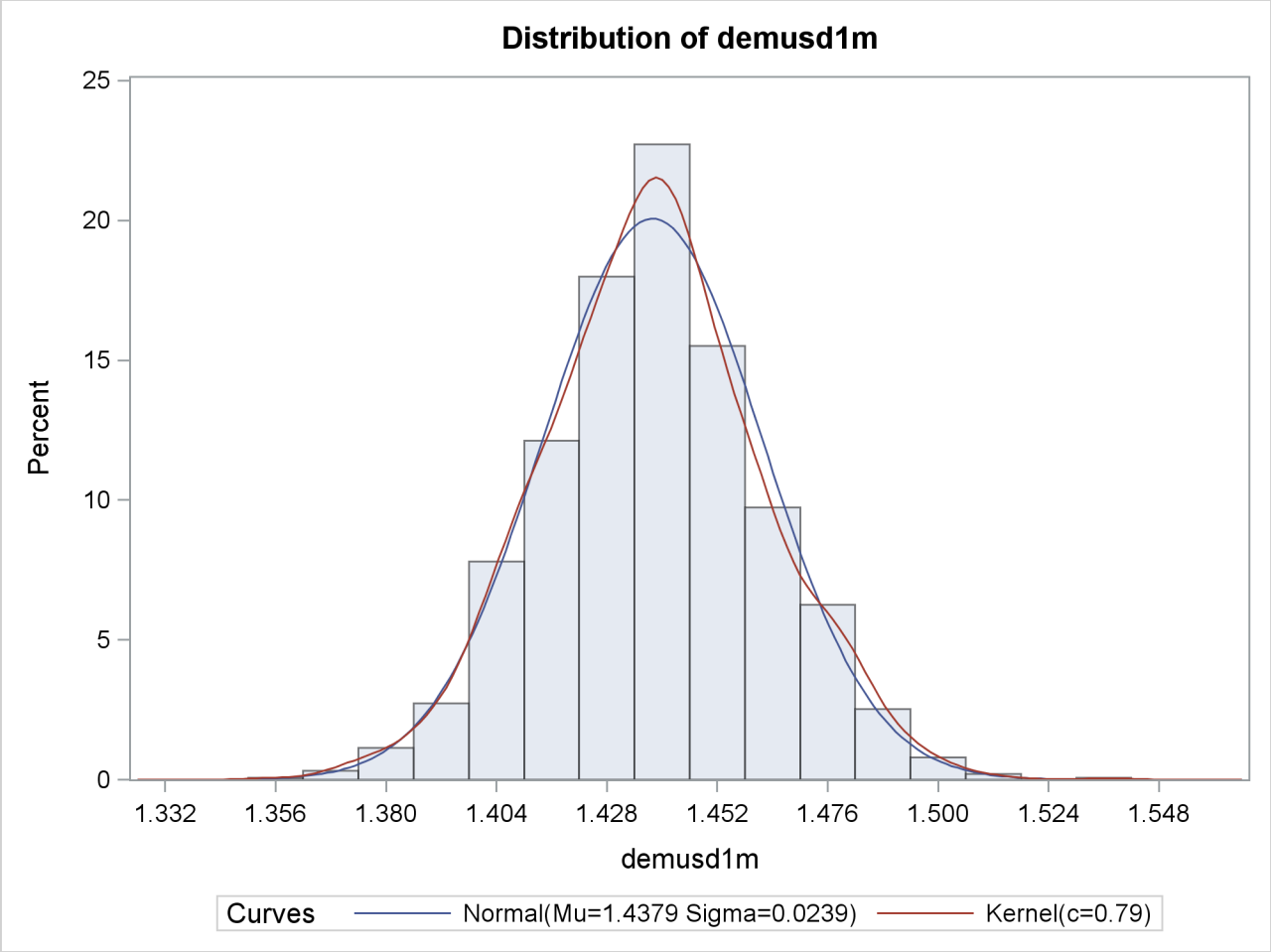
```
title "Daily Multivariate Geometric Brownian Motion Model "
        "of D-Mark/USDollar Forward Rates";

proc model data=xfrate;

   parms df 15;       /* Give initial value to df */

   demusd1m = lag(demusd1m) + mu1m * lag(demusd1m);
   var_demusd1m = sigma1m ** 2 * lag(demusd1m **2);
   demusd3m = lag(demusd3m) + mu3m * lag(demusd3m);
   var_demusd3m = sigma3m ** 2 * lag(demusd3m ** 2);
```

```
   demusd6m = lag(demusd6m) + mu6m * lag(demusd6m);
   var_demusd6m = sigma6m ** 2 * lag(demusd6m ** 2);

      /* Specify the error distribution */
   errormodel demusd1m demusd3m demusd6m
        ~ t( var_demusd1m var_demusd3m var_demusd6m, df );

      /* output normalized S matrix */
   fit demusd1m demusd3m demusd6m / outsn=s;
run;
      /* forecast five days in advance */
   solve demusd1m demusd3m demusd6m /
         data=five sdata=s seed=1 random=1500 out=monte;
   id date;
run;

   /* select out the last date ---*/
data monte; set monte;
   if date = '10dec95'd then output;
run;

title "Distribution of demusd1m Five Days Ahead";
proc univariate data=monte noprint;
   var demusd1m;
   histogram demusd1m /
             normal(noprint color=red)
             kernel(noprint color=blue) cfill=ligr;
run;
```

The Monte Carlo simulation specified in the preceding example draws from a multivariate t distribution with constant degrees of freedom and forecasted variance, and it computes future states of DEMUSD1M, DEMUSD3M, and DEMUSD6M. The OUTSN= option in the FIT statement is used to specify the data set for the normalized $\mathbf{\Sigma}$ matrix. That is, the $\mathbf{\Sigma}$ matrix is created by crossing the normally distributed residuals. The normally distributed residuals are created from the t distributed residuals by using the normal inverse CDF and the t CDF. This matrix is a correlation matrix.

The distribution of DEMUSD1M on the fifth day is shown in the Figure 19.72. The two curves overlaid on the graph are a kernel density estimation and a normal distribution fit to the results.

Figure 19.72 Distribution of DEMUSD1M



Alternate Distribution Simulation

As an alternate to the normal distribution, the ERRORMODEL statement can be used in a simulation to specify other distributions. The distributions available for simulation are Cauchy, chi-squared, *F*, Poisson, *t*, and uniform. An empirical distribution can also be used if the residuals are specified by using the RESIDDATA= option in the SOLVE statement.

Except for the *t* distribution, all of these alternate distributions are univariate but can be used together in a multivariate simulation. The ERRORMODEL statement applies to solved for equations only. That is, the normal form or general form equation referred to by the ERRORMODEL statement must be one of the equations you have selected in the SOLVE statement.

In the following example, two Poisson distributed variables are used to simulate the calls that arrive at and leave a call center.

```
data s;    /* Covariance between arriving and leaving */
   arriving = 1; leaving = 0.7; _name_= "arriving";
   output;
   arriving = 0.7; leaving = 1.0; _name_= "leaving";
   output;
run;

data calls;
   date = '20mar2001'd;
   output;
run;
```

The first DATA step generates a data set that contains a covariance matrix for the ARRIVING and LEAVING variables. The covariance is

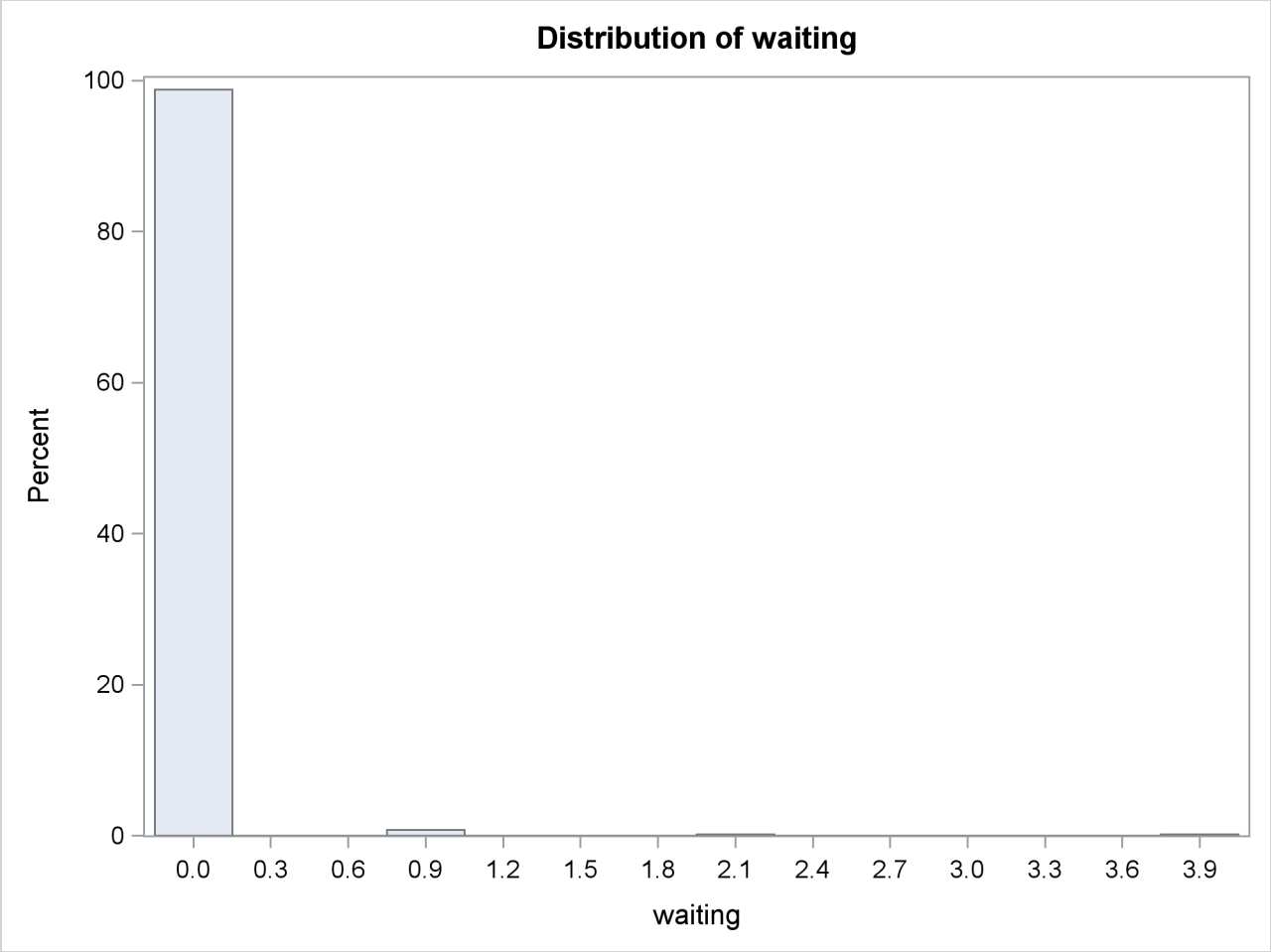
$$\begin{vmatrix} 1 & .7 \\ .7 & 1 \end{vmatrix}$$

The following statements create the number of waiting clients data:

```
proc model data=calls;
   arriving = 0;
   errormodel arriving ~ poisson( 10 );
   leaving  = 4;
   errormodel leaving ~ poisson( 11 );

   waiting = arriving - leaving;
   if waiting < 0 then waiting=0;
   outvars waiting;

   solve arriving leaving / seed=1 random=500 sdata=s out=sim;
run;

title "Distribution of Clients Waiting";
proc univariate data=sim noprint;
   var waiting ;
   histogram waiting / cfill=ligr;
run;
```

The distribution of number of waiting clients is shown in Figure 19.73.

Figure 19.73 Distribution of Number of Clients Waiting



Mixtures of Distributions—Copulas

The theory of copulas is what enables the MODEL procedure to combine and simulate multivariate distributions with different marginals. This section provides a brief overview of copulas.

Modeling a system of variables accurately is a difficult task. The underlying, ideal, distributional assumptions for each variable are usually different from each other. An individual variable might be best modeled as a *t* distribution or as a Poisson process. The correlation of the various variables are very important to estimate as well. A joint estimation of a set of variables would make it possible to estimate a correlation structure but would restrict the modeling to single, simple multivariate distribution (for example, the normal). Even with a simple multivariate distribution, the joint estimation would be computationally difficult and would have to deal with issues of missing data.

By using the MODEL procedure ERRORMODEL statement, you can combine and simulate from models of different distributions. The covariance matrix for the combined model is constructed by using the copula induced by the multivariate normal distribution. A copula is a function that couples joint distributions to their marginal distributions.

By default, the copula used in the MODEL procedure is based on the multivariate normal. This particular multivariate normal has zero mean and covariance matrix **R**. The user provides **R**, which can be created by using the following steps:

1. Each model is estimated separately and their residuals are saved.
2. The residuals for each model are converted to a normal distribution by using their CDFs, $F_i(.)$, using the relationship $\Phi^{-1}(F(\epsilon_{it}))$.
3. These normal residuals are crossed to create a covariance matrix **R**.

If the model of interest can be estimated jointly, such as multivariate T, then the OUTSN= option can be used to generate the correct covariance matrix.

A draw from this mixture of distributions is created by using the following steps that are performed automatically by the MODEL procedure.

1. Independent $N(0, 1)$ variables are generated.
2. These variables are transformed to a correlated set by using the covariance matrix **R**.
3. These correlated normals are transformed to a uniform by using $\Phi()$.
4. $F^{-1}()$ is used to compute the final sample value.

Alternate Copulas

The Gaussian, *t*, and the normal mixture copula are available in the MODEL procedure. These copulas support asymmetric parameters and can use alternate estimation methods for creating the base covariance matrix.

The normal (Gaussian) copula is the default. A draw from a Gaussian copula is obtained from

$$\mathbf{x} = \mathbf{A}\mathbf{z}$$

where $\mathbf{z} \in R^d$ is a vector of independent random normal(0, 1) draws, $\mathbf{A} \in R^{d \times d}$ is the square root of the covariance matrix, **R**. For the normal mixture and *t* copula, a draw is created as

$$\mathbf{x} = w\boldsymbol{\gamma} + \sqrt{w}\mathbf{A}\mathbf{z}$$

where w is a scalar random variable and $\boldsymbol{\gamma} \in R^d$ is a vector of asymmetry parameters. $\boldsymbol{\gamma}$ is specified in the SDATA= data set. If $W \sim$inverse gamma$(df/2, df/2)$, then **x** is multivariate *t* or skewed *t* if $\boldsymbol{\gamma}$ is provided. When NORMALMIX is specified, w is distributed as a step function with each of the n positive variances, $v_1 \ldots v_n$, having probability $p_1 \ldots p_n$.

The covariance matrix $\mathbf{R} = \mathbf{A}'\mathbf{A}$ is specified with the SDATA= option. The vector of asymmetry parameters, $\boldsymbol{\gamma}$, defaults to zero or is specified in the SDATA= data set with _TYPE_=ASYM. The ASYM option specifies that the nonzero asymmetry vector, $\boldsymbol{\gamma}$, is to be used.

In the event the covariance matrix specified with the SDATA= option is not positive semidefinite the matrix is modified to be positive semidefinite. See Rebonato and Jäckel (1999) for more details.

The actual draw for an individual variable, y_i, depends on the marginal distribution of the variable, $\tilde{F}$, and the chosen copula F as

$$y_i = \tilde{F}_i^{-1}(F(x_i))$$

Archimedean Copulas

The three Archimedean copulas available in the MODEL procedure are the Clayton, Gumbel, and Frank copulas. Archimedean copulas require only a single parameter, θ, to define the joint distribution's covariance structure for a simulation problem, Therefore, a covariance matrix is not required to perform simulations that use Archimedean copulas, and the SDATA= option does not have to be specified for these simulations. For more information about Archimedean copulas, including the functional forms of the Clayton, Gumbel, and Frank copulas, see the section "Archimedean Copulas" in Chapter 10, "The COPULA Procedure."

Asymmetrical Copula Example

In this example, an asymmetrical *t* copula is used to correlate two uniform distributions. The asymmetrical parameter is varied over a range of values to demonstrate its effect. The resulting graphs is produced by using ODS graphics.

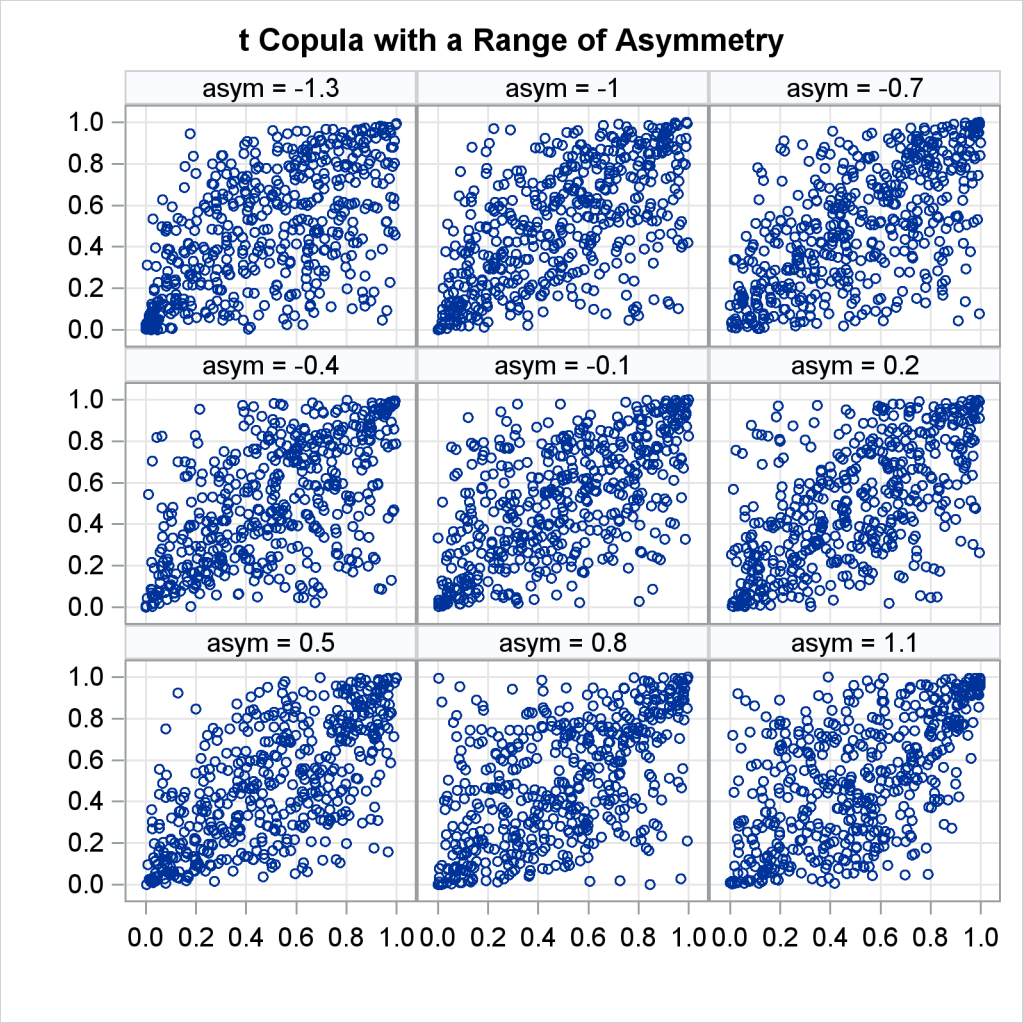
```
data histdata;
   do asym = -1.3 to 1.1 by .3;
      date='01aug2007'd;
      y = .5;
      z = .5;
      output;
   end;
run ;

  /* Add the asymmetric parameter to cov mat */
data asym;
   do asym = -1.3 to 1.1 by .3;
      y = asym;
      z = 0;
      _name_ = "    ";
      _type_ = "asym";
      output;
      y = 1;
      z = .65;
      _name_ = "y";
      _type_ = "cov";
      output;
      y = .65;
      z = 1;
      _name_ = "z";
      _type_ = "cov";
      output;
   end;
run;
```

```
proc model out=sim(where=(_REP_ > 0)) data=histdata sdata=asym;
   y = 0;
   errormodel y ~ Uniform(0,1);

   z = 0;
   errormodel z ~ Uniform(0,1);

   solve y z / random=500 seed=12345 copula=(t(5) asym );
   by asym;
run;
```

To produce a panel plot of this joint distribution, use the following SAS/GRAPH statements.

```
ods graphics on / height=800 width=800;
proc template;
  define statgraph myplot.panel;
  BeginGraph;
    entrytitle halign=left halign=center
          textattrs=GRAPHTITLETEXT "t Copula with a Range of Asymmetry";

    layout datapanel classvars=(asym) / rows=3 columns=3
          order=rowmajor height=1024 width=1420
          rowaxisopts=(griddisplay=on label=' ')
          columnaxisopts=(griddisplay=on label=' ');
       layout prototype;
          scatterplot x=z y=y ;
       endlayout;
    endlayout;
  EndGraph;
  end;
run;

proc sgrender data=sim template='myplot.panel';
run;
```

Figure 19.74 *t* Copula with Asymmetry



Quasi-Random Number Generators

Traditionally high-discrepancy pseudo-random number generators are used to generate innovations in Monte Carlo simulations. Loosely translated, a high-discrepancy pseudo-random number generator is one in which there is very little correlation between the current number generated and the past numbers generated. This property is ideal if indeed independence of the innovations is required. If, on the other hand, the efficient spanning of a multidimensional space is desired, a low discrepancy, quasi-random number generator can be used. A quasi-random number generator produces numbers that have no random component.

A simple one-dimensional quasi-random sequence is the van der Corput sequence. Given a prime number r ($r \geq 2$), any integer has a unique representation in terms of base r. A number in the interval [0,1) can be created by inverting the representation base power by base power. For example, consider r=3 and n=1, 1 in base 3 is

$$1_{10} = 1 \cdot 3^0 = 1_3$$

When the powers of 3 are inverted,

$$\phi(1) = \frac{1}{3}$$

Also, 11 in base 3 is

$$11_{10} = 1 \cdot 3^2 + 2 \cdot 3^0 = 102_3$$

When the powers of 3 are inverted,

$$\phi(11) = \frac{1}{9} + 2 \cdot \frac{1}{3} = \frac{7}{9}$$

The first 10 numbers in this sequence $\phi(1) \ldots \phi(10)$ are provided below

$$0, \frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{4}{9}, \frac{7}{9}, \frac{2}{9}, \frac{5}{9}, \frac{8}{9}, \frac{1}{27}$$

As the sequence proceeds, it fills in the gaps in a uniform fashion.

Several authors have expanded this idea to many dimensions. Two versions supported by the MODEL procedure are the Sobol sequence (QUASI=SOBOL) and the Faure sequence (QUASI=FAURE). The Sobol sequence is based on binary numbers and is generally computationally faster than the Faure sequence. The Faure sequence uses the dimensionality of the problem to determine the number base to use to generate the sequence. The Faure sequence has better distributional properties than the Sobol sequence for dimensions greater than 8.

As an example of the difference between a pseudo-random number and a quasi-random number, consider simulating a bivariate normal with 100 draws.

Figure 19.75 Kernel Density of a Bivariate Normal produced by 100 Faure-Random Draws

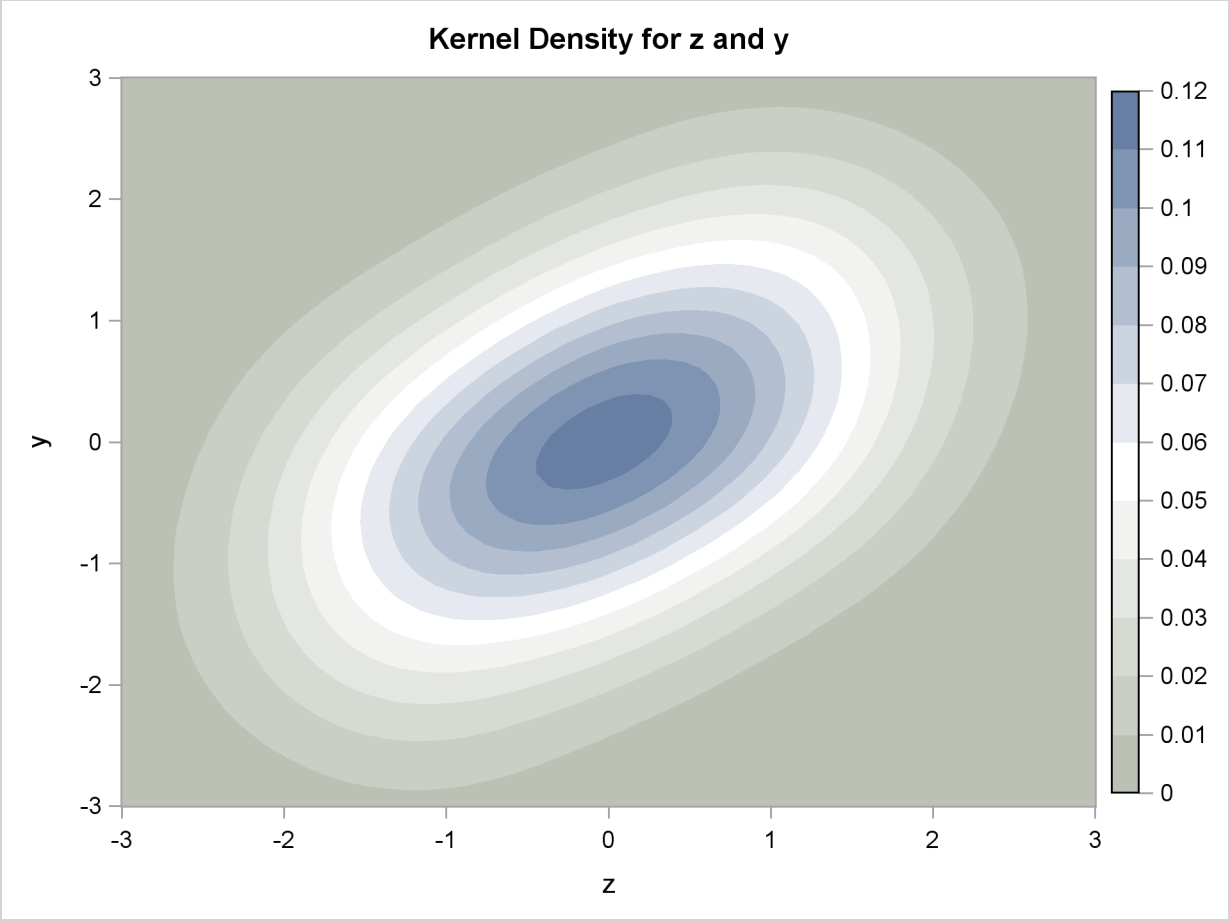
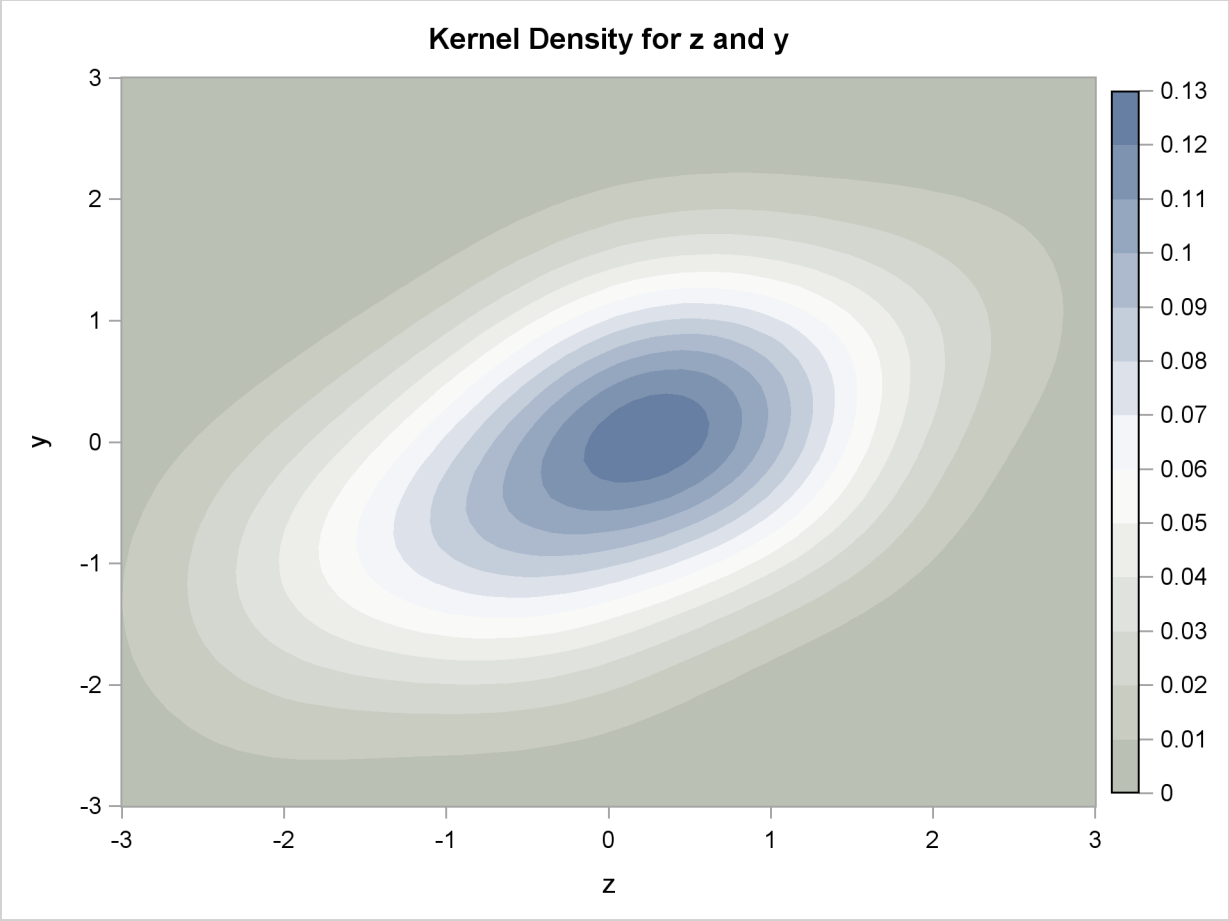


Figure 19.76 Kernel Density of a Bivariate Normal produced by 100 Pseudo-Random Draws



Solution Mode Output

The following SAS statements dynamically forecast the solution to a nonlinear equation:

```
proc model data=sashelp.citimon;
   parameters a 0.010708  b  -0.478849 c 0.929304;
   lhur = 1/(a * ip) + b + c * lag(lhur);
   solve lhur / out=sim forecast dynamic;
run;
```

The first page of output produced by the SOLVE step is shown in Figure 19.77. This is the summary description of the model. The error message states that the simulation was aborted at observation 144 because of missing input values.

Figure 19.77 Solve Step Summary Output

The MODEL Procedure

Model Summary	
Model Variables	1
Parameters	3
Equations	1
Number of Statements	1
Program Lag Length	1

Model Variables	LHUR
Parameters(Value)	a(0.010708) b(-0.478849) c(0.929304)
Equations	LHUR

The second page of output, shown in Figure 19.78, gives more information on the failed observation.

Figure 19.78 Solve Step Error Message

The MODEL Procedure
Dynamic Single-Equation Forecast

Error: Solution values are missing because of missing input values for observation 144 at NEWTON iteration 0.

Note: Additional information on the values of the variables at this observation, which may be helpful in determining the cause of the failure of the solution process, is printed below.

Observation	144	**Iteration**	0	**CC**	-1.000000
		Missing	1		

Iteration Errors - Missing.

```
                    The MODEL Procedure
                  Dynamic Single-Equation Forecast

                  --- Listing of Program Data Vector ---
_N_:                144     ACTUAL.LHUR:          .     ERROR.LHUR:           .
IP:                   .     LHUR:           7.10000     PRED.LHUR:            .
a:              0.01071     b:             -0.47885     c:              0.92930
```

Note: Simulation aborted.

From the program data vector, you can see the variable IP is missing for observation 144. LHUR could not be computed, so the simulation aborted.

The solution summary table is shown in Figure 19.79.

Figure 19.79 Solution Summary Report

The MODEL Procedure
Dynamic Single-Equation Forecast

Data Set Options	
DATA=	SASHELP.CITIMON
OUT=	SIM

Solution Summary	
Variables Solved	1
Forecast Lag Length	1
Solution Method	NEWTON
CONVERGE=	1E-8
Maximum CC	0
Maximum Iterations	1
Total Iterations	143
Average Iterations	1

Observations Processed	
Read	145
Lagged	1
Solved	143
First	2
Last	145
Failed	1

Variables Solved For	LHUR

This solution summary table includes the names of the input data set and the output data set followed by a description of the model. The table also indicates that the solution method defaulted to Newton's method. The remaining output is defined as follows.

Maximum CC	is the maximum convergence value accepted by the Newton procedure. This number is always less than the value for the CONVERGE= option.
Maximum Iterations	is the maximum number of Newton iterations performed at each observation and each replication of Monte Carlo simulations.
Total Iterations	is the sum of the number of iterations required for each observation and each Monte Carlo simulation.
Average Iterations	is the average number of Newton iterations required to solve the system at each step.
Solved	is the number of observations used times the number of random replications selected plus one, for Monte Carlo simulations. The one additional simulation is the original unperturbed solution. For simulations that do not involve Monte Carlo, this number is the number of observations used.

Summary Statistics

The STATS and THEIL options are used to select goodness-of-fit statistics. Actual values must be provided in the input data set for these statistics to be printed. When the RANDOM= option is specified, the statistics do not include the unperturbed (_REP_=0) solution.

STATS Option Output

The following statements show the addition of the STATS and THEIL options to the model in the previous section:

```
proc model data=sashelp.citimon;
   parameters a 0.010708  b  -0.478849 c 0.929304;
   lhur= 1/(a * ip) + b + c * lag(lhur) ;
   solve lhur / out=sim dynamic stats theil;
   range date to '01nov91'd;
run;
```

The STATS output in Figure 19.80 and the THEIL output in Figure 19.81 are generated.

Figure 19.80 STATS Output

The MODEL Procedure
Dynamic Single-Equation Simulation

Solution Range DATE = FEB1980 To NOV1991

Descriptive Statistics							
			Actual		Predicted		
Variable	N Obs	N	Mean	Std Dev	Mean	Std Dev	Label
LHUR	142	142	7.0887	1.4509	7.2473	1.1465	UNEMPLOYMENT RATE: ALL WORKERS, 16 YEARS

Figure 19.80 *continued*

Statistics of fit									
Variable	N	Mean Error	Mean % Error	Mean Abs Error	Mean Abs % Error	RMS Error	RMS % Error	R-Square	Label
LHUR	142	0.1585	3.5289	0.6937	10.0001	0.7854	11.2452	0.7049	UNEMPLOYMENT RATE: ALL WORKERS, 16 YEARS

The number of observations (Nobs), the number of observations with both predicted and actual values nonmissing (N), and the mean and standard deviation of the actual and predicted values of the determined variables are printed first. The next set of columns in the output are defined as follows:

Mean Error	$\frac{1}{N}\sum_{j=1}^{N}(\hat{y}_j - y_j)$
Mean % Error	$\frac{100}{N}\sum_{j=1}^{N}(\hat{y}_j - y_j)/y_j$
Mean Abs Error	$\frac{1}{N}\sum_{j=1}^{N}\lvert\hat{y}_j - y_j\rvert$
Mean Abs % Error	$\frac{100}{N}\sum_{j=1}^{N}\lvert(\hat{y}_j - y_j)/y_j\rvert$
RMS Error	$\sqrt{\frac{1}{N}\sum_{j=1}^{N}(\hat{y}_j - y_j)^2}$
RMS % Error	$100\sqrt{\frac{1}{N}\sum_{j=1}^{N}((\hat{y}_j - y_j)/y_j)^2}$
R-square	$1 - SSE/CSSA$
SSE	$\sum_{j=1}^{N}(\hat{y}_j - y_j)^2$
SSA	$\sum_{j=1}^{N}(y_j)^2$
CSSA	$SSA - \left(\sum_{j=1}^{N} y_j\right)^2$
$\hat{y}$	predicted value
y	actual value

When the RANDOM= option is specified, the statistics do not include the unperturbed (_REP_=0) solution.

THEIL Option Output

The THEIL option specifies that Theil forecast error statistics be computed for the actual and predicted values and for the relative changes from lagged values. Mathematically, the quantities are

$$\hat{yc} = (\hat{y} - lag(y))/lag(y)$$

$$yc = (y - lag(y))/lag(y)$$

where $\hat{yc}$ is the relative change for the predicted value and yc is the relative change for the actual value.

Figure 19.81 THEIL Output

Theil Forecast Error Statistics

			MSE Decomposition Proportions						Inequality Coef		
Variable	N	MSE	Corr (R)	Bias (UM)	Reg (UR)	Dist (UD)	Var (US)	Covar (UC)	U1	U	Label
LHUR	142	0.6168	0.85	0.04	0.01	0.95	0.15	0.81	0.1086	0.0539	UNEMPLOYMENT RATE: ALL WORKERS, 16 YEARS

Theil Relative Change Forecast Error Statistics

		Relative Change	MSE Decomposition Proportions						Inequality Coef		
Variable	N	MSE	Corr (R)	Bias (UM)	Reg (UR)	Dist (UD)	Var (US)	Covar (UC)	U1	U	Label
LHUR	142	0.0126	-0.08	0.09	0.85	0.06	0.43	0.47	4.1226	0.8348	UNEMPLOYMENT RATE: ALL WORKERS, 16 YEARS

The columns have the following meaning:

Corr (R) is the correlation coefficient, ρ, between the actual and predicted values.

$$\rho = \frac{\text{cov}(y, \hat{y})}{\sigma_a \sigma_p}$$

where σ_p and σ_a are the standard deviations of the predicted and actual values.

Bias (UM) is an indication of systematic error and measures the extent to which the average values of the actual and predicted deviate from each other.

$$\frac{(\text{E}(y) - \text{E}(\hat{y}))^2}{\frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2}$$

Reg (UR) is defined as $(\sigma_p - \rho * \sigma_a)^2 / MSE$. Consider the regression

$$y = \alpha + \beta \hat{y}$$

If $\hat{\beta} = 1$, UR will equal zero.

Dist (UD) is defined as $(1 - \rho^2)\sigma_a \sigma_a / MSE$ and represents the variance of the residuals obtained by regressing yc on $\hat{y}c$.

Var (US) is the variance proportion. US indicates the ability of the model to replicate the degree of variability in the endogenous variable.

$$US = \frac{(\sigma_p - \sigma_a)^2}{MSE}$$

Covar (UC) represents the remaining error after deviations from average values and average variabilities have been accounted for.

$$UC = \frac{2(1 - \rho)\sigma_p \sigma_a}{MSE}$$

U1 is a statistic that measures the accuracy of a forecast defined as follows:

$$U1 = \frac{\sqrt{MSE}}{\sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t)^2}}$$

U is the Theil's inequality coefficient defined as follows:

$$U = \frac{\sqrt{MSE}}{\sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t)^2} + \sqrt{\frac{1}{N}\sum_{t=1}^{N}(\hat{y}_t)^2}}$$

MSE is the mean square error. In the case of the relative change Theil statistics, the MSE is computed as follows:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(\hat{yc}_t - yc_t)^2$$

More information about these statistics can be found in the references Maddala (1977, pp. 344–347) and Pindyck and Rubinfeld (1981, pp. 364–365).

Goal Seeking: Solving for Right-Hand-Side Variables

The process of computing input values that are needed to produce target results is often called *goal seeking*. To compute a goal-seeking solution, use a SOLVE statement that lists the variables you want to solve for and provide a data set that contains values for the remaining variables.

Consider the following demand model for packaged rice

$$quantity\ demanded = \alpha_1 + \alpha_2 price^{2/3} + \alpha_3 income$$

where *price* is the price of the package and *income* is disposable personal income. The only variable the company has control over is the price it charges for rice. This model is estimated by using the following simulated data and PROC MODEL statements:

```
data demand;
   do t=1 to 40;
      price = (rannor(10) +5) * 10;
      income = 8000 * t ** (1/8);
      demand = 7200 - 1054 * price ** (2/3) +
               7 * income + 100 * rannor(1);
      output;
   end;
run;

data goal;
   demand = 85000;
   income = 12686;
run;
```

The goal is to find the price the company would have to charge to meet a sales target of 85,000 units. To do this, a data set is created with a DEMAND variable set to 85000 and with an INCOME variable set to 12686, the last income value.

The desired price is then determined by using the following PROC MODEL statements:

```
proc model data=demand
           outmodel=demandModel;
   demand = a1 - a2 * price ** (2/3) + a3 * income;
   fit demand / outest=demest;
   solve price / estdata=demest data=goal solveprint;
run;
```

The SOLVEPRINT option prints the solution values, number of iterations, and final residuals at each observation. The SOLVEPRINT output from this solve is shown in Figure 19.82.

Figure 19.82 Goal Seeking, SOLVEPRINT Output

The MODEL Procedure
Single-Equation Simulation

Observation 1	**Iterations** 6	**CC** 0.000000	**ERROR.demand** 0.000000

Solution Values
price
33.59016

The output indicates that it took six Newton iterations to determine the PRICE of 33.5902, which makes the DEMAND value within 16E–11 of the goal of 85,000 units.

Consider a more ambitious goal of 100,000 units. The output shown in Figure 19.83 indicates that the sales target of 100,000 units is not attainable according to this model.

```
data goal;
   demand = 100000;
   income = 12686;
run;

proc model model=demandModel;
   solve price / estdata=demest data=goal solveprint;
run;
```

Figure 19.83 Goal Seeking, Convergence Failure

The MODEL Procedure
Single-Equation Simulation

Error: Could not reduce norm of residuals in 10 subiterations.

Error: The solution failed because 1 equations are missing or have extreme values for observation 1 at NEWTON iteration 1.

Observation 1	**Iteration** 1	**CC** -1.000000
	Missing 1	

Figure 19.83 *continued*

```
                          The MODEL Procedure
                        Single-Equation Simulation

                     --- Listing of Program Data Vector ---
_N_:                    12   ACTUAL.demand:     100000   ERROR.demand:           .
PRED.demand:             .   a1:           7126.437997   a2:           1040.841492
a3:               6.992694   demand:            100000   income:              12686
price:           -0.000172
@PRED.demand/@pri:            .
```

The program data vector with the error note indicates that even after 10 subiterations, the norm of the residuals could not be reduced. The sales target of 100,000 units are unattainable with the given model. You might need to reformulate your model or collect more data to more accurately reflect the market response.

Numerical Solution Methods

If the SINGLE option is not used, PROC MODEL computes values that simultaneously satisfy the model equations for the variables named in the SOLVE statement. PROC MODEL provides three iterative methods, Newton, Jacobi, and Seidel, for computing a simultaneous solution of the system of nonlinear equations.

Single-Equation Solution

For normalized form equation systems, the solution either can simultaneously satisfy all the equations or can be computed for each equation separately, by using the actual values of the solution variables in the current period to compute each predicted value. By default, PROC MODEL computes a simultaneous solution. The SINGLE option in the SOLVE statement selects single-equation solutions.

Single-equation simulations are often used to produce residuals (which estimate the random terms of the stochastic equations) rather than the predicted values themselves. If the input data and range are the same as those used for parameter estimation, a static single-equation simulation reproduces the residuals of the estimation.

Newton's Method

The NEWTON option in the SOLVE statement requests Newton's method to simultaneously solve the equations for each observation. Newton's method is the default solution method. Newton's method is an iterative scheme that uses the derivatives of the equations with respect to the solution variables, $\mathbf{J}$, to compute a change vector as

$$\Delta \mathbf{y}^i = \mathbf{J}^{-1}\mathbf{q}(\mathbf{y}^i, \mathbf{x}, \boldsymbol{\theta})$$

PROC MODEL builds and solves $\mathbf{J}$ by using efficient sparse matrix techniques. The solution variables $\boldsymbol{y}^i$ at the ith iteration are then updated as

$$\mathbf{y}^{i+1} = \mathbf{y}^i + d \times \Delta \mathbf{y}^i$$

where d is a damping factor between 0 and 1 chosen iteratively so that

$$\|\mathbf{q}(\mathbf{y}^{i+1}, \mathbf{x}, \boldsymbol{\theta})\| < \|\mathbf{q}(\mathbf{y}^{i}, \mathbf{x}, \boldsymbol{\theta})\|$$

The number of subiterations that are allowed for finding a suitable d is controlled by the MAXSUBITER= option. The number of iterations of Newton's method that are allowed for each observation is controlled by MAXITER= option. See Ortega and Rheinbolt (1970) for more details.

Optimization Method

The OPTIMIZE option in the SOLVE statement requests that an optimization algorithm be used to minimize a norm of the errors in equations subject to constraints on the solution variables. The OPTIMIZE method is the only solution method that supports constraints on solution variables that are specified using the BOUNDS and RESTRICT statements. Constraints are ignored by the other solution methods. The OPTIMIZE method performs the following optimization:

$$\begin{array}{rl} \text{minimize} & \|\mathbf{q}(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta})\| \\ \text{subject to} & \mathbf{y}_l \le \mathbf{y} \le \mathbf{y}_u \\ \text{and} & f(\mathbf{y}) \ge 0) \end{array}$$

The norm used in the minimization process is

$$\|\mathbf{q}(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta})\| = \mathbf{q}(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta})' \text{diag}(\mathbf{S})^{-1} \mathbf{q}(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta})$$

where the $\mathbf{S}$ matrix is the covariance of equation errors that is specified by the SDATA= option in the SOLVE statement. If no SDATA= option is specified, the identity matrix is used. Both strict inequality and inequality constraints on the solution variables can be imposed using the BOUNDS or RESTRICT statement. For bounded problems, each lower and upper strict inequality is transformed into an inequality by using the equations

$$\begin{aligned} y_l &= (y_{\text{lower strict}} + \epsilon)/(1 - \epsilon) \\ y_u &= (y_{\text{upper strict}} - \epsilon)/(1 + \epsilon) \end{aligned}$$

When strict inequality expressions are imposed using the RESTRICT statement, these expressions are transformed into an inequality by using the equation

$$f(\mathbf{y}) = (f_{\text{strict}}(\mathbf{y}) + \epsilon)/(1 - \epsilon)$$

where $f_{\text{strict}}(\mathbf{y})$ is a nonlinear strict inequality constraint. The tolerance ϵ is controlled by the EPSILON= option in the SOLVE statement and defaults to 10^{-8}. To achieve the best performance from the minimization algorithm, both the first and second analytic derivatives of the equation errors with respect to the solution variables are used to compute the gradient and second derivatives of the objective function, $\|\mathbf{q}(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta})\|$. Analytic derivatives of the restriction expressions that are used to specify constraints are also used in the minimization. The gradient of the objective function is

$$\nabla\|\mathbf{q}(\mathbf{y},\mathbf{x},\boldsymbol{\theta})\| = 2\,\mathbf{J}'\mathrm{diag}(\mathbf{S})^{-1}\mathbf{q}(\mathbf{y},\mathbf{x},\boldsymbol{\theta})$$

The matrix of second derivatives of the objective function with respect to the solution variables is

$$\frac{\partial^2\|\mathbf{q}(\mathbf{y},\mathbf{x},\boldsymbol{\theta})\|}{\partial\mathbf{y}^2} = 2\left(\mathbf{J}'\mathrm{diag}(\mathbf{S})^{-1}\mathbf{J} + \sum_{k=1}^{d}\frac{\partial^2 q_k(\mathbf{y},\mathbf{x},\boldsymbol{\theta})}{\partial\mathbf{y}^2}\mathrm{diag}(\mathbf{S})^{-1}q_k(\mathbf{y},\mathbf{x},\boldsymbol{\theta})\right)$$

where d is the number of equations.

The algorithm that is used to find a minimum of $\|\mathbf{q}(\mathbf{y},\mathbf{x},\boldsymbol{\theta})\|$ subject to bounds on the solution variables employs the interior point technique for nonlinear optimization problems. For further information about this optimization method, see Chapter 10, "The Nonlinear Programming Solver" (*SAS/OR User's Guide: Mathematical Programming*).

When constraints are active in a solution, the minimum value of the objective function, $\|\mathbf{q}(\mathbf{y},\mathbf{x},\boldsymbol{\theta})\|$, is typically greater than 0. The diagnostic quantities that are produced by the OUTOBJVALS and OUTVIOLATIONS options are available to help identify and characterize solutions that have active bounds constraints. The following program contains a boundary constraint that becomes active in steps 6, 8, 10, 12, 13, and 16 of a Monte Carlo simulation:

```
proc model data=d sdata=s;
   dependent rate stock;
   parms theta   0.2
         kappa   0.002
         sigma   0.4
         sinit   1
         vol     .1;
   id i;

   bounds rate >= 0;

   rate   = zlag(rate) + kappa*(theta - zlag(rate));
   h.rate = sigma**2 * zlag(rate);
   eq.stock = log(stock/sinit) - (rate + vol*vol/2);
   h.stock = vol**2;

   solve / optimize converge=1e-6 seed=1 random=1 out=o outobjvals outviolations;
quit;

proc print data=o(where=(_objval_>1e-6));
run;
```

Figure 19.84 shows how the OUTOBJVALS option can be used to identify simulation steps with an active bounds constraint, and how the OUTVIOLATIONS option can be used to determine that the RATE equation is not satisfied for those steps.

Figure 19.84 Objective Function and Violation Values

Obs	i	_TYPE_	_MODE_	_REP_	_ERRORS_	_OBJVAL_	rate	stock
51	6	PREDICT	SIMULATE	1	0	.000363415	0.000027	1.03050
52	6	VIOL	SIMULATE	1	0	.000363415	-0.019073	0.00000
55	8	PREDICT	SIMULATE	1	0	.000123866	0.000045	1.08828
56	8	VIOL	SIMULATE	1	0	.000123866	-0.011151	0.00000
59	10	PREDICT	SIMULATE	1	0	.000330766	0.000028	0.96248
60	10	VIOL	SIMULATE	1	0	.000330766	-0.018207	-0.00000
63	12	PREDICT	SIMULATE	1	0	.000034095	0.000086	0.85526
64	12	VIOL	SIMULATE	1	0	.000034095	-0.005895	-0.00000
65	13	PREDICT	SIMULATE	1	0	.000011997	0.000141	1.10514
66	13	VIOL	SIMULATE	1	0	.000011997	-0.003573	-0.00000
71	16	PREDICT	SIMULATE	1	0	.000118982	0.000046	1.07103
72	16	VIOL	SIMULATE	1	0	.000118982	-0.010931	0.00000

Jacobi Method

The JACOBI option in the SOLVE statement selects a matrix-free alternative to Newton's method. This method is the traditional nonlinear Jacobi method found in the literature. The Jacobi method as implemented in PROC MODEL substitutes predicted values for the endogenous variables and iterates until a fixed point is reached. Then necessary derivatives are computed only for the diagonal elements of the jacobian, $\mathbf{J}$.

If the normalized form equation is

$$\mathbf{y} = \mathbf{f}(\mathbf{y}, \mathbf{x}, \boldsymbol{\theta})$$

the Jacobi iteration has the form

$$\mathbf{y}^{i+1} = \mathbf{f}(\mathbf{y}^{i}, \mathbf{x}, \boldsymbol{\theta})$$

Seidel Method

The Seidel method is an order-dependent alternative to the Jacobi method. You select the Seidel method by specifying the SEIDEL option in the SOLVE statement. The Seidel method is like the Jacobi method, except that in the Seidel method the model is further edited to substitute the predicted values into the solution variables immediately after they are computed. The Seidel method thus differs from the other methods in that the values of the solution variables are not fixed within an iteration. With the other methods, the order of the equations in the model program makes no difference, but the Seidel method might work much differently when the equations are specified in a different sequence. This fixed-point method is the traditional nonlinear Seidel method found in the literature.

The iteration has the form

$$\mathbf{y}_j^{i+1} = \mathbf{f}(\hat{\mathbf{y}}^{i}, \mathbf{x}, \boldsymbol{\theta})$$

where $\mathbf{y}_j^{i+1}$ is the jth equation variable at the ith iteration and

$$\hat{\mathbf{y}}^{i} = (y_1^{i+1}, y_2^{i+1}, y_3^{i+1}, \ldots, y_{j-1}^{i+1}, y_j^{i}, y_{j+1}^{i}, \ldots, y_g^{i})'$$

If the model is recursive, and if the equations are in recursive order, the Seidel method converges at once. If the model is block-recursive, the Seidel method might converge faster if the equations are grouped by block and the blocks are placed in block-recursive order. The BLOCK option can be used to determine the block-recursive form.

Jacobi and Seidel Methods with General Form Equations

Jacobi and Seidel solution methods support general form equations.

There are two cases where derivatives are (automatically) computed. The first case is for equations with the solution variable on the right-hand side and on the left-hand side of the equation

$$y^i = f(\mathbf{x}, y^i)$$

In this case the derivative of ERROR.y with respect to y is computed, and the new y approximation is computed as

$$y^{i+1} = y^i - \frac{f(\mathbf{x}, y^i) - y^i}{\partial(f(\mathbf{x}, y^i) - y^i)/\partial y}$$

The second case is a system of equations that contains one or more EQ.var equations. In this case, the MODEL procedure assigns a unique solution variable to each equation if such an assignment exists. Use the DETAILS option in the SOLVE statement to print a listing of the assigned variables.

Once the assignment is made, the new y approximation is computed as

$$y^{i+1} = y^i - \frac{f(\mathbf{x}, \mathbf{y}^i) - y^i}{\partial(f(\mathbf{x}, \mathbf{y}^i) - y^i)/\partial y}$$

If k is the number of general form equations, then k derivatives are required.

The convergence properties of the Jacobi and Seidel solution methods remain significantly poorer than the default Newton's method.

Comparison of Methods

Newton's method is the default and should work better than the others for most small- to medium-sized models. The Seidel method is always faster than the Jacobi for recursive models with equations in recursive order. For very large models and some highly nonlinear smaller models, the Jacobi or Seidel methods can sometimes be faster. Newton's method uses more memory than the Jacobi or Seidel methods.

Both the Newton's method and the Jacobi method are order-invariant in the sense that the order in which equations are specified in the model program has no effect on the operation of the iterative solution process. In order-invariant methods, the values of the solution variables are fixed for the entire execution of the model program. Assignments to model variables are automatically changed to assignments to corresponding equation variables. Only after the model program has completed execution are the results used to compute the new solution values for the next iteration.

Troubleshooting Problems

In solving a simultaneous nonlinear dynamic model you might encounter some of the following problems.

Missing Values

For SOLVE tasks, there can be no missing parameter values. Missing right-hand-side variables result in missing left-hand-side variables for that observation.

Unstable Solutions

A solution might exist but be unstable. An unstable system can cause the Jacobi and Seidel methods to diverge.

Explosive Dynamic Systems

A model might have well-behaved solutions at each observation but be dynamically unstable. The solution might oscillate wildly or grow rapidly with time.

Propagation of Errors

During the solution process, solution variables can take on values that cause computational errors. For example, a solution variable that appears in a LOG function might be positive at the solution but might be given a negative value during one of the iterations. When computational errors occur, missing values are generated and propagated, and the solution process might collapse.

Convergence Problems

The following items can cause convergence problems:

- There are illegal function values (for example $\sqrt{-1}$).
- There are local minima in the model equation.
- No solution exists.
- Multiple solutions exist.
- Initial values are too far from the solution.
- The CONVERGE= value is too small.

When PROC MODEL fails to find a solution to the system, the current iteration information and the program data vector are printed. The simulation halts if actual values are not available for the simulation to proceed. Consider the following program, which produces the output shown in Figure 19.85:

```
data test1;
   do t=1 to 50;
      x1 = sqrt(t) ;
      y = .;
      output;
   end;

proc model data=test1;
```

```
   exogenous x1 ;
   control a1 -1 b1 -29 c1 -4 ;
   y = a1 * sqrt(y) + b1 * x1 * x1 + c1 * lag(x1);
   solve y / out=sim forecast dynamic ;
run;
```

Figure 19.85 SOLVE Convergence Problems

The MODEL Procedure
Dynamic Single-Equation Forecast

Error: Could not reduce norm of residuals in 10 subiterations.

Error: The solution failed because 1 equations are missing or have extreme values for observation 1 at NEWTON iteration 1.

Note: Additional information on the values of the variables at this observation, which may be helpful in determining the cause of the failure of the solution process, is printed below.

Observation	1	**Iteration**	1	**CC**	-1.000000
		Missing	1		

Iteration Errors - Missing.

```
                        The MODEL Procedure
                    Dynamic Single-Equation Forecast

                   --- Listing of Program Data Vector ---
_N_:                 12     ACTUAL.x1:    1.41421     ACTUAL.y:            .
ERROR.y:              .     PRED.y:              .     a1:                 -1
b1:                 -29     c1:                 -4     x1:            1.41421
y:              -0.00109
@PRED.y/@y:            .    @ERROR.y/@y:           .
```

Note: Check for missing input data or uninitialized lags.

(Note that the LAG and DIF functions return missing values for the initial lag starting observations. This is a change from the 1982 and earlier versions of SAS/ETS which returned zero for uninitialized lags.)

Note: Simulation aborted.

At the first observation, a solution to the following equation is attempted:

$$y = -\sqrt{y} - 62$$

There is no solution to this problem. The iterative solution process got as close as it could to making Y negative while still being able to evaluate the model. This problem can be avoided in this case by altering the equation.

In other models, the problem of missing values can be avoided by either altering the data set to provide better starting values for the solution variables or by altering the equations.

You should be aware that, in general, a nonlinear system can have any number of solutions and the solution found might not be the one that you want. When multiple solutions exist, the solution that is found is usually determined by the starting values for the iterations. If the value from the input data set for a solution variable is missing, the starting value for it is taken from the solution of the last period (if nonmissing) or else the solution estimate is started at 0.

Iteration Output

The iteration output, produced by the ITPRINT option, is useful in determining the cause of a convergence problem. The ITPRINT option forces the printing of the solution approximation and equation errors at each iteration for each observation. A portion of the ITPRINT output from the following statements is shown in Figure 19.86.

```
proc model data=test1;
   exogenous x1 ;
   control a1 -1 b1 -29 c1 -4 ;
   y = a1 * sqrt(abs(y)) + b1 * x1 * x1 + c1 * lag(x1);
   solve y / out=sim forecast dynamic itprint;
run;
```

For each iteration, the equation with the largest error is listed in parentheses after the Newton convergence criteria measure. From this output you can determine which equation or equations in the system are not converging well.

Figure 19.86 SOLVE, ITPRINT Output

The MODEL Procedure
Dynamic Single-Equation Forecast

Observation 1 **Iteration** 0 **CC** 613961.39 **ERROR.y** -62.01010

Predicted Values
y
0.0001000

Iteration Errors
y
-62.01010

Observation 1 **Iteration** 1 **CC** 50.902771 **ERROR.y** -61.88684

Predicted Values
y
-1.215784

Iteration Errors
y
-61.88684

Observation 1 **Iteration** 2 **CC** 0.364806 **ERROR.y** 41.752112

Figure 19.86 *continued*

Predicted Values
y
-114.4503

Iteration Errors
y
41.75211

Numerical Integration

The differential equation system is numerically integrated to obtain a solution for the derivative variables at each data point. The integration is performed by evaluating the provided model at multiple points between each data point. The integration method used is a variable order, variable step-size backward difference scheme; for more detailed information, see Aiken (1985); Byrne and Hindmarsh (1975). The step size or time step is chosen to satisfy a *local truncation error* requirement. The term *truncation error* comes from the fact that the integration scheme uses a truncated series expansion of the integrated function to do the integration. Because the series is truncated, the integration scheme is within the truncation error of the true value.

To further improve the accuracy of the integration, the total integration time is broken up into small intervals (time steps or step sizes), and the integration scheme is applied to those intervals. The integration at each time step uses the values computed at the previous time step so that the truncation error tends to accumulate. It is usually not possible to estimate the global error with much precision. The best that can be done is to monitor and to control the local truncation error, which is the truncation error committed at each time step relative to

$$d = \max_{0 \le t \le T} (\|y(t)\|_\infty, 1)$$

where $y(t)$ is the integrated variable. Furthermore, the $y(t)$s are dynamically scaled to within two orders of magnitude of one to keep the error monitoring well-behaved.

The local truncation error requirement defaults to 1.0E–9. You can specify the LTEBOUND= option to modify that requirement. The LTEBOUND= option is a relative measure of accuracy, so a value smaller than 1.0E–10 is usually not practical. A larger bound increases the speed of the simulation and estimation but decreases the accuracy of the results. If the LTEBOUND= option is set too small, the integrator is not able to take time steps small enough to satisfy the local truncation error requirement and still have enough machine precision to compute the results. Since the integrations are scaled to within 1.0E–2 of one, the simulated values should be correct to at least seven decimal places.

There is a default minimum time step of 1.0E–14. This minimum time step is controlled by the MINTIMESTEP= option and the machine epsilon. If the minimum time step is smaller than the machine epsilon times the final time value, the minimum time step is increased automatically.

For the points between each observation in the data set, the values for nonintegrated variables in the data set are obtained from a linear interpolation from the two closest points. Lagged variables can be used with integrations, but their values are discrete and are not interpolated between points. Lagging, therefore, can then be used to input step functions into the integration.

The derivatives necessary for estimation (the gradient with respect to the parameters) and goal seeking (the Jacobian) are computed by numerically integrating analytical derivatives. The accuracy of the derivatives is controlled by the same integration techniques mentioned previously.

Limitations

There are limitations to the types of differential equations that can be solved or estimated. One type is an explosive differential equation (finite escape velocity) for which the following differential equation is an example:

$$y' = a \times y, a > 0$$

If this differential equation is integrated too far in time, y exceeds the maximum value allowed on the computer, and the integration terminates.

Likewise, differential systems that are singular cannot be solved or estimated in general. For example, consider the following differential system:

$$\begin{aligned} x' &= -y' + 2x + 4y + \exp(t) \\ y' &= -x' + y + \exp(4*t) \end{aligned}$$

This system has an analytical solution, but an accurate numerical solution is very difficult to obtain. The reason is that y' and x' cannot be isolated on the left-hand side of the equation. If the equation is modified slightly to

$$\begin{aligned} x' &= -y' + 2x + 4y + \exp(t) \\ y' &= x' + y + \exp(4t) \end{aligned}$$

the system is nonsingular, but the integration process could still fail or be extremely slow. If the MODEL procedure encounters either system, a warning message is issued.

This system can be rewritten as the following recursive system, which can be estimated and simulated successfully with the MODEL procedure:

$$\begin{aligned} x' &= 0.5y + 0.5\exp(4t) + x + 1.5y - 0.5\exp(t) \\ y' &= x' + y + \exp(4t) \end{aligned}$$

Petzold (1982) mentions a class of differential algebraic equations that, when integrated numerically, could produce incorrect or misleading results. An example of such a system is

$$\begin{aligned} y_2'(t) &= y_1(t) + g_1(t) \\ 0 &= y_2(t) + g_2(t) \end{aligned}$$

The analytical solution to this system depends on g and its derivatives at the current time only and not on its initial value or past history. You should avoid systems of this and other similar forms mentioned in Petzold (1982).

SOLVE Data Sets

SDATA= Input Data Set

The SDATA= option reads a cross-equation covariance matrix from a data set. The covariance matrix read from the SDATA= data set specified in the SOLVE statement is used to generate random equation errors when the RANDOM= option specifies Monte Carlo simulation.

Typically, the SDATA= data set is created by the OUTS= option in a previous FIT statement. (The OUTS= data set from a FIT statement can be read back in by a SOLVE statement in the same PROC MODEL step.)

You can create an input SDATA= data set by using the DATA step. PROC MODEL expects to find a character variable _NAME_ in the SDATA= data set as well as variables for the equations in the estimation or solution. For each observation with a _NAME_ value that matches the name of an equation, PROC MODEL fills the corresponding row of the **S** matrix with the values of the names of equations found in the data set. If a row or column is omitted from the data set, an identity matrix row or column is assumed. Missing values are ignored. Since the **S** matrix is symmetric, you can include only a triangular part of the **S** matrix in the SDATA= data set with the omitted part indicated by missing values. If the SDATA= data set contains multiple observations with the same _NAME_, the last values supplied for the _NAME_ variable are used. The section "OUTS= Data Set" on page 1228 contains more details on the format of this data set.

Use the TYPE= option to specify the type of estimation method used to produce the **S** matrix you want to input.

ESTDATA= Input Data Set

The ESTDATA= option specifies an input data set that contains an observation with values for some or all of the model parameters. It can also contain observations with the rows of a covariance matrix for the parameters.

When the ESTDATA= option is used, parameter values are set from the first observation. If the RANDOM= option is used and the ESTDATA= data set contains a covariance matrix, the covariance matrix of the parameter estimates is read and used to generate pseudo-random shocks to the model parameters for Monte Carlo simulation. These random perturbations have a multivariate normal distribution with the covariance matrix read from the ESTDATA= data set.

The ESTDATA= data set is usually created by the OUTEST= option in a FIT statement. The OUTEST= data set contains the parameter estimates produced by the FIT statement and also contains the estimated covariance of the parameter estimates if the OUTCOV option is used. This OUTEST= data set can be read in by the ESTDATA= option in a SOLVE statement.

You can also create an ESTDATA= data set with a SAS DATA step program. The data set must contain a numeric variable for each parameter to be given a value or covariance column. The name of the variable in the ESTDATA= data set must match the name of the parameter in the model. Parameters with names longer than 32 characters cannot be set from an ESTDATA= data set. The data set must also contain a character variable _NAME_ of length 32. _NAME_ has a blank value for the observation that gives values to the parameters. _NAME_ contains the name of a parameter for observations that define rows of the covariance matrix.

More than one set of parameter estimates and covariances can be stored in the ESTDATA= data set if the observations for the different estimates are identified by the variable _TYPE_. _TYPE_ must be a character

variable of length eight. The TYPE= option is used to select for input the part of the ESTDATA= data set for which the value of the _TYPE_ variable matches the value of the TYPE= option.

OUT= Data Set

The OUT= data set contains solution values, residual values, and actual values of the solution variables.

The OUT= data set contains the following variables:

- BY variables
- RANGE variable
- ID variables
- _TYPE_, a character variable of length eight that identifies the type of observation. The _TYPE_ variable can be PREDICT, RESIDUAL, ACTUAL, or ERROR.
- _MODE_, a character variable of length eight that identifies the solution mode. _MODE_ takes the value FORECAST or SIMULATE.
- if lags are used, a numeric variable, _LAG_, that contains the number of dynamic lags that contribute to the solution. The value of _LAG_ is always zero for STATIC mode solutions. _LAG_ is set to a missing value for lag-starting observations.
- if the RANDOM= option is used, _REP_, a numeric variable that contains the replication number. For example, if RANDOM=10, each input observation results in eleven output observations with _REP_ values 0 through 10. The observations with _REP_=0 are from the unperturbed solution. (The random-number generator functions are suppressed, and the parameter and endogenous perturbations are zero when _REP_=0.)
- _ERRORS_, a numeric variable that contains the number of errors that occurred during the execution of the program for the last iteration for the observation. If the solution failed to converge, this is counted as one error, and the _ERRORS_ variable is made negative.
- solution and other variables. The solution variables contain solution or predicted values for _TYPE_=PREDICT observations, residuals for _TYPE_=RESIDUAL observations, or actual values for _TYPE_=ACTUAL observations. The other model variables, and any other variables read from the input data set, are always actual values from the input data set.
- any other variables named in the OUTVARS statement. These can be program variables computed by the model program, CONTROL variables, parameters, or special variables in the model program. Compound variable names longer than 32 characters are truncated in the OUT= data set.

By default, only the predicted values are written to the OUT= data set. The OUTRESID, OUTACTUAL, and OUTERROR options are used to add the residual, actual, and ERROR. values, respectively, to the data set.

For examples of the OUT= data set, see Example 19.6.

DATA= Input Data Set

The input data set should contain all of the exogenous variables and should supply nonmissing values for them for each period to be solved.

Solution variables can be supplied in the input data set and are used as follows:

- to supply initial lags. For example, if the lag length of the model is three, three observations are read in to feed the lags before any solutions are computed.
- to evaluate the goodness of fit. Goodness-of-fit measures are computed based on the difference between the solved values and the actual values supplied from the data set.
- to supply starting values for the iterative solution. If the value from the input data set for a solution variable is missing, the starting value for it is taken from the solution of the last period (if nonmissing) or else the solution estimate is started at zero.
- for STATIC mode solutions, actual values from the data set are used by the lagging functions for the solution variables.
- for FORECAST mode solutions, actual values from the data set are used as the solution values when nonmissing.

Programming Language Overview: MODEL Procedure

Variables in the Model Program

Variable names are alphanumeric but must start with a letter. The length is limited to 32 characters.

PROC MODEL uses several classes of variables, and different variable classes are treated differently. The variable class is controlled by *declaration statements*: the VAR, ENDOGENOUS, and EXOGENOUS statements for model variables, the PARAMETERS statement for parameters, and the CONTROL statement for control class variables. These declaration statements have several valid abbreviations. Various *internal variables* are also made available to the model program to allow communication between the model program and the procedure. RANGE, ID, and BY variables are also available to the model program. Those variables not declared as any of the preceding classes are *program variables*.

Some classes of variables can be lagged; that is, their value at each observation is remembered, and previous values can be referred to by the lagging functions. Other classes have only a single value and are not affected by lagging functions. For example, parameters have only one value and are not affected by lagging functions; therefore, if P is a parameter, DIF*n* (P) is always 0, and LAG*n* (P) is always the same as P for all values of *n*.

The different variable classes and their roles in the model are described in the following.

Model Variables

Model variables are declared by VAR, ENDOGENOUS, or EXOGENOUS statements, or by FIT and SOLVE statements. The model variables are the variables that the model is intended to explain or predict.

PROC MODEL enables you to use expressions on the left-hand side of the equal sign to define model equations. For example, a log-linear model for Y can be written as

```
log( y ) = a + b * x;
```

Previously, only a variable name was allowed on the left-hand side of the equal sign.

The text on the left-hand side of the equation serves as the equation name used to identify the equation in printed output, in the OUT= data sets, and in FIT or SOLVE statements. To refer to equations specified by using left-hand side expressions (in the FIT statement, for example), place the left-hand side expression in quotes. For example, the following statements fit a log-linear model to the dependent variable Y:

```
proc model data=in;
   log( y ) = a + b * x;
   fit "log(y)";
run;
```

The estimation and simulation is performed by transforming the models into general form equations. No actual or predicted value is available for general form equations, so no R^2 or adjusted R^2 is computed.

Equation Variables

An equation variable is one of several special variables used by PROC MODEL to control the evaluation of model equations. An equation variable name consists of one of the prefixes EQ, RESID, ERROR, PRED, or ACTUAL, followed by a period and the name of a model equation.

Equation variable names can appear in parts of the PROC MODEL printed output, and they can be used in the model program. For example, RESID-prefixed variables can be used in LAG functions to define equations with moving-average error terms. See the section "Autoregressive Moving-Average Error Processes" on page 1205 for details.

The meaning of these prefixes is detailed in the section "Equation Translations" on page 1272.

Parameters

Parameters are variables that have the same value for each observation. Parameters can be given values or can be estimated by fitting the model to data. During the SOLVE stage, parameters are treated as constants. If no estimation is performed, the SOLVE stage uses the initial value provided in the ESTDATA= data set, the MODEL= file, or in the PARAMETER statement, as the value of the parameter.

The PARAMETERS statement declares the parameters of the model. Parameters are not lagged, and they cannot be changed by the model program.

Control Variables

Control variables supply constant values to the model program that can be used to control the model in various ways. The CONTROL statement declares control variables and specifies their values. A control variable is like a parameter except that it has a fixed value and is not estimated from the data.

Control variables are not reinitialized before each pass through the data and can thus be used to retain values between passes. You can use control variables to vary the program logic. Control variables are not affected by lagging functions.

For example, if you have two versions of an equation for a variable Y, you could put both versions in the model and, by using a CONTROL statement to select one of them, produce two different solutions to explore the effect the choice of equation has on the model, as shown in the following statements:

```
select (case);
   when (1) y =  ...first version of equation... ;
   when (2) y =  ...second version of equation... ;
end;

control case 1;
solve / out=case1;
run;

control case 2;
solve / out=case2;
run;
```

RANGE, ID, and BY Variables

The RANGE statement controls the range of observations in the input data set that is processed by PROC MODEL. The ID statement lists variables in the input data set that are used to identify observations in the printout and in the output data set. The BY statement can be used to make PROC MODEL perform a separate analysis for each BY group. The variable in the RANGE statement, the ID variables, and the BY variables are available for the model program to examine, but their values should not be changed by the program. The BY variables are not affected by lagging functions.

Internal Variables

You can use several internal variables in the model program to communicate with the procedure. For example, if you want PROC MODEL to list the values of all the variables when more than 10 iterations are performed and the procedure is past the 20th observation, you can write

```
if _obs_ > 20 then if _iter_ > 10 then _list_ = 1;
```

Internal variables are not affected by lagging functions, and they cannot be changed by the model program except as noted. The following internal variables are available. The variables are all numeric except where noted.

ERRORS is a flag that is set to 0 at the start of program execution and is set to a nonzero value whenever an error occurs. The program can also set the _ERRORS_ variable.

ITER is the iteration number. For FIT tasks, the value of _ITER_ is negative for preliminary grid-search passes. The iterative phase of the estimation starts with iteration 0. After the estimates have converged, a final pass is made to collect statistics with _ITER_ set to a missing value. Note that at least one pass, and perhaps several subiteration passes as well, is made for each iteration. For SOLVE tasks, _ITER_ counts the iterations used to compute the simultaneous solution of the system.

LAG is the number of dynamic lags that contribute to the solution at the current observation. _LAG_ is always 0 for FIT tasks and for STATIC solutions. _LAG_ is set to a missing value during the lag starting phase.

LIST is a list flag that is set to 0 at the start of program execution. The program can set _LIST_ to a nonzero value to request a listing of the values of all the variables in the program after the program has finished executing.

METHOD is the solution method in use for SOLVE tasks. _METHOD_ is set to a blank value for FIT tasks. _METHOD_ is a character-valued variable. Values are NEWTON, JACOBI, SIEDEL, or ONEPASS.

MODE takes the value ESTIMATE for FIT tasks and the value SIMULATE or FORECAST for SOLVE tasks. _MODE_ is a character-valued variable.

NMISS is the number of missing or otherwise unusable observations during the model estimation. For FIT tasks, _NMISS_ is initially set to 0; at the start of each iteration, _NMISS_ is set to the number of unusable observations for the previous iteration. For SOLVE tasks, _NMISS_ is set to a missing value.

NUSED is the number of nonmissing observations used in the estimation. For FIT tasks, PROC MODEL initially sets _NUSED_ to the number of parameters; at the start of each iteration, _NUSED_ is reset to the number of observations used in the previous iteration. For SOLVE tasks, _NUSED_ is set to a missing value.

OBS counts the observations being processed. _OBS_ is negative or 0 for observations in the lag starting phase.

REP is the replication number for Monte Carlo simulation when the RANDOM= option is specified in the SOLVE statement. _REP_ is 0 when the RANDOM= option is not used and for FIT tasks. When _REP_=0, the random-number generator functions always return 0.

WEIGHT is the weight of the observation. For FIT tasks, _WEIGHT_ provides a weight for the observation in the estimation. _WEIGHT_ is initialized to 1.0 at the start of execution for FIT tasks. For SOLVE tasks, _WEIGHT_ is ignored.

Program Variables

Variables not in any of the other classes are called program variables. Program variables are used to hold intermediate results of calculations. Program variables are reinitialized to missing values before each observation is processed. Program variables can be lagged. The RETAIN statement can be used to give program variables initial values and enable them to keep their values between observations.

Character Variables

PROC MODEL supports both numeric and character variables. Character variables are not involved in the model specification but can be used to label observations, to write debugging messages, or for documentation purposes. All variables are numeric unless they are the following.

- character variables in a DATA= SAS data set
- program variables assigned a character value
- declared to be character by a LENGTH or ATTRIB statement

Equation Translations

Equations written in normalized form are always automatically converted to general form equations. For example, when a normalized form equation such as

```
y = a + b*x;
```

is encountered, it is translated into the equations

```
PRED.y = a + b*x;
RESID.y = PRED.y - ACTUAL.y;
ERROR.y = PRED.y - y;
```

If the same system is expressed as the following general form equation, then this equation is used unchanged.

```
EQ.y = y - (a + b*x);
```

This makes it easy to solve for arbitrary variables and to modify the error terms for autoregressive or moving average models.

Use the LIST option to see how this transformation is performed. For example, the following statements produce the listing shown in Figure 19.87.

```
proc model data=line list;
   y = a1 + b1*x1 + c1*x2;
   fit y;
run;
```

Figure 19.87 LIST Output

The MODEL Procedure

Listing of Compiled Program Code		
Stmt	**Line:Col**	**Statement as Parsed**
1	4268:4	PRED.y = a1 + b1 * x1 + c1 * x2;
1	4268:4	RESID.y = PRED.y - ACTUAL.y;
1	4268:4	ERROR.y = PRED.y - y;

PRED.Y is the predicted value of Y, and ACTUAL.Y is the value of Y in the data set. The predicted value minus the actual value, RESID.Y, is then the error term, ϵ, for the original Y equation. Note that the residuals obtained from the OUTRESID option in the OUT=dataset for both the FIT and SOLVE statements are defined as $actual - predicted$, the negative of RESID.Y. See the section "Syntax: MODEL Procedure" on page 1084 for details. ACTUAL.Y and Y have the same value for parameter estimation. For solve tasks, ACTUAL.Y is still the value of Y in the data set but Y becomes the solved value; the value that satisfies PRED.Y – Y = 0.

The following are the equation variable definitions.

EQ. The value of an EQ.-prefixed equation variable (normally used to define a general form equation) represents the failure of the equation to hold. When the EQ.*name* variable is 0, the *name* equation is satisfied.

RESID. The RESID.*name* variables represent the stochastic parts of the equations and are used to define the objective function for the estimation process. A RESID.-prefixed equation variable is like an EQ.-prefixed variable but makes it possible to use or transform the stochastic part of the equation. The RESID. equation is used in place of the ERROR. equation for model solutions if it has been reassigned or used in the equation.

ERROR. An ERROR.*name* variable is like an EQ.-prefixed variable, except that it is used only for model solution and does not affect parameter estimation.

PRED. For a normalized form equation (specified by assignment to a model variable), the PRED.*name* equation variable holds the predicted value, where *name* is the name of both the model variable and the corresponding equation. (PRED.-prefixed variables are not created for general form equations.)

ACTUAL. For a normalized form equation (specified by assignment to a model variable), the ACTUAL.*name* equation variable holds the value of the *name* model variable read from the input data set.

DERT. The DERT.*name* variable defines a differential equation. Once defined, it might be used on the right-hand side of another equation.

H. The H.*name* variable specifies the functional form for the variance of the named equation.

GMM_H. This is created for H.*vars* and is the moment equation for the variance for GMM. This variable is used only for GMM.

```
GMM_H.name = RESID.name**2 - H.name;
```

MSE. The MSE.*y* variable contains the value of the mean squared error for *y* at each iteration. An MSE. variable is created for each dependent/endogenous variable in the model. These

variables can be used to specify the missing lagged values in the estimation and simulation of GARCH type models.

```
demret = intercept ;
h.demret = arch0 +
           arch1 * xlag( resid.demret ** 2, mse.demret) +
           garch1 * xlag(h.demret, mse.demret) ;
```

NRESID. This is created for H.*vars* and is the normalized residual of the variable <*name* >. The formula is

```
NRESID.name = RESID.name/ sqrt(H.name);
```

The three equation variable prefixes, RESID., ERROR., and EQ. allow for control over the objective function for the FIT, the SOLVE, or both the FIT and the SOLVE stages. For FIT tasks, PROC MODEL looks first for a RESID.*name* variable for each equation. If defined, the RESID.-prefixed equation variable is used to define the objective function for the parameter estimation process. Otherwise, PROC MODEL looks for an EQ.-prefixed variable for the equation and uses it instead.

For SOLVE tasks, PROC MODEL looks first for an ERROR.*name* variable for each equation. If defined, the ERROR.-prefixed equation variable is used for the solution process. Otherwise, PROC MODEL looks for an EQ.-prefixed variable for the equation and uses it instead. To solve the simultaneous equation system, PROC MODEL computes values of the solution variables (the model variables being solved for) that make all of the ERROR.name and EQ.*name* variables close to 0.

Derivatives

Nonlinear modeling techniques require the calculation of derivatives of certain variables with respect to other variables. The MODEL procedure includes an analytic differentiator that determines the model derivatives and generates program code to compute these derivatives. When parameters are estimated, the MODEL procedure takes the derivatives of the equation with respect to the parameters. When the model is solved, Newton's method requires the derivatives of the equations with respect to the variables solved for.

PROC MODEL uses exact mathematical formulas for derivatives of non-user-defined functions. For other functions, numerical derivatives are computed and used.

The differentiator differentiates the entire model program, including the conditional logic and flow of control statements. Delayed definitions, as when the LAG of a program variable is referred to before the variable is assigned a value, are also differentiated correctly.

The differentiator includes optimization features that produce efficient code for the calculation of derivatives. However, when flow of control statements such as GOTO statements are used, the optimization process is impeded, and less efficient code for derivatives might be produced. Optimization is also reduced by conditional statements, iterative DO loops, and multiple assignments to the same variable.

The table of derivatives is printed with the LISTDER option. The code generated for the computation of the derivatives is printed with the LISTCODE option.

Derivative Variables

When the differentiator needs to generate code to evaluate the expression for the derivative of a variable, the result is stored in a special derivative variable. Derivative variables are not created when the derivative expression reduces to a previously computed result, a variable, or a constant. The names of derivative variables, which might sometimes appear in the printed output, have the form @*obj* /@*wrt*, where *obj* is the variable whose derivative is being taken and *wrt* is the variable that the differentiation is with respect to. For example, the derivative variable for the derivative of *Y* with respect to *X* is named @*Y*/@*X*.

The derivative variables can be accessed or used as part of the model program using the GETDER() function.

GETDER(*x*, *a*) the derivative of *x* with respect to *a*.

GETDER(*x*, *a*, *b*) the second derivative of *x* with respect to *a* and *b*.

The main purpose of the GETDER() function is for surfacing the derivatives so they can be stored in a data set for further processing. Only derivatives that are implied by the problem are available to the GETDER() function. When derivatives are requested that aren't already created, a missing value will be returned. The derivative of the GETDER() function is always zero so the results of the GETDER() function shouldn't be used in any of the equations in the FIT or the SOLVE statement.

The following example adds the gradient of the PRED.*y* value with respect to the parameters to the OUT= data set.

```
proc model data=line ;
   y = a1 + b1**2 *x1 + c1*x2;
   Dy_a1 = getder(PRED.y,a1);
   Dy_b1 = getder(PRED.y,b1);
   Dy_c1 = getder(PRED.y,c1);
   outvars Dy_a1 Dy_b1 Dy_c1;
   fit y / out=grad;
run;
```

Mathematical Functions

The following is a brief summary of SAS functions that are useful for defining models. Additional functions and details are in *SAS Language: Reference*. Information about creating new functions can be found in *SAS/BASE Software: Procedure Reference*, Chapter 18, "The FCMP Procedure."

ABS(*x*) the absolute value of *x*

ARCOS(*x*) the arccosine in radians of *x*; *x* should be between −1 and 1.

ARSIN(*x*) the arcsine in radians of *x*; *x* should be between −1 and 1.

ATAN(*x*) the arctangent in radians of *x*

COS(*x*) the cosine of *x*; *x* is in radians.

COSH(*x*) the hyperbolic cosine of *x*

EXP(*x*) e^x

LOG(*x*) the natural logarithm of *x*

LOG10(x) the log base ten of x

LOG2(x) the log base two of x

SIN(x) the sine of x; x is in radians.

SINH(x) the hyperbolic sine of x

SQRT(x) the square root of x

TAN(x) the tangent of x; x is in radians and is not an odd multiple of $\pi/2$.

TANH(x) the hyperbolic tangent of x

Random-Number Functions

The MODEL procedure provides several functions for generating random numbers for Monte Carlo simulation. These functions use the same generators as the corresponding SAS DATA step functions.

The following random number functions are supported: RANBIN, RANCAU, RAND, RANEXP, RANGAM, RANNOR, RANPOI, RANTBL, RANTRI, and RANUNI. For more information, refer to *SAS Language: Reference*.

Each reference to a random number function sets up a separate pseudo-random sequence. Note that this means that two calls to the same random function with the same seed produce identical results. This is different from the behavior of the random number functions used in the SAS DATA step. For example, the following statements produce identical values for X and Y, but Z is from an independent pseudo-random sequence:

```
x=rannor(123);
y=rannor(123);
z=rannor(567);
q=rand('BETA', 1, 12 );
```

For FIT tasks, all random number functions always return 0. For SOLVE tasks, when Monte Carlo simulation is requested, a random number function computes a new random number on the first iteration for an observation (if it is executed on that iteration) and returns that same value for all later iterations of that observation. When Monte Carlo simulation is not requested, random number functions always return 0.

Functions across Time

PROC MODEL provides four types of special built-in functions that refer to the values of variables and expressions in previous time periods. These functions have the following forms where n represents the number of periods, x is any expression, and the argument i is a variable or expression that gives the lag length ($0 <= i <= n$). If the index value i is omitted, the maximum lag length n is used.

LAGn ($< i, > x$) returns the ith lag of x, where n is the maximum lag;

DIFn (x) is the difference of x at lag n

ZLAGn ($< i, > x$) returns the ith lag of x, where n is the maximum lag, with missing lags replaced with zero

XLAG*n* (*x*, *y*) returns the *n*th lag of *x* if *x* is nonmissing, or *y* if x is missing

ZDIF*n* (*x*) is the difference with lag length truncated and missing values converted to zero; *x* is the variable or expression to compute the moving average of

MOVAVG*n*(*x*) is the moving average if X_t denotes the observation at time point *t*, to ensure compatibility with the number *n* of observations used to calculate the moving average MOVAVGn, the following definition is used:

$$MOVAVGn(X_t) = \frac{X_t + X_{t-1} + X_{t-2} + \ldots + X_{t-n+1}}{n}$$

The moving average calculation for SAS 9.1 and earlier releases is as follows:

$$MOVAVGn(X_t) = \frac{X_t + X_{t-1} + X_{t-2} + \ldots + X_{t-n}}{n+1}$$

Missing values of *x* are omitted in computing the average.

If you do not specify *n*, the number of periods is assumed to be one. For example, LAG(X) is the same as LAG1(X). No more than four digits can be used with a lagging function; that is, LAG9999 is the greatest LAG function, ZDIF9999 is the greatest ZDIF function, and so on.

The LAG functions get values from previous observations and make them available to the program. For example, LAG(X) returns the value of the variable X as it was computed in the execution of the program for the preceding observation. The expression LAG2(X+2*Y) returns the value of the expression X+2*Y, computed by using the values of the variables X and Y that were computed by the execution of the program for the observation two periods ago.

The DIF functions return the difference between the current value of a variable or expression and the value of its LAG. For example, DIF2(X) is a short way of writing X–LAG2(X), and DIF15(SQRT(2*Z)) is a short way of writing SQRT(2*Z)–LAG15(SQRT(2*Z)).

The ZLAG and ZDIF functions are like the LAG and DIF functions, but they are not counted in the determination of the program lag length, and they replace missing values with 0s. The ZLAG function returns the lagged value if the lagged value is nonmissing, or 0 if the lagged value is missing. The ZDIF function returns the differenced value if the differenced value is nonmissing, or 0 if the value of the differenced value is missing. The ZLAG function is especially useful for models with ARMA error processes. See the next section for details.

Lag Logic

The LAG and DIF lagging functions in the MODEL procedure are different from the queuing functions with the same names in the DATA step. Lags are determined by the final values that are set for the program variables by the execution of the model program for the observation. This can have upsetting consequences for programs that take lags of program variables that are given different values at various places in the program, as shown in the following statements:

```
temp = x + w;
t    = lag( temp );
temp = q - r;
s    = lag( temp );
```

The expression LAG(TEMP) always refers to LAG(Q–R), never to LAG(X+W), since Q–R is the final value assigned to the variable TEMP by the model program. If LAG(X+W) is wanted for T, it should be computed as T=LAG(X+W) and not T=LAG(TEMP), as in the preceding example.

Care should also be exercised in using the DIF functions with program variables that might be reassigned later in the program. For example, the program

```
temp =  x ;
s    = dif( temp );
temp = 3 * y;
```

computes values for S equivalent to

```
s =  x  - lag( 3 * y );
```

Note that in the preceding examples, TEMP is a program variable, *not* a model variable. If it were a model variable, the assignments to it would be changed to assignments to a corresponding equation variable.

Note that whereas LAG1(LAG1(X)) is the same as LAG2(X), DIF1(DIF1(X)) is *not* the same as DIF2(X). The DIF2 function is the difference between the current period value at the point in the program where the function is executed and the final value at the end of execution two periods ago; DIF2 is not the second difference. In contrast, DIF1(DIF1(X)) is equal to DIF1(X)-LAG1(DIF1(X)), which equals X–2*LAG1(X)+LAG2(X), which is the second difference of X.

More information about the differences between PROC MODEL and the DATA step LAG and DIF functions is found in Chapter 3, "Working with Time Series Data."

Lag Lengths

The lag length of the model program is the number of lags needed for any relevant equation. The program lag length controls the number of observations used to initialize the lags.

PROC MODEL keeps track of the use of lags in the model program and automatically determines the lag length of each equation and of the model as a whole. PROC MODEL sets the program lag length to the maximum number of lags needed to compute any equation to be estimated, solved, or needed to compute any instrument variable used.

In determining the lag length, the ZLAG and ZDIF functions are treated as always having a lag length of 0. For example, if Y is computed as

```
y = lag2( x + zdif3( temp ) );
```

then Y has a lag length of 2 (regardless of how TEMP is defined). If Y is computed as

```
y = zlag2( x + dif3( temp ) );
```

then Y has a lag length of 0.

This is so that ARMA errors can be specified without causing the loss of additional observations to the lag starting phase and so that recursive lag specifications, such as moving-average error terms, can be used. Recursive lags are not permitted unless the ZLAG or ZDIF functions are used to truncate the lag length. For example, the following statement produces an error message:

```
t = a + b * lag( t );
```

The program variable T depends recursively on its own lag, and the lag length of T is therefore undefined.

In the following equation RESID.Y depends on the predicted value for the Y equation but the predicted value for the Y equation depends on the LAG of RESID.Y, and thus, the predicted value for the Y equation depends recursively on its own lag.

```
y = yhat + ma * lag( resid.y );
```

The lag length is infinite, and PROC MODEL prints an error message and stops. Since this kind of specification is allowed, the recursion must be truncated at some point. The ZLAG and ZDIF functions do this.

The following equation is valid and results in a lag length for the Y equation equal to the lag length of YHAT:

```
y = yhat + ma * zlag( resid.y );
```

Initially, the lags of RESID.Y are missing, and the ZLAG function replaces the missing residuals with 0s, their unconditional expected values.

The ZLAG0 function can be used to zero out the lag length of an expression. ZLAG0(x) returns the current period value of the expression x, if nonmissing, or else returns 0, and prevents the lag length of x from contributing to the lag length of the current statement.

Initializing Lags

At the start of each pass through the data set or BY group, the lag variables are set to missing values and an initialization is performed to fill the lags. During this phase, observations are read from the data set, and the model variables are given values from the data. If necessary, the model is executed to assign values to program variables that are used in lagging functions. The results for variables used in lag functions are saved. These observations are not included in the estimation or solution.

If, during the execution of the program for the lag starting phase, a lag function refers to lags that are missing, the lag function returns missing. Execution errors that occur while starting the lags are not reported unless requested. The modeling system automatically determines whether the program needs to be executed during the lag starting phase.

If L is the maximum lag length of any equation being fit or solved, then the first L observations are used to prime the lags. If a BY statement is used, the first L observations in the BY group are used to prime the lags. If a RANGE statement is used, the first L observations prior to the first observation requested in the RANGE statement are used to prime the lags. Therefore, there should be at least L observations in the data set.

Initial values for the lags of model variables can also be supplied in VAR, ENDOGENOUS, and EXOGENOUS statements. This feature provides initial lags of solution variables for dynamic solution when initial values for the solution variable are not available in the input data set. For example, the statement

```
var x 2 3 y 4 5 z 1;
```

feeds the initial lags exactly like these values in an input data set:

Lag	X	Y	Z
2	3	5	.
1	2	4	1

If initial values for lags are available in the input data set and initial lag values are also given in a declaration statement, the values in the VAR, ENDOGENOUS, or EXOGENOUS statements take priority.

The RANGE statement is used to control the range of observations in the input data set that are processed by PROC MODEL. In the following statement, '01jan1924' specifies the starting period of the range, and '01dec1943' specifies the ending period:

```
range date = '01jan1924'd to '01dec1943'd;
```

The observations in the data set immediately prior to the start of the range are used to initialize the lags.

Language Differences

For the most part, PROC MODEL programming statements work the same as they do in the DATA step as documented in *SAS Language: Reference*. However, there are several differences that should be noted.

DO Statement Differences

The DO statement in PROC MODEL does not allow a character index variable. Thus, the following DO statement is not valid in PROC MODEL, although it is supported in the DATA step:

```
do i = 'A', 'B', 'C';  /* invalid PROC MODEL code */
```

IF Statement Differences

The IF statement in PROC MODEL does not allow a character-valued condition. For example, the following IF statement is not supported by PROC MODEL:

```
if 'this' then  statement;
```

Comparisons of character values are supported in IF statements, so the following IF statement is acceptable:

```
if 'this' < 'that' then  statement;
```

PROC MODEL allows for embedded conditionals in expressions. For example the following two statements are equivalent:

```
flag = if time = 1 or time = 2 then conc+30/5 + dose*time
       else if time > 5 then (0=1) else (patient * flag);
```

```
if time = 1 or time = 2 then flag= conc+30/5 + dose*time;
else if time > 5 then flag=(0=1); else flag=patient*flag;
```

Note that the ELSE operator involves only the first object or token after it so that the following assignments are not equivalent:

```
total = if sum > 0 then sum else sum + reserve;
total = if sum > 0 then sum else (sum + reserve);
```

The first assignment makes TOTAL always equal to SUM plus RESERVE.

PUT Statement Differences

The PUT statement, mostly used in PROC MODEL for program debugging, supports only some of the features of the DATA step PUT statement. It also has some new features that the DATA step PUT statement does not support.

The PROC MODEL PUT statement does not support line pointers, factored lists, iteration factors, overprinting, the _INFILE_ option, or the colon (:) format modifier.

The PROC MODEL PUT statement does support expressions, but an expression must be enclosed in parentheses. For example, the following statement prints the square root of x:

```
put (sqrt(x));
```

Subscripted array names must be enclosed in parentheses. For example, the following statement prints the *i*th element of the array A:

```
put (a i);
```

However, the following statement is an error:

```
put a i;
```

The PROC MODEL PUT statement supports the print item _PDV_ to print a formatted listing of all the variables in the program. For example, the following statement prints a much more readable listing of the variables than does the _ALL_ print item:

```
put _pdv_;
```

To print all the elements of the array A, use the following statement:

```
put a;
```

To print all the elements of A with each value labeled by the name of the element variable, use the following statement:

```
put a=;
```

ABORT Statement Difference

In the MODEL procedure, the ABORT statement does not allow any arguments.

SELECT/WHEN/OTHERWISE Statement Differences

The WHEN and OTHERWISE statements allow more than one target statement. That is, DO groups are not necessary for multiple statement WHENs. For example in PROC MODEL, the following syntax is valid:

```
select;
   when(exp1)
      stmt1;
      stmt2;
   when(exp2)
      stmt3;
      stmt4;
end;
```

The ARRAY Statement

ARRAY *arrayname* < *{dimensions}* > < *$ [length]* > < *variables and constants* > *;* **;**

The ARRAY statement is used to associate a name with a list of variables and constants. The array name can then be used with subscripts in the model program to refer to the items in the list.

In PROC MODEL, the ARRAY statement does not support all the features of the DATA step ARRAY statement. Implicit indexing cannot be used; all array references must have explicit subscript expressions. Only exact array dimensions are allowed; lower-bound specifications are not supported. A maximum of six dimensions is allowed.

On the other hand, the ARRAY statement supported by PROC MODEL does allow both variables and constants to be used as array elements. You cannot make assignments to constant array elements. Both dimension specification and the list of elements are optional, but at least one must be supplied. When the list of elements is not given or fewer elements than the size of the array are listed, array variables are created by suffixing element numbers to the array name to complete the element list.

The following are valid PROC MODEL array statements:

```
array x[120];           /* array X of length 120            */
array q[2,2];           /* Two dimensional array Q          */
array b[4] va vb vc vd; /* B[2] = VB, B[4] = VD             */
array x x1-x30;         /* array X of length 30, X[7] = X7  */
array a[5] (1 2 3 4 5); /* array A initialized to 1,2,3,4,5 */
```

RETAIN Statement

RETAIN *variables initial-values* **;**

The RETAIN statement causes a program variable to hold its value from a previous observation until the variable is reassigned. The RETAIN statement can be used to initialize program variables.

The RETAIN statement does not work for model variables, parameters, or control variables because the values of these variables are under the control of PROC MODEL and not programming statements. Use the PARMS and CONTROL statements to initialize parameters and control variables. Use the VAR, ENDOGENOUS, or EXOGENOUS statement to initialize model variables.

Storing Programs in Model Files

Models can be saved in and recalled from SAS catalog files as well as XML-based data sets. SAS catalogs are special files that can store many kinds of data structures as separate units in one SAS file. Each separate unit is called an entry, and each entry has an entry type that identifies its structure to the SAS system. Starting with SAS 9.2, model files are being stored as SAS data sets instead of being stored as members of a SAS catalog as in earlier releases. This makes MODEL files more readily extendable in the future and enables Java-based applications to read the MODEL files directly. You can choose between the two formats by specifying a global CMPMODEL option in an OPTIONS statement. Details are given below.

In general, to save a model, use the OUTMODEL=*name* option in the PROC MODEL statement, where *name* is specified as *libref.catalog.entry*, *libref.entry*, or *entry* for catalog entry and, starting with SAS 9.2, *libref.datasetname* or *datasetname* for XML-based SAS data sets. The *libref*, *catalog*, *datasetnames* and *entry* names must be valid SAS names no more than 32 characters long. The *catalog* name is restricted to seven characters on the CMS operating system. If not given, the *catalog* name defaults to MODELS, and the *libref* defaults to WORK. The entry type is always MODEL. Thus, OUTMODEL=X writes the model to the file WORK.MODELS.X.MODEL in the SAS catalog or creates a WORK.X XML-based dataset in the WORK library depending on the format chosen by using the CMPMODEL= option. By default, both these formats are chosen.

The CMPMODEL= option can be used in an OPTIONS statement to modify the behavior when reading and writing MODEL files. The values allowed are CMPMODEL= BOTH | XML | CATALOG. For example, the following statements restore the previous behavior:

```
options cmpmodel=catalog;
```

The CMPMODEL= option defaults to BOTH in SAS 9.2 and is intended for transitional use. If CMPMODEL=BOTH, the MODEL procedure writes both formats; when loading model files PROC MODEL attempts to load the XML version first and the CATALOG version second (if the XML version is not found). If CMPMODEL=XML, the MODEL procedure reads and writes only the XML format. If CMPMODEL=CATALOG, only the catalog format is used.

The MODEL= option is used to read in a model. A list of model files can be specified in the MODEL= option, and a range of names with numeric suffixes can be given, as in MODEL=(MODEL1–MODEL10). When more than one model file is given, the list must be placed in parentheses, as in MODEL=(A B C), except in case of a single name. If more than one model file is specified, the files are combined in the order listed in the MODEL= option.

The MODEL procedure continues to read and write catalog MODEL files, and model files created by previous releases of SAS/ETS continue to work, so you should experience no direct impact from this change.

When the MODEL= option is specified in the PROC MODEL statement and model definition statements are also given later in the PROC MODEL step, the model files are read in first, in the order listed, and the model program specified in the PROC MODEL step is appended after the model program read from the MODEL= files. The class that is assigned to a variable, when multiple model files are used, is the last declaration of that variable. For example, if Y1 is declared endogenous in the model file M1 and exogenous in the model file M2, the following statement causes Y1 to be declared exogenous:

```
proc model model=(m1 m2);
```

The INCLUDE statement can be used to append model code to the current model code. In contrast, when the MODEL= option is specified in the RESET statement, the current model is deleted before the new model is read.

By default, no model file is output if the PROC MODEL step performs any FIT or SOLVE tasks, or if the MODEL= option or the NOSTORE option is specified. However, to ensure compatibility with previous versions of SAS/ETS software, if the PROC MODEL step does nothing but compile the model program, no input model file is read, and the NOSTORE option is not used, then a model file is written. This model file is the default input file for a later PROC SYSLIN or PROC SIMLIN step. The default output model filename in this case is WORK.MODELS._MODEL_.MODEL.

If FIT statements are used to estimate model parameters, the parameter estimates that are written to the output model file are the estimates from the last estimation performed for each parameter.

Macro Return Codes (SYSINFO)

The MODEL procedure stores a return code in the automatic macro variable SYSINFO upon completion of the PROC MODEL step. In the event any FIT or SOLVE task fails to converge during the completion of a PROC MODEL step, the value 1 is stored in the SYSINFO macro variable. Any subsequent SAS step resets the value of SYSINFO.

Diagnostics and Debugging

PROC MODEL provides several features to aid in finding errors in the model program. These debugging features are not usually needed; most models can be developed without them.

The example model program that follows is used in the following sections to illustrate the diagnostic and debugging capabilities. This example is the estimation of a segmented model.

```
/*--- Diagnostics and Debugging ---*/

*---------Fitting a Segmented Model using MODEL----*
|      |                                           |
|   y  | quadratic          plateau                |
|      | y=a+b*x+c*x*x      y=p                    |
|      |                     ..................... |
|      |              .     :                      |
|      |          .         :                      |
|      |       .            :                      |
|      |     .              :                      |
|      |    .               :                      |
|      +-------------------------------------X     |
|                          x0                       |
|                                                   |
| continuity restriction: p=a+b*x0+c*x0**2          |
| smoothness restriction: 0=b+2*c*x0 so x0=-b/(2*c)|
*---------------------------------------------------*;
title 'QUADRATIC MODEL WITH PLATEAU';
data a;
   input y x @@;
datalines;
.46 1  .47  2 .57  3 .61  4 .62  5 .68  6 .69  7
.78 8  .70  9 .74 10 .77 11 .78 12 .74 13 .80 13
.80 15 .78 16
;

proc model data=a list xref listcode;
   parms a 0.45 b 0.5 c -0.0025;

   x0 = -.5*b / c;      /* join point */
   if x < x0 then       /* Quadratic part of model */
      y = a + b*x + c*x*x;
   else                 /* Plateau part of model */
      y = a + b*x0 + c*x0*x0;

   fit y;
run;
```

Program Listing

The LIST option produces a listing of the model program. The statements are printed one per line with the original line number and column position of the statement.

The program listing from the example program is shown in Figure 19.88.

Figure 19.88 LIST Output for Segmented Model

QUADRATIC MODEL WITH PLATEAU

The MODEL Procedure

Listing of Compiled Program Code

Stmt	Line:Col	Statement as Parsed
1	4314:4	x0 = (-0.5 * b) / c;
2	4315:4	if x < x0 then
3	4316:7	PRED.y = a + b * x + c * x * x;
3	4316:7	RESID.y = PRED.y - ACTUAL.y;
3	4316:7	ERROR.y = PRED.y - y;
4	4317:4	else
5	4318:7	PRED.y = a + b * x0 + c * x0 * x0;
5	4318:7	RESID.y = PRED.y - ACTUAL.y;
5	4318:7	ERROR.y = PRED.y - y;

The LIST option also shows the model translations that PROC MODEL performs. LIST output is useful for understanding the code generated by the %AR and the %MA macros.

Cross-Reference

The XREF option produces a cross-reference listing of the variables in the model program. The XREF listing is usually used in conjunction with the LIST option. The XREF listing does not include derivative (@-prefixed) variables. The XREF listing does not include generated assignments to equation variables, PRED., RESID., and ERROR.-prefixed variables, unless the DETAILS option is used.

The cross-reference from the example program is shown in Figure 19.89.

Figure 19.89 XREF Output for Segmented Model

QUADRATIC MODEL WITH PLATEAU

The MODEL Procedure

Cross Reference Listing For Program

Symbol----------	Kind	Type	References (statement)/(line):(col)
a	Var	Num	Used: 3/62740:13 5/62742:13
b	Var	Num	Used: 1/62738:12 3/62740:16 5/62742:16
c	Var	Num	Used: 1/62738:15 3/62740:22 5/62742:23
x0	Var	Num	Assigned: 1/62738:15
			Used: 2/62739:11 5/62742:16 5/62742:23 5/62742:26
x	Var	Num	Used: 2/62739:11 3/62740:16 3/62740:22 3/62740:24
PRED.y	Var	Num	Assigned: 3/62740:19 5/62742:20

Compiler Listing

The LISTCODE option lists the model code and derivatives tables produced by the compiler. This listing is useful only for debugging and should not normally be needed.

LISTCODE prints the operator and operands of each operation generated by the compiler for each model program statement. Many of the operands are temporary variables generated by the compiler and given names such as #temp1. When derivatives are taken, the code listing includes the operations generated for the derivatives calculations. The derivatives tables are also listed.

A LISTCODE option prints the transformed equations from the example shown in Figure 19.90 and Figure 19.91.

Figure 19.90 LISTCODE Output for Segmented Model—Statements as Parsed

Derivatives		
WRT-Variable	**Object-Variable**	**Derivative-Variable**
a	RESID.y	@RESID.y/@a
b	RESID.y	@RESID.y/@b
c	RESID.y	@RESID.y/@c

Listing of Compiled Program Code		
Stmt	**Line:Col**	**Statement as Parsed**
1	4314:4	x0 = (-0.5 * b) / c;
1	4314:4	@x0/@b = -0.5 / c;
1	4314:4	@x0/@c = - x0 / c;
2	4315:4	if x < x0 then
3	4316:7	PRED.y = a + b * x + c * x * x;
3	4316:7	@PRED.y/@a = 1;
3	4316:7	@PRED.y/@b = x;
3	4316:7	@PRED.y/@c = x * x;
3	4316:7	RESID.y = PRED.y - ACTUAL.y;
3	4316:7	@RESID.y/@a = @PRED.y/@a;
3	4316:7	@RESID.y/@b = @PRED.y/@b;
3	4316:7	@RESID.y/@c = @PRED.y/@c;
3	4316:7	ERROR.y = PRED.y - y;
4	4317:4	else
5	4318:7	PRED.y = a + b * x0 + c * x0 * x0;
5	4318:7	@PRED.y/@a = 1;
5	4318:7	@PRED.y/@b = x0 + b * @x0/@b + (c * @x0/@b * x0 + c * x0 * @x0/@b);
5	4318:7	@PRED.y/@c = b * @x0/@c + ((x0 + c * @x0/@c) * x0 + c * x0 * @x0/@c);
5	4318:7	RESID.y = PRED.y - ACTUAL.y;
5	4318:7	@RESID.y/@a = @PRED.y/@a;
5	4318:7	@RESID.y/@b = @PRED.y/@b;
5	4318:7	@RESID.y/@c = @PRED.y/@c;
5	4318:7	ERROR.y = PRED.y - y;

Figure 19.91 LISTCODE Output for Segmented Model—Compiled Code

```
1 Stmt ASSIGN  line 4314 column 4. (1) arg=x0 argsave=x0
               Source Text:                                    x0 = -.5*b / c;
Oper *         at 4314:12 (30,0,2).                            * : _temp1 <- -0.5 b
Oper /         at 4314:15 (31,0,2).                            / : x0 <- _temp1 c
Oper eeocf     at 4314:15 (18,0,1).                            eeocf : _DER_ <- _DER_
Oper /         at 4314:15 (31,0,2).                            / : @x0/@b <- -0.5 c
Oper -         at 4314:15 (24,0,1).                            - : @1dt1_2 <- x0
Oper /         at 4314:15 (31,0,2).                            / : @x0/@c <- @1dt1_2 c

2 Stmt IF      line 4315 column 4. (2) arg=_temp1 argsave=_temp1 ref.st=ASSIGN stmt number 5 at 4318:7
               Source Text:                                    if x < x0 then
Oper <         at 4315:11 (36,0,2).                            < : _temp1 <- x x0

3 Stmt ASSIGN  line 4316 column 7. (1) arg=PRED.y argsave=y
               Source Text:                                    /* Quadratic part of model */ y = a + b*x + c*x*x;
Oper *         at 4316:16 (30,0,2).                            * : _temp1 <- b x
Oper +         at 4316:13 (32,0,2).                            + : _temp2 <- a _temp1
Oper *         at 4316:22 (30,0,2).                            * : _temp3 <- c x
Oper *         at 4316:24 (30,0,2).                            * : _temp4 <- _temp3 x
Oper +         at 4316:19 (32,0,2).                            + : PRED.y <- _temp2 _temp4
Oper eeocf     at 4316:19 (18,0,1).                            eeocf : _DER_ <- _DER_
Oper =         at 4316:19 (1,0,1).                             = : @PRED.y/@a <- 1
Oper =         at 4316:19 (1,0,1).                             = : @PRED.y/@b <- x
Oper *         at 4316:24 (30,0,2).                            * : @1dt1_1 <- x x
Oper =         at 4316:19 (1,0,1).                             = : @PRED.y/@c <- @1dt1_1

3 Stmt Assign  line 4316 column 7. (1) arg=RESID.y argsave=y
Oper -         at 4316:7 (33,0,2).                             - : RESID.y <- PRED.y ACTUAL.y
Oper eeocf     at 4316:7 (18,0,1).                             eeocf : _DER_ <- _DER_
Oper =         at 4316:7 (1,0,1).                              = : @RESID.y/@a <- @PRED.y/@a
Oper =         at 4316:7 (1,0,1).                              = : @RESID.y/@b <- @PRED.y/@b
Oper =         at 4316:7 (1,0,1).                              = : @RESID.y/@c <- @PRED.y/@c

3 Stmt Assign  line 4316 column 7. (1) arg=ERROR.y argsave=y
Oper -         at 4316:7 (33,0,2).                             - : ERROR.y <- PRED.y y

4 Stmt ELSE    line 4317 column 4. (9)                         ref.st=FIT stmt number 5 at 4320:4
               Source Text:                                    else

5 Stmt ASSIGN  line 4318 column 7. (1) arg=PRED.y argsave=y
               Source Text:                                    /* Plateau part of model */ y = a + b*x0 + c*x0*x0;
Oper *         at 4318:16 (30,0,2).                            * : _temp1 <- b x0
Oper +         at 4318:13 (32,0,2).                            + : _temp2 <- a _temp1
Oper *         at 4318:23 (30,0,2).                            * : _temp3 <- c x0
Oper *         at 4318:26 (30,0,2).                            * : _temp4 <- _temp3 x0
Oper +         at 4318:20 (32,0,2).                            + : PRED.y <- _temp2 _temp4
Oper eeocf     at 4318:20 (18,0,1).                            eeocf : _DER_ <- _DER_
Oper =         at 4318:20 (1,0,1).                             = : @PRED.y/@a <- 1
```

Figure 19.91 *continued*

Oper *	at 4318:16 (30,0,2).	* : @1dt1_1 <- b @x0/@b
Oper +	at 4318:16 (32,0,2).	+ : @1dt1_2 <- x0 @1dt1_1
Oper *	at 4318:23 (30,0,2).	* : @1dt1_3 <- c @x0/@b
Oper *	at 4318:26 (30,0,2).	* : @1dt1_4 <- @1dt1_3 x0
Oper *	at 4318:26 (30,0,2).	* : @1dt1_5 <- _temp3 @x0/@b
Oper +	at 4318:26 (32,0,2).	+ : @1dt1_6 <- @1dt1_4 @1dt1_5
Oper +	at 4318:20 (32,0,2).	+ : @PRED.y/@b <- @1dt1_2 @1dt1_6
Oper *	at 4318:16 (30,0,2).	* : @1dt1_8 <- b @x0/@c
Oper *	at 4318:23 (30,0,2).	* : @1dt1_9 <- c @x0/@c
Oper +	at 4318:23 (32,0,2).	+ : @1dt1_10 <- x0 @1dt1_9
Oper *	at 4318:26 (30,0,2).	* : @1dt1_11 <- @1dt1_10 x0
Oper *	at 4318:26 (30,0,2).	* : @1dt1_12 <- _temp3 @x0/@c
Oper +	at 4318:26 (32,0,2).	+ : @1dt1_13 <- @1dt1_11 @1dt1_12
Oper +	at 4318:20 (32,0,2).	+ : @PRED.y/@c <- @1dt1_8 @1dt1_13
5 Stmt Assign	line 4318 column 7. (1) arg=RESID.y argsave=y	
Oper -	at 4318:7 (33,0,2).	- : RESID.y <- PRED.y ACTUAL.y
Oper eeocf	at 4318:7 (18,0,1).	eeocf : _DER_ <- _DER_
Oper =	at 4318:7 (1,0,1).	= : @RESID.y/@a <- @PRED.y/@a
Oper =	at 4318:7 (1,0,1).	= : @RESID.y/@b <- @PRED.y/@b
Oper =	at 4318:7 (1,0,1).	= : @RESID.y/@c <- @PRED.y/@c
5 Stmt Assign	line 4318 column 7. (1) arg=ERROR.y argsave=y	
Oper -	at 4318:7 (33,0,2).	- : ERROR.y <- PRED.y y

Analyzing the Structure of Large Models

PROC MODEL provides several features to aid in analyzing the structure of the model program. These features summarize properties of the model in various forms.

Simulation Dependency Analysis

During the development of model programs for simulation, misspecification of the equations or variables that compose the systems of nonlinear equations is common. These misspecification errors can occur both in the original formulation of the model and in the encoding of the model into PROC MODEL statements. For large systems these errors can be difficult and time consuming to isolate and repair. Similarly, the process of becoming familiar with an existing simulation model that is encoded in PROC MODEL can be laborious when available documentation is insufficient to understand the model's implementation. To address these issues, the ANALYZEDEP= option can be applied to SOLVE steps to produce graphical analyses of a model's structure.

The graphical output that is produced by the ANALYZEDEP= option displays the results of two separate, hierarchical analyses that are both based on the dependence of equations on solve variables in the nonlinear system of equations. First, the system is partitioned to identify which equations overdetermine solve variables, which equations underdetermine solve variables, and which equations consistently determine solve variables. These three partitions of equations and their corresponding three partitions of solve variables are identified in the graphical output and listing produced by the ANALYZEDEP= option. Second, each partition from the first analysis is analyzed to identify subpartitions of equations and solve variables such that all the solve variables within each subpartition depend either directly or indirectly on one another. In the graphical output the subpartitions are represented as blocks in a dependency matrix. The subpartition blocks are ordered so that the matrix of dependencies has a block upper-triangular form.

The first-level partitioning of the system into underdetermined, overdetermined, and consistent systems of equations and variables uses a Dulmage-Mendelsohn (DM) decomposition to define the three partitions, following the work by Dulmage and Mendelsohn (1958); Pothen and Fan (1990). The overdetermining equations in a DM decomposition are the set of all equations that do not have dependent variables on the diagonal of any dependency matrix that contains the maximum possible number of entries on the diagonal. The dependency matrices for a problem consist of the set of pairs of orderings of the problem's equations and solve variables. Correspondingly, the DM decomposition defines underdetermined variables as the set of all variables that do not appear on the diagonal of any dependency matrix that contains the maximum number of entries on the diagonal. Therefore, the DM decomposition is canonical in the sense that its partitioning of the system is invariant to the order equations and variables are specified in the model program. The following PROC MODEL statements illustrate how to partition a simple model with five equations and five unknowns:

```
proc model data=_null_;
   endo a b c d e;

   f(a)     = 0;
   g(a,b)   = 0;
   h(a,b)   = 0;
   i(b,d)   = 0;
   j(c,d,e) = 0;

   solve / analyzedep=(block);
quit;
```

Figure 19.92 Block Dependency Analysis

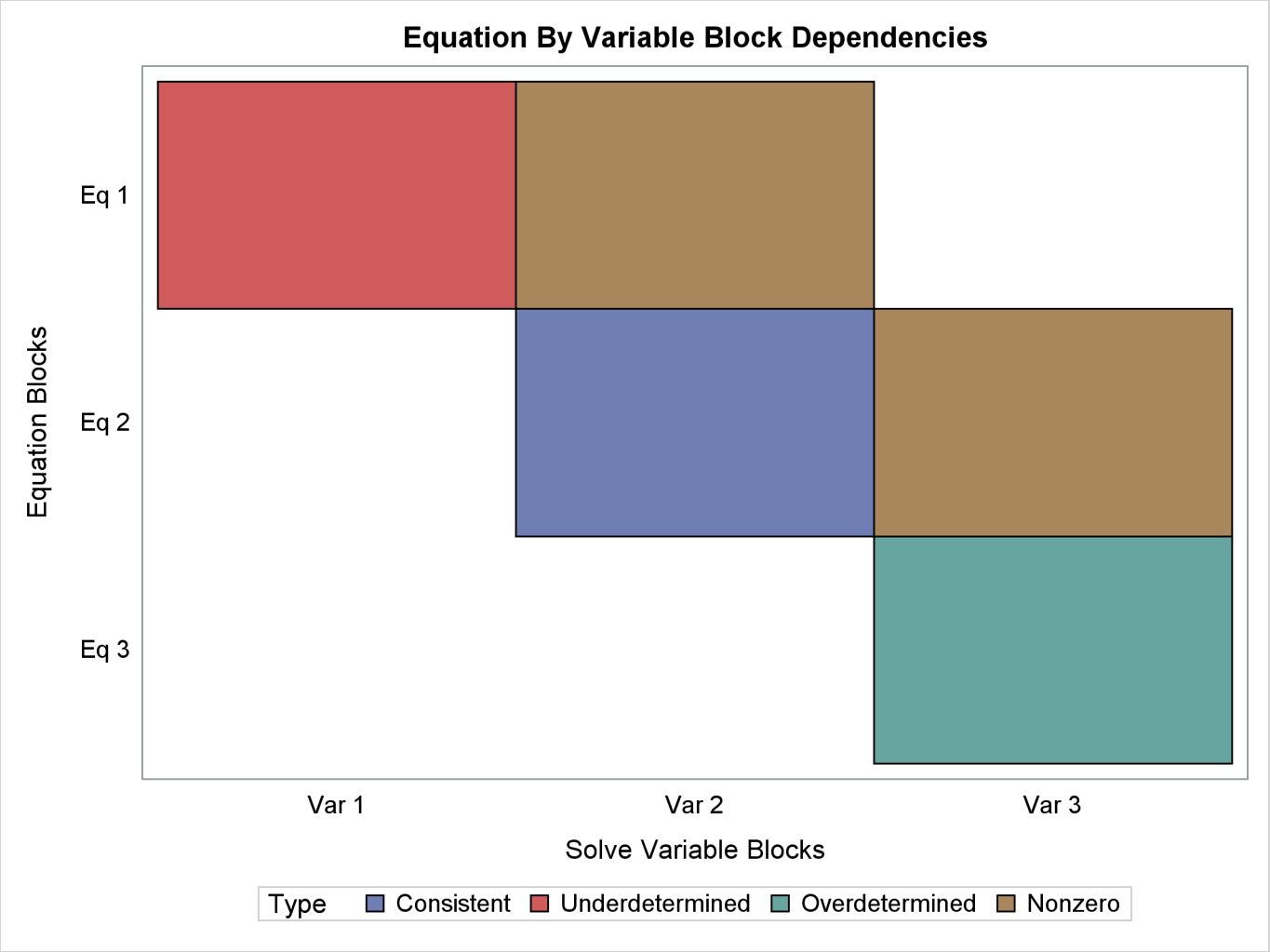


Figure 19.93 Block Partitions

Equation and Variable Blocks		
Type	**Block**	**Symbols**
Underdetermined	Eq 1	j(c,d,e)
	Var 1	c e
Consistent	Eq 2	i(b,d)
	Var 2	d
Overdetermined	Eq 3	f(a) g(a,b) h(a,b)
	Var 3	a b

Figure 19.92 and Figure 19.93 illustrate which equations and variables belong to each block and which blocks are in each partition. The cells that are marked "Nonzero" in the plot represent a dependency between blocks that are above the diagonal in the dependency matrix. The exact functional forms of the equations in this example are not shown; however, the dependency analysis here reveals that this model is structurally

singular because it contains overdetermined and underdetermined components. Some modification of the model specification is necessary before a SOLVE step can be executed.

For large systems of equations, the graphical output that the ANALYZEDEP= option produces can be used as a starting point to explore dependency relationships when the models' programming statement listings and dependency tables are too long to read and comprehend. For example, one econometric model of U.S. agriculture involves thousand of equation and variable dependencies whose structure is difficult to interpret in textual listings of the model. If you examine the block triangular form of its dependency matrix in Figure 19.94, one pattern of dependencies that becomes apparent is the vertical grouping of block dependencies in the middle of the plot. Figure 19.95 shows the dependency matrix for this important subpartition of equations and variables responsible for coupling the vertical grouping of blocks.

Figure 19.94 Block Triangular Form of U.S. Agriculture Model

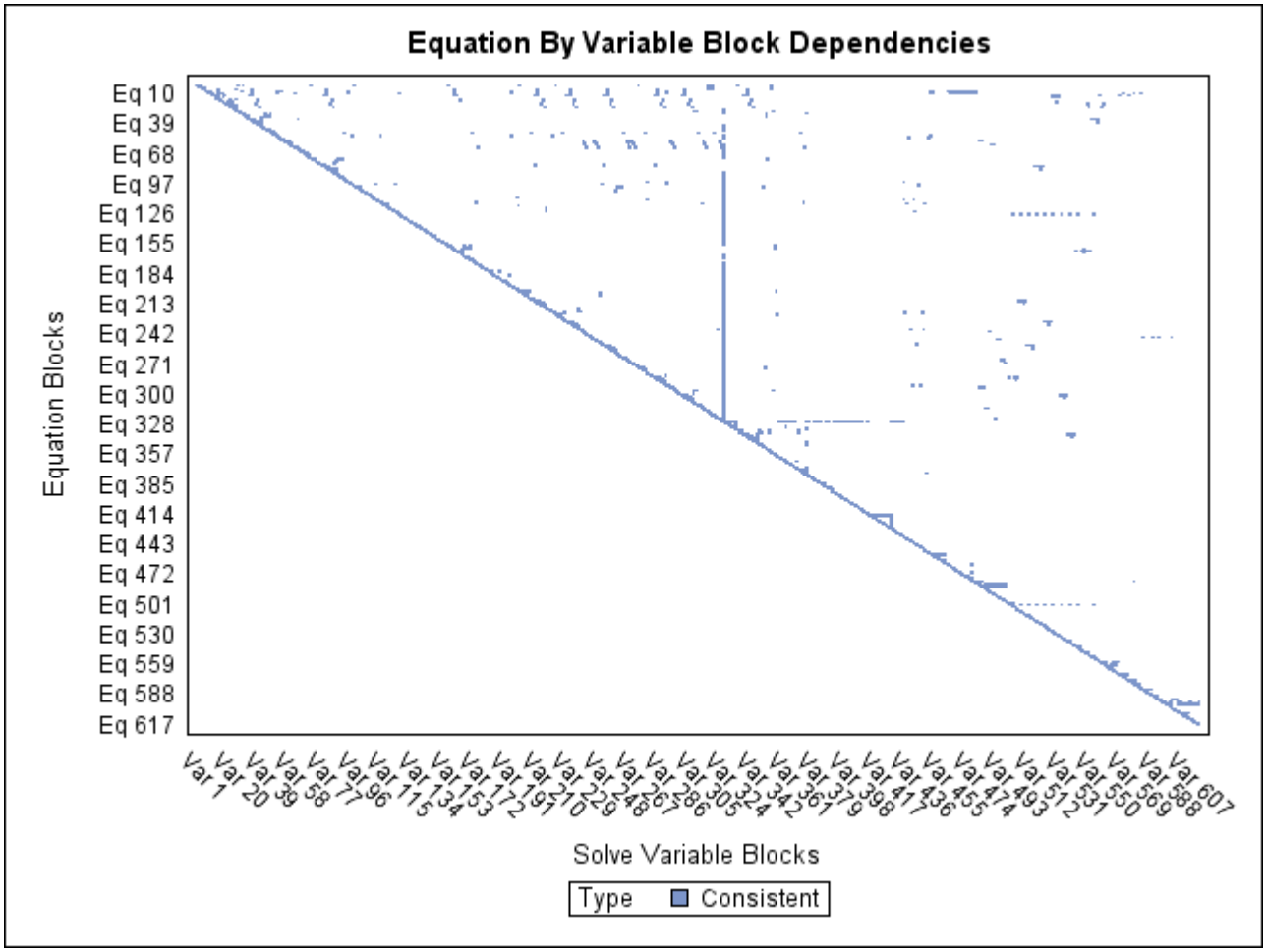
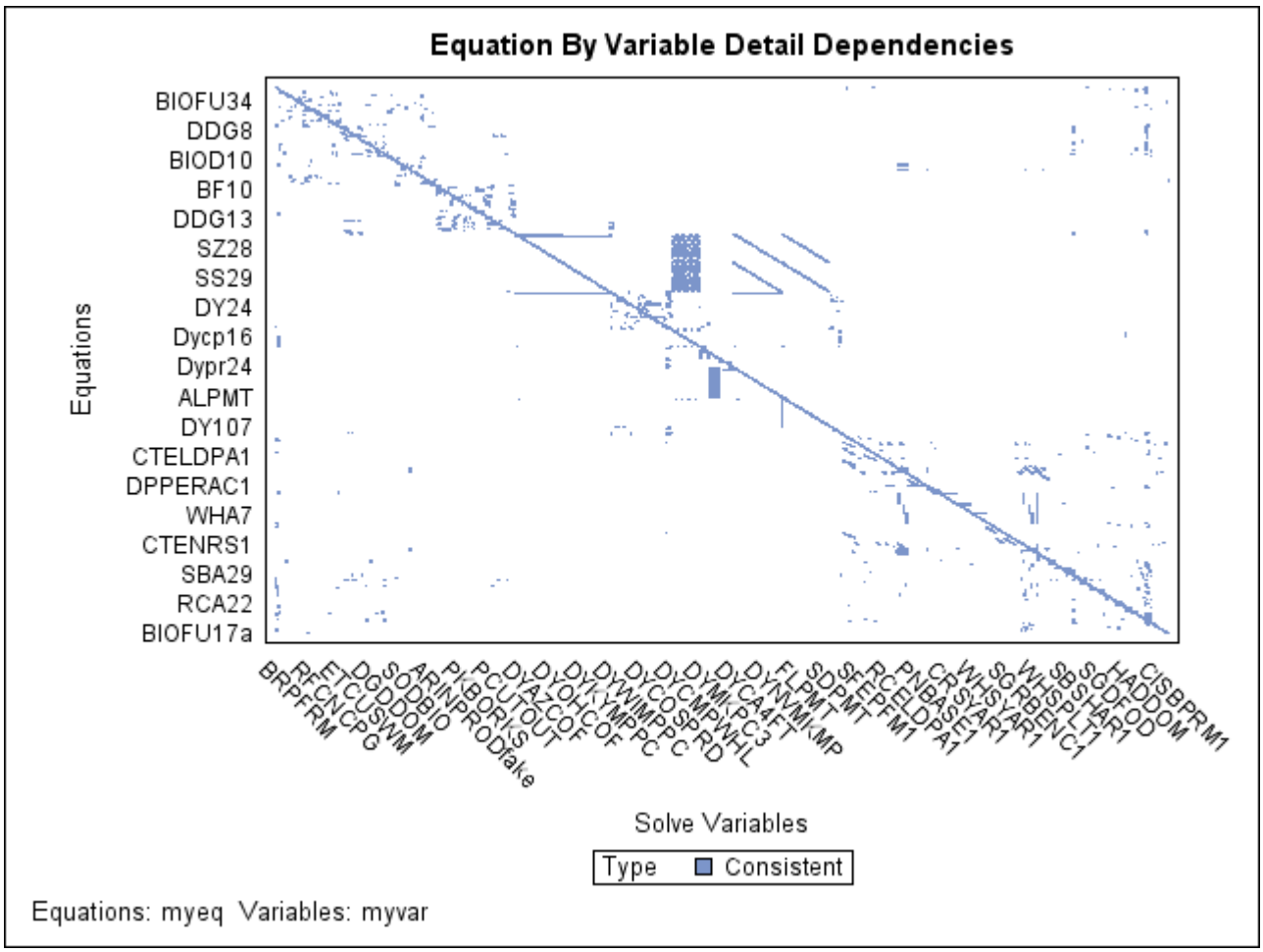


Figure 19.95 Important Component in U.S. Agriculture Model



Compared to the BLOCK and GRAPH options, the ANALYZEDEP= option has the following advantages:

- shows which equations and solve variables are overdetermined, consistent, and underdetermined
- works with any combination of normal form and general form equations
- can display dependency matrices involving many more equations and variables
- can be limited to a subset of the equations and variables in the model

The following Klein's model program is used to introduce the LISTDEP, BLOCK, and GRAPH options:

```
proc model  out=m data=klein listdep graph block;
   endogenous c p w i x wsum k y;
   exogenous  wp g t year;
   parms c0-c3 i0-i3 w0-w3;
   a: c = c0 + c1 * p + c2 * lag(p) + c3 * wsum;
   b: i = i0 + i1 * p + i2 * lag(p) + i3 * lag(k);
   c: w = w0 + w1 * x + w2 * lag(x) + w3 * year;
   x = c + i + g;
   y = c + i + g-t;
   p = x-w-t;
   k = lag(k) + i;
   wsum = w + wp;
   id year;
quit;
```

Dependency List

The LISTDEP option produces a dependency list for each variable in the model program. For each variable, a list of variables that depend on it and a list of variables it depends on is given. The dependency list produced by the example program is shown in Figure 19.96.

Figure 19.96 A Portion of the LISTDEP Output for Klein's Model

The MODEL Procedure

Dependency Listing For Program	
Symbol----------	**Dependencies**
c	Current values affect: RESID.c ERROR.c PRED.x RESID.x ERROR.x PRED.y RESID.y ERROR.y
p	Current values affect: PRED.c RESID.c ERROR.c PRED.i RESID.i ERROR.i RESID.p ERROR.p
	Lagged values affect: PRED.c PRED.i
w	Current values affect: RESID.w ERROR.w PRED.p RESID.p ERROR.p PRED.wsum RESID.wsum ERROR.wsum
i	Current values affect: RESID.i ERROR.i PRED.x RESID.x ERROR.x PRED.y RESID.y ERROR.y PRED.k RESID.k ERROR.k
x	Current values affect: PRED.w RESID.w ERROR.w RESID.x ERROR.x PRED.p RESID.p ERROR.p
	Lagged values affect: PRED.w
wsum	Current values affect: PRED.c RESID.c ERROR.c RESID.wsum ERROR.wsum
k	Current values affect: RESID.k ERROR.k
	Lagged values affect: PRED.i RESID.i ERROR.i PRED.k
y	Current values affect: RESID.y ERROR.y
wp	Current values affect: PRED.wsum RESID.wsum ERROR.wsum
g	Current values affect: PRED.x RESID.x ERROR.x PRED.y RESID.y ERROR.y
t	Current values affect: PRED.y RESID.y ERROR.y PRED.p RESID.p ERROR.p
year	Current values affect: PRED.w RESID.w ERROR.w
c0	Current values affect: PRED.c RESID.c ERROR.c
c1	Current values affect: PRED.c RESID.c ERROR.c
c2	Current values affect: PRED.c RESID.c ERROR.c
c3	Current values affect: PRED.c RESID.c ERROR.c
i0	Current values affect: PRED.i RESID.i ERROR.i
i1	Current values affect: PRED.i RESID.i ERROR.i
i2	Current values affect: PRED.i RESID.i ERROR.i
i3	Current values affect: PRED.i RESID.i ERROR.i
w0	Current values affect: PRED.w RESID.w ERROR.w
w1	Current values affect: PRED.w RESID.w ERROR.w
w2	Current values affect: PRED.w RESID.w ERROR.w
w3	Current values affect: PRED.w RESID.w ERROR.w
PRED.c	Depends on current values of: p wsum c0 c1 c2 c3
	Depends on lagged values of: p
	Current values affect: RESID.c ERROR.c
RESID.c	Depends on current values of: PRED.c c p wsum c0 c1 c2 c3
ERROR.c	Depends on current values of: PRED.c c p wsum c0 c1 c2 c3
ACTUAL.c	Current values affect: RESID.c ERROR.c PRED.x RESID.x ERROR.x PRED.y RESID.y ERROR.y
PRED.i	Depends on current values of: p i0 i1 i2 i3
	Depends on lagged values of: p k
	Current values affect: RESID.i ERROR.i
RESID.i	Depends on current values of: PRED.i p i i0 i1 i2 i3
	Depends on lagged values of: k
ERROR.i	Depends on current values of: PRED.i p i i0 i1 i2 i3
	Depends on lagged values of: k
ACTUAL.i	Current values affect: RESID.i ERROR.i PRED.x RESID.x ERROR.x PRED.y RESID.y ERROR.y PRED.k RESID.k ERROR.k
PRED.w	Depends on current values of: x year w0 w1 w2 w3
	Depends on lagged values of: x
	Current values affect: RESID.w ERROR.w

Figure 19.96 *continued*

The MODEL Procedure

Dependency Listing For Program	
Symbol----------	**Dependencies**
RESID.w	Depends on current values of: PRED.w w x year w0 w1 w2 w3
ERROR.w	Depends on current values of: PRED.w w x year w0 w1 w2 w3
ACTUAL.w	Current values affect: RESID.w ERROR.w PRED.p RESID.p ERROR.p PRED.wsum RESID.wsum ERROR.wsum
PRED.x	Depends on current values of: c i g
	Current values affect: RESID.x ERROR.x
RESID.x	Depends on current values of: PRED.x c i x g
ERROR.x	Depends on current values of: PRED.x c i x g
ACTUAL.x	Current values affect: PRED.w RESID.w ERROR.w RESID.x ERROR.x PRED.p RESID.p ERROR.p
	Lagged values affect: PRED.w
PRED.y	Depends on current values of: c i g t
	Current values affect: RESID.y ERROR.y
RESID.y	Depends on current values of: PRED.y c i y g t
ERROR.y	Depends on current values of: PRED.y c i y g t
ACTUAL.y	Current values affect: RESID.y ERROR.y
PRED.p	Depends on current values of: w x t
	Current values affect: RESID.p ERROR.p
RESID.p	Depends on current values of: PRED.p p w x t
ERROR.p	Depends on current values of: PRED.p p w x t
ACTUAL.p	Current values affect: PRED.c RESID.c ERROR.c PRED.i RESID.i ERROR.i RESID.p ERROR.p
	Lagged values affect: PRED.c PRED.i
PRED.k	Depends on current values of: i
	Depends on lagged values of: k
	Current values affect: RESID.k ERROR.k
RESID.k	Depends on current values of: PRED.k i k
ERROR.k	Depends on current values of: PRED.k i k
ACTUAL.k	Current values affect: RESID.k ERROR.k
	Lagged values affect: PRED.i RESID.i ERROR.i PRED.k
PRED.wsum	Depends on current values of: w wp
	Current values affect: RESID.wsum ERROR.wsum
RESID.wsum	Depends on current values of: PRED.wsum w wsum wp
ERROR.wsum	Depends on current values of: PRED.wsum w wsum wp
ACTUAL.wsum	Current values affect: PRED.c RESID.c ERROR.c RESID.wsum ERROR.wsum

BLOCK Listing

The BLOCK option prints an analysis of the program variables based on the assignments in the model program. The output produced by the example is shown in Figure 19.97.

Figure 19.97 The BLOCK Output for Klein's Model

The MODEL Procedure
Model Structure Analysis
(Based on Assignments to Endogenous Model Variables)

Exogenous Variables	wp g t year
Endogenous Variables	c p w i x wsum k y

Block Structure of the System	
Block 1	c p w i x wsum

Dependency Structure of the System	
Block 1	Depends On All_Exogenous
k	Depends On Block 1 All_Exogenous
y	Depends On Block 1 All_Exogenous

One use for the block output is to put a model in recursive form. Simulations of the model can be done with the SEIDEL method, which is efficient if the model is recursive and if the equations are in recursive order. By examining the block output, you can determine how to reorder the model equations for the most efficient simulation.

Adjacency Graph

The GRAPH option displays the same information as the BLOCK option with the addition of an adjacency graph. An X in a column in an adjacency graph indicates that the variable associated with the row depends on the variable associated with the column. The output produced by the example is shown in Figure 19.98.

The first and last graphs are straightforward. The middle graph represents the dependencies of the nonexogenous variables after transitive closure has been performed (that is, A depends on B, and B depends on C, so A depends on C). The preceding transitive closure matrix indicates that K and Y do not directly or indirectly depend on each other.

Figure 19.98 The GRAPH Output for Klein's Model

Adjacency Matrix for Graph of System

Variable	c	p	w	i	x	wsum	k	y	wp	g	t	year
									*	*	*	*
c	X	X	.	.	.	X	.	.	.	.	.	.
p	.	X	X	.	X	.	.	.	.	.	X	.
w	.	.	X	.	X	.	.	.	.	.	.	X
i	.	X	.	X	.	.	.	.	.	.	.	.
x	X	.	.	X	X	.	.	.	.	X	.	.
wsum	.	.	X	.	.	X	.	.	X	.	.	.
k	.	.	.	X	.	.	X	.	.	.	.	.
y	X	.	.	X	.	.	.	X	.	X	X	.
wp	*	.	.	.	.	.	.	.	X	.	.	.
g	*	.	.	.	.	.	.	.	.	X	.	.
t	*	.	.	.	.	.	.	.	.	.	X	.
year	*	.	.	.	.	.	.	.	.	.	.	X

(Note: * = Exogenous Variable.)

Transitive Closure Matrix of Sorted System

Block	Variable	c	p	w	i	x	wsum	k	y
1	c	X	X	X	X	X	X	.	.
1	p	X	X	X	X	X	X	.	.
1	w	X	X	X	X	X	X	.	.
1	i	X	X	X	X	X	X	.	.
1	x	X	X	X	X	X	X	.	.
1	wsum	X	X	X	X	X	X	.	.
	k	X	X	X	X	X	X	X	.
	y	X	X	X	X	X	X	.	X

Adjacency Matrix for Graph of System Including Lagged Impacts

Block	Variable	c	p	w	i	x	wsum	k	y	wp	g	t	year
										*	*	*	*
1	c	X	L	.	.	.	X	.	.	.	.	.	.
1	p	.	X	X	.	X	.	.	.	.	.	X	.
1	w	.	.	X	.	L	.	.	.	.	.	.	X
1	i	.	L	.	X	.	.	L	.	.	.	.	.
1	x	X	.	.	X	X	.	.	.	.	X	.	.
1	wsum	.	.	X	.	.	X	.	.	X	.	.	.
	k	.	.	.	X	.	.	L	.	.	.	.	.
	y	X	.	.	X	.	.	.	X	.	X	X	.
	wp	*	.	.	.	.	.	.	.	X	.	.	.
	g	*	.	.	.	.	.	.	.	.	X	.	.
	t	*	.	.	.	.	.	.	.	.	.	X	.
	year	*	.	.	.	.	.	.	.	.	.	.	X

(Note: * = Exogenous Variable.)

Examples: MODEL Procedure

Example 19.1: OLS Single Nonlinear Equation

This example illustrates the use of the MODEL procedure for nonlinear ordinary least squares (OLS) regression. The model is a logistic growth curve for the population of the United States. The data is the population in millions recorded at ten-year intervals starting in 1790 and ending in 2000. For an explanation of the starting values given by the START= option, see the section "Troubleshooting Convergence Problems" on page 1153. Portions of the output from the following statements are shown in Output 19.1.1 through Output 19.1.3.

```
title 'Logistic Growth Curve Model of U.S. Population';
data uspop;
   input pop :6.3 @@;
   retain year 1780;
   year=year+10;
   label pop='U.S. Population in Millions';
   datalines;
3929  5308  7239   9638  12866  17069  23191  31443  39818 50155
62947 75994 91972 105710 122775 131669 151325 179323 203211
226542 248710
;

proc model data=uspop;
   label a = 'Maximum Population'
         b = 'Location Parameter'
         c = 'Initial Growth Rate';
   pop = a / ( 1 + exp( b - c * (year-1790) ) );
   fit pop start=(a 1000  b 5.5  c .02) / out=resid outresid;
run;
```

Output 19.1.1 Logistic Growth Curve Model Summary

Logistic Growth Curve Model of U.S. Population

The MODEL Procedure

Model Summary	
Model Variables	1
Parameters	3
Equations	1
Number of Statements	1

Model Variables	pop
Parameters(Value)	a(1000) b(5.5) c(0.02)
Equations	pop

The Equation to Estimate is
pop = F(a, b, c)

Output 19.1.2 Logistic Growth Curve Estimation Summary

Logistic Growth Curve Model of U.S. Population

The MODEL Procedure
OLS Estimation Summary

Data Set Options	
DATA=	USPOP
OUT=	RESID

Minimization Summary	
Parameters Estimated	3
Method	Gauss
Iterations	7
Subiterations	6
Average Subiterations	0.857143

Final Convergence Criteria	
R	0.00068
PPC(a)	0.000145
RPC(a)	0.001507
Object	0.000065
Trace(S)	19.20198
Objective Value	16.45884

Observations Processed	
Read	21
Solved	21

Output 19.1.3 Logistic Growth Curve Estimates

Logistic Growth Curve Model of U.S. Population

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq	Label
pop	3	18	345.6	19.2020	4.3820	0.9972	0.9969	U.S. Population in Millions

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
a	387.9307	30.0404	12.91	<.0001	Maximum Population
b	3.990385	0.0695	57.44	<.0001	Location Parameter
c	0.022703	0.00107	21.22	<.0001	Initial Growth Rate

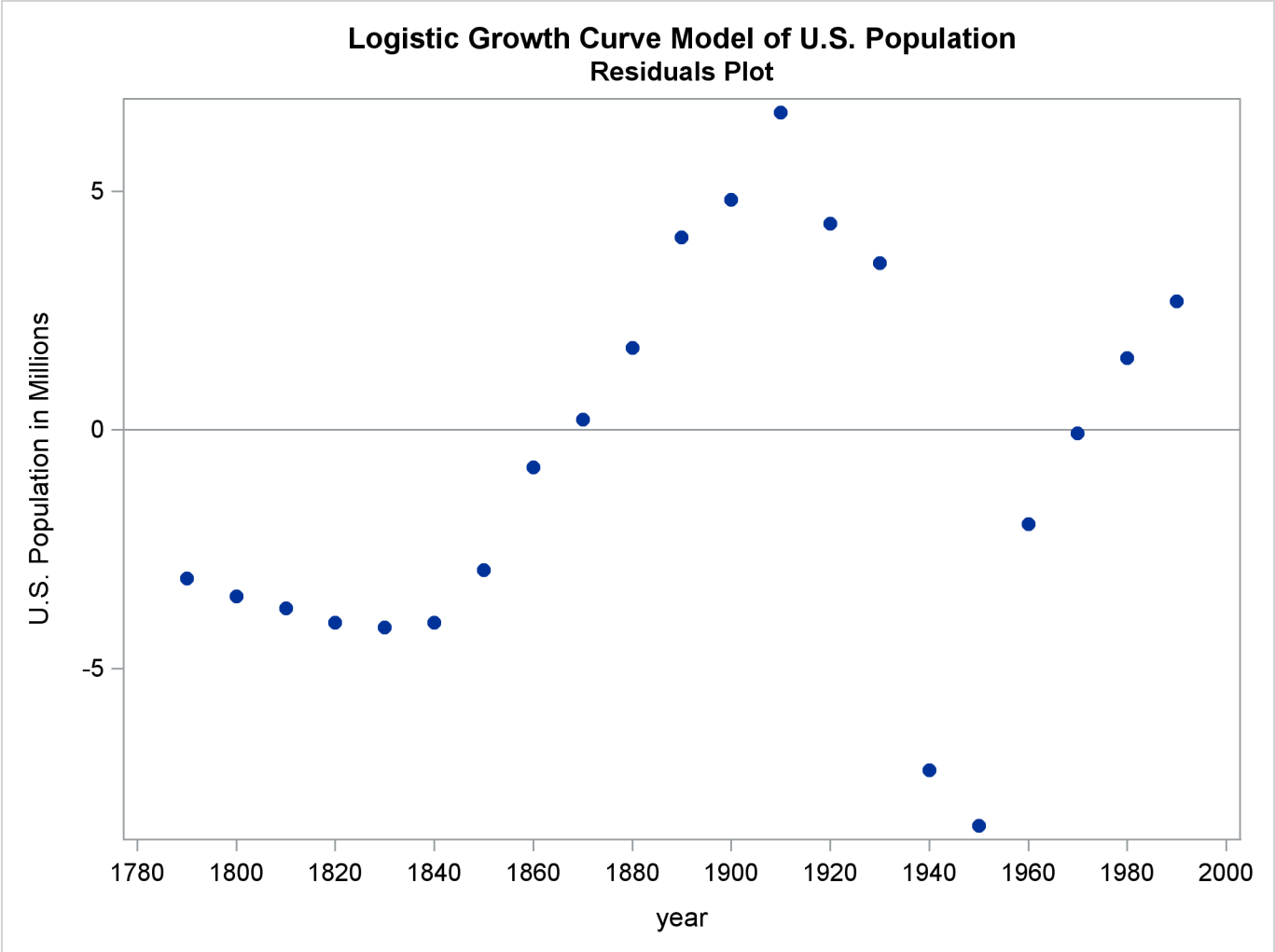
The adjusted R^2 value indicates the model fits the data well. There are only 21 observations and the model is nonlinear, so significance tests on the parameters are only approximate. The significance tests and associated approximate probabilities indicate that all the parameters are significantly different from 0.

The FIT statement included the options OUT=RESID and OUTRESID so that the residuals from the estimation are saved to the data set RESID. The residuals are plotted to check for heteroscedasticity by using PROC SGPLOT as follows.

```
title2 "Residuals Plot";
proc sgplot data=resid;
   refline 0;
   scatter x=year y=pop / markerattrs=(symbol=circlefilled);
   xaxis values=(1780 to 2000 by 20);
run;
```

The plot is shown in Output 19.1.4.

Output 19.1.4 Residual for Population Model (Actual–Predicted)



The residuals do not appear to be independent, and the model could be modified to explain the remaining nonrandom errors.

Example 19.2: A Consumer Demand Model

This example shows the estimation of a system of nonlinear consumer demand equations based on the translog functional form by using seemingly unrelated regression (SUR). Expenditure shares and corresponding normalized prices are given for three goods.

Since the shares add up to one, the system is singular; therefore, one equation is omitted from the estimation process. The choice of which equation to omit is arbitrary. The nonlinear system is first estimated in unrestricted form by the following statements:

```
title1 'Consumer Demand--Translog Functional Form';
title2 'Asymmetric Model';

proc model data=tlog1;
   endogenous share1 share2;
   parms a1 a2 b11 b12 b13 b21 b22 b23 b31 b32 b33;

   bm1 = b11 + b21 + b31;
   bm2 = b12 + b22 + b32;
   bm3 = b13 + b23 + b33;
   lp1 = log(p1);
   lp2 = log(p2);
   lp3 = log(p3);
   share1 = ( a1 + b11 * lp1 + b12 * lp2 + b13 * lp3 ) /
            ( -1 + bm1 * lp1 + bm2 * lp2 + bm3 * lp3 );
   share2 = ( a2 + b21 * lp1 + b22 * lp2 + b23 * lp3 ) /
            ( -1 + bm1 * lp1 + bm2 * lp2 + bm3 * lp3 );

   fit share1 share2
       start=( a1 -.14 a2 -.45 b11 .03 b12 .47 b22 .98 b31 .20
               b32 1.11 b33 .71 ) / outsused=smatrix sur;
run;
```

A portion of the printed output produced by this example is shown in Output 19.2.1 through Output 19.2.3.

Output 19.2.1 Translog Demand Model Summary

Consumer Demand--Translog Functional Form
Asymmetric Model

The MODEL Procedure

Model Summary	
Model Variables	2
Endogenous	2
Parameters	11
Equations	2
Number of Statements	8

Model Variables	share1 share2
Parameters(Value)	a1(-0.14) a2(-0.45) b11(0.03) b12(0.47) b13 b21 b22(0.98) b23 b31(0.2) b32(1.11) b33(0.71)
Equations	share1 share2

Output 19.2.1 *continued*

The 2 Equations to Estimate	
share1 =	F(a1, b11, b12, b13, b21, b22, b23, b31, b32, b33)
share2 =	F(a2, b11, b12, b13, b21, b22, b23, b31, b32, b33)

Output 19.2.2 Estimation Summary for the Unrestricted Model

NOTE: At SUR Iteration 2 CONVERGE=0.001 Criteria Met.

Consumer Demand--Translog Functional Form
Asymmetric Model

The MODEL Procedure
SUR Estimation Summary

Data Set Options	
DATA=	TLOG1
OUTSUSED=	SMATRIX

Minimization Summary	
Parameters Estimated	11
Method	Gauss
Iterations	2

Final Convergence Criteria	
R	0.00016
PPC(b11)	0.00116
RPC(b11)	0.012106
Object	2.921E-6
Trace(S)	0.000078
Objective Value	1.749312

Observations Processed	
Read	44
Solved	44

Output 19.2.3 Estimation Results for the Unrestricted Model

Consumer Demand--Translog Functional Form
Asymmetric Model

The MODEL Procedure

Nonlinear SUR Summary of Residual Errors							
Equation	**DF Model**	**DF Error**	**SSE**	**MSE**	**Root MSE**	**R-Square**	**Adj R-Sq**
share1	5.5	38.5	0.00166	0.000043	0.00656	0.8067	0.7841
share2	5.5	38.5	0.00135	0.000035	0.00592	0.9445	0.9380

Output 19.2.3 *continued*

Nonlinear SUR Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
a1	-0.14881	0.00225	-66.08	<.0001
a2	-0.45776	0.00297	-154.29	<.0001
b11	0.048382	0.0498	0.97	0.3379
b12	0.43655	0.0502	8.70	<.0001
b13	0.248588	0.0516	4.82	<.0001
b21	0.586326	0.2089	2.81	0.0079
b22	0.759776	0.2565	2.96	0.0052
b23	1.303821	0.2328	5.60	<.0001
b31	0.297808	0.1504	1.98	0.0550
b32	0.961551	0.1633	5.89	<.0001
b33	0.8291	0.1556	5.33	<.0001

Number of Observations		Statistics for System	
Used	44	**Objective**	1.7493
Missing	0	**Objective*N**	76.9697

The model is then estimated under the restriction of symmetry ($b_{ij} = b_{ji}$), as shown in the following statements:

```
title2 'Symmetric Model';
proc model data=tlog1;
   var share1 share2 p1 p2 p3;
   parms a1 a2 b11 b12 b22 b31 b32 b33;
   bm1 = b11 + b12 + b31;
   bm2 = b12 + b22 + b32;
   bm3 = b31 + b32 + b33;
   lp1 = log(p1);
   lp2 = log(p2);
   lp3 = log(p3);
   share1 = ( a1 + b11 * lp1 + b12 * lp2 + b31 * lp3 ) /
            ( -1 + bm1 * lp1 + bm2 * lp2 + bm3 * lp3 );
   share2 = ( a2 + b12 * lp1 + b22 * lp2 + b32 * lp3 ) /
            ( -1 + bm1 * lp1 + bm2 * lp2 + bm3 * lp3 );
   fit share1 share2
       start=( a1 -.14 a2 -.45 b11 .03 b12 .47 b22 .98 b31 .20
               b32 1.11 b33 .71 ) / sdata=smatrix sur;
run;
```

A portion of the printed output produced for the symmetry restricted model is shown in Output 19.2.4 and Output 19.2.5.

Output 19.2.4 Model Summary from the Restricted Model

Consumer Demand--Translog Functional Form Symmetric Model

The MODEL Procedure

The 2 Equations to Estimate	
share1 =	F(a1, b11, b12, b22, b31, b32, b33)
share2 =	F(a2, b11, b12, b22, b31, b32, b33)

Output 19.2.5 Estimation Results for the Restricted Model

Consumer Demand--Translog Functional Form Symmetric Model

The MODEL Procedure

Nonlinear SUR Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
share1	4	40	0.00166	0.000041	0.00644	0.8066	0.7920
share2	4	40	0.00139	0.000035	0.00590	0.9428	0.9385

Nonlinear SUR Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
a1	-0.14684	0.00135	-108.99	<.0001
a2	-0.4597	0.00167	-275.34	<.0001
b11	0.02886	0.00741	3.89	0.0004
b12	0.467827	0.0115	40.57	<.0001
b22	0.970079	0.0177	54.87	<.0001
b31	0.208143	0.00614	33.88	<.0001
b32	1.102415	0.0127	86.51	<.0001
b33	0.694245	0.0168	41.38	<.0001

Number of Observations		Statistics for System	
Used	44	**Objective**	1.7820
Missing	0	**Objective*N**	78.4097

Hypothesis testing requires that the **S** matrix from the unrestricted model be imposed on the restricted model, as explained in the section "Tests on Parameters" on page 1196. The **S** matrix saved in the data set SMATRIX is requested by the SDATA= option.

A chi-square test is used to see if the hypothesis of symmetry is accepted or rejected. (Oc–Ou) has a chi-square distribution asymptotically, where Oc is the constrained OBJECTIVE*N and Ou is the unconstrained OBJECTIVE*N. The degrees of freedom is equal to the difference in the number of free parameters in the two models.

In this example, Ou is 76.9697 and Oc is 78.4097, resulting in a difference of 1.44 with 3 degrees of freedom. You can obtain the probability value by using the following statements:

```
data _null_;
   /* probchi( reduced-full, n-restrictions )*/
   p = 1-probchi( 1.44, 3 );
   put p=;
run;
```

The output from this DATA step run is $p = 0.6961858724$. With this p-value you cannot reject the hypothesis of symmetry. This test is asymptotically valid.

Example 19.3: Vector AR(1) Estimation

This example shows the estimation of a two-variable vector AR(1) error process for the Grunfeld model (Grunfeld and Griliches 1960) by using the %AR macro. First, the full model is estimated. Second, the model is estimated with the restriction that the errors are univariate AR(1) instead of a vector process. The following statements produce Output 19.3.1 through Output 19.3.5.

```
data grunfeld;
   input year gei gef gec whi whf whc;
   label gei = 'Gross Investment GE'
         gec = 'Capital Stock Lagged GE'
         gef = 'Value of Outstanding Shares GE Lagged'
         whi = 'Gross Investment WH'
         whc = 'Capital Stock Lagged WH'
         whf = 'Value of Outstanding Shares Lagged WH';
datalines;
1935     33.1      1170.6    97.8      12.93     191.5     1.8
1936     45.0      2015.8    104.4     25.90     516.0     .8
1937     77.2      2803.3    118.0     35.05     729.0     7.4

   ... more lines ...


title1 'Example of Vector AR(1) Error Process Using Grunfeld''s Model';
/* Note: GE stands for General Electric
         WH stands for Westinghouse     */

proc model outmodel=grunmod;
   var gei whi gef gec whf whc;
   parms ge_int ge_f ge_c wh_int wh_f wh_c;
   label ge_int = 'GE Intercept'
         ge_f   = 'GE Lagged Share Value Coef'
         ge_c   = 'GE Lagged Capital Stock Coef'
         wh_int = 'WH Intercept'
         wh_f   = 'WH Lagged Share Value Coef'
         wh_c   = 'WH Lagged Capital Stock Coef';
   gei = ge_int + ge_f * gef + ge_c * gec;
   whi = wh_int + wh_f * whf + wh_c * whc;
run;
```

The preceding PROC MODEL step defines the structural model and stores it in the model file named GRUNMOD.

The following PROC MODEL step reads in the model, adds the vector autoregressive terms using %AR, and requests SUR estimation by using the FIT statement.

```
title2 'With Unrestricted Vector AR(1) Error Process';

proc model data=grunfeld model=grunmod;
   %ar( ar, 1, gei whi )
   fit gei whi / sur;
run;
```

The final PROC MODEL step estimates the restricted model, as shown in the following statements:

```
title2 'With restricted AR(1) Error Process';

proc model data=grunfeld model=grunmod;
   %ar( gei, 1 )
   %ar( whi, 1)
   fit gei whi / sur;
run;
```

Output 19.3.1 Model Summary for the Unrestricted Model

Example of Vector AR(1) Error Process Using Grunfeld's Model With Unrestricted Vector AR(1) Error Process

The MODEL Procedure

Model Summary	
Model Variables	6
Parameters	10
Equations	2
Number of Statements	7

Model Variables	gei whi gef gec whf whc
Parameters(Value)	ge_int ge_f ge_c wh_int wh_f wh_c ar_l1_1_1(0) ar_l1_1_2(0) ar_l1_2_1(0) ar_l1_2_2(0)
Equations	gei whi

The 2 Equations to Estimate	
gei =	F(ge_int, ge_f, ge_c, wh_int, wh_f, wh_c, ar_l1_1_1, ar_l1_1_2)
whi =	F(ge_int, ge_f, ge_c, wh_int, wh_f, wh_c, ar_l1_2_1, ar_l1_2_2)

NOTE: At SUR Iteration 9 CONVERGE=0.001 Criteria Met.

Output 19.3.2 Estimation Summary for the Unrestricted Model

Example of Vector AR(1) Error Process Using Grunfeld's Model With Unrestricted Vector AR(1) Error Process

The MODEL Procedure
SUR Estimation Summary

Data Set Options
DATA= GRUNFELD

Output 19.3.2 *continued*

Minimization Summary	
Parameters Estimated	10
Method	Gauss
Iterations	9

Final Convergence Criteria	
R	0.000609
PPC(wh_int)	0.002798
RPC(wh_int)	0.005411
Object	6.243E-7
Trace(S)	720.2454
Objective Value	1.374476

Observations Processed	
Read	20
Solved	20

Output 19.3.3 Estimation Results for the Unrestricted Model

Example of Vector AR(1) Error Process Using Grunfeld's Model With Unrestricted Vector AR(1) Error Process

The MODEL Procedure

Nonlinear SUR Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq	Label
gei	5	15	9374.5	625.0	24.9993	0.7910	0.7352	Gross Investment GE
whi	5	15	1429.2	95.2807	9.7612	0.7940	0.7391	Gross Investment WH

Nonlinear SUR Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
ge_int	-42.2858	30.5284	-1.39	0.1863	GE Intercept
ge_f	0.049894	0.0153	3.27	0.0051	GE Lagged Share Value Coef
ge_c	0.123946	0.0458	2.70	0.0163	GE Lagged Capital Stock Coef
wh_int	-4.68931	8.9678	-0.52	0.6087	WH Intercept
wh_f	0.068979	0.0182	3.80	0.0018	WH Lagged Share Value Coef
wh_c	0.019308	0.0754	0.26	0.8015	WH Lagged Capital Stock Coef
ar_l1_1_1	0.990902	0.3923	2.53	0.0233	AR(ar) gei: LAG1 parameter for gei
ar_l1_1_2	-1.56252	1.0882	-1.44	0.1716	AR(ar) gei: LAG1 parameter for whi
ar_l1_2_1	0.244161	0.1783	1.37	0.1910	AR(ar) whi: LAG1 parameter for gei
ar_l1_2_2	-0.23864	0.4957	-0.48	0.6372	AR(ar) whi: LAG1 parameter for whi

Output 19.3.4 Model Summary for the Restricted Model

Example of Vector AR(1) Error Process Using Grunfeld's Model
With restricted AR(1) Error Process

The MODEL Procedure

Model Summary	
Model Variables	6
Parameters	8
Equations	2
Number of Statements	7

Model Variables	gei whi gef gec whf whc
Parameters(Value)	ge_int ge_f ge_c wh_int wh_f wh_c gei_l1(0) whi_l1(0)
Equations	gei whi

Output 19.3.5 Estimation Results for the Restricted Model

Example of Vector AR(1) Error Process Using Grunfeld's Model
With restricted AR(1) Error Process

The MODEL Procedure

Nonlinear SUR Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq	Label
gei	4	16	10558.8	659.9	25.6890	0.7646	0.7204	Gross Investment GE
whi	4	16	1669.8	104.4	10.2157	0.7594	0.7142	Gross Investment WH

Nonlinear SUR Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
ge_int	-30.1239	29.7227	-1.01	0.3259	GE Intercept
ge_f	0.043527	0.0149	2.93	0.0099	GE Lagged Share Value Coef
ge_c	0.119206	0.0423	2.82	0.0124	GE Lagged Capital Stock Coef
wh_int	3.112671	9.2765	0.34	0.7416	WH Intercept
wh_f	0.053932	0.0154	3.50	0.0029	WH Lagged Share Value Coef
wh_c	0.038246	0.0805	0.48	0.6410	WH Lagged Capital Stock Coef
gei_l1	0.482397	0.2149	2.24	0.0393	AR(gei) gei lag1 parameter
whi_l1	0.455711	0.2424	1.88	0.0784	AR(whi) whi lag1 parameter

Example 19.4: MA(1) Estimation

This example estimates parameters for an MA(1) error process for the Grunfeld model, using both the unconditional least squares and the maximum likelihood methods. The ARIMA procedure estimates for Westinghouse equation are shown for comparison. The output of the following statements is summarized in Output 19.4.1:

```
proc model outmodel=grunmod;
   var gei whi gef gec whf whc;
   parms ge_int ge_f ge_c wh_int wh_f wh_c;
   label ge_int = 'GE Intercept'
         ge_f   = 'GE Lagged Share Value Coef'
         ge_c   = 'GE Lagged Capital Stock Coef'
         wh_int = 'WH Intercept'
         wh_f   = 'WH Lagged Share Value Coef'
         wh_c   = 'WH Lagged Capital Stock Coef';
   gei = ge_int + ge_f * gef + ge_c * gec;
   whi = wh_int + wh_f * whf + wh_c * whc;
run;

title1 'Example of MA(1) Error Process Using Grunfeld''s Model';
title2 'MA(1) Error Process Using Unconditional Least Squares';

proc model data=grunfeld model=grunmod;
   %ma(gei,1, m=uls);
   %ma(whi,1, m=uls);
   fit whi gei start=( gei_m1 0.8 -0.8) / startiter=2;
run;
```

Output 19.4.1 PROC MODEL Results by Using ULS Estimation

Example of MA(1) Error Process Using Grunfeld's Model
MA(1) Error Process Using Unconditional Least Squares

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq	Label
whi	4	16	1874.0	117.1	10.8224	0.7299	0.6793	Gross Investment WH
resid.whi		16	1295.6	80.9754	8.9986			Gross Investment WH
gei	4	16	13835.0	864.7	29.4055	0.6915	0.6337	Gross Investment GE
resid.gei		16	7646.2	477.9	21.8607			Gross Investment GE

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
ge_int	-26.839	32.0908	-0.84	0.4153	GE Intercept
ge_f	0.038226	0.0150	2.54	0.0217	GE Lagged Share Value Coef
ge_c	0.137099	0.0352	3.90	0.0013	GE Lagged Capital Stock Coef
wh_int	3.680835	9.5448	0.39	0.7048	WH Intercept
wh_f	0.049156	0.0172	2.85	0.0115	WH Lagged Share Value Coef
wh_c	0.067271	0.0708	0.95	0.3559	WH Lagged Capital Stock Coef
gei_m1	-0.87615	0.1614	-5.43	<.0001	MA(gei) gei lag1 parameter
whi_m1	-0.75001	0.2368	-3.17	0.0060	MA(whi) whi lag1 parameter

The estimation summary from the following PROC ARIMA statements is shown in Output 19.4.2.

```
title2 'PROC ARIMA Using Unconditional Least Squares';

proc arima data=grunfeld;
   identify var=whi cross=(whf whc ) noprint;
   estimate q=1 input=(whf whc) method=uls maxiter=40;
run;
```

Output 19.4.2 PROC ARIMA Results by Using ULS Estimation

Example of MA(1) Error Process Using Grunfeld's Model
PROC ARIMA Using Unconditional Least Squares

The ARIMA Procedure

Unconditional Least Squares Estimation

Parameter	Estimate	Standard Error	t Value	Approx Pr > |t|	Lag	Variable	Shift
MU	3.68608	9.54425	0.39	0.7044	0	whi	0
MA1,1	-0.75005	0.23704	-3.16	0.0060	1	whi	0
NUM1	0.04914	0.01723	2.85	0.0115	0	whf	0
NUM2	0.06731	0.07077	0.95	0.3557	0	whc	0

Constant Estimate	3.686077
Variance Estimate	80.97535
Std Error Estimate	8.998631
AIC	149.0044
SBC	152.9873
Number of Residuals	20

The model stored in Example 19.3 is read in by using the MODEL= option and the moving-average terms are added using the %MA macro.

The MA(1) model using maximum likelihood is estimated by using the following statements:

```
title2 'MA(1) Error Process Using Maximum Likelihood ';

proc model data=grunfeld model=grunmod;
   %ma(gei,1, m=ml);
   %ma(whi,1, m=ml);
   fit whi gei;
run;
```

For comparison, the model is estimated by using PROC ARIMA as follows:

```
title2 'PROC ARIMA Using Maximum Likelihood ';

proc arima data=grunfeld;
   identify var=whi cross=(whf whc) noprint;
   estimate q=1 input=(whf whc) method=ml;
run;
```

PROC ARIMA does not estimate systems, so only one equation is evaluated.

The estimation results are shown in Output 19.4.3 and Output 19.4.4. The small differences in the parameter values between PROC MODEL and PROC ARIMA can be eliminated by tightening the convergence criteria for both procedures.

Output 19.4.3 PROC MODEL Results by Using ML Estimation

Example of MA(1) Error Process Using Grunfeld's Model
MA(1) Error Process Using Maximum Likelihood

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq	Label
whi	4	16	1857.5	116.1	10.7746	0.7323	0.6821	Gross Investment WH
resid.whi		16	1344.0	84.0012	9.1652			Gross Investment WH
gei	4	16	13742.5	858.9	29.3071	0.6936	0.6361	Gross Investment GE
resid.gei		16	8095.3	506.0	22.4935			Gross Investment GE

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
ge_int	-25.002	34.2933	-0.73	0.4765	GE Intercept
ge_f	0.03712	0.0161	2.30	0.0351	GE Lagged Share Value Coef
ge_c	0.137788	0.0380	3.63	0.0023	GE Lagged Capital Stock Coef
wh_int	2.946761	9.5638	0.31	0.7620	WH Intercept
wh_f	0.050395	0.0174	2.89	0.0106	WH Lagged Share Value Coef
wh_c	0.066531	0.0729	0.91	0.3749	WH Lagged Capital Stock Coef
gei_m1	-0.78516	0.1942	-4.04	0.0009	MA(gei) gei lag1 parameter
whi_m1	-0.69389	0.2540	-2.73	0.0148	MA(whi) whi lag1 parameter

Output 19.4.4 PROC ARIMA Results by Using ML Estimation

Example of MA(1) Error Process Using Grunfeld's Model
PROC ARIMA Using Maximum Likelihood

The ARIMA Procedure

Maximum Likelihood Estimation

Parameter	Estimate	Standard Error	t Value	Approx Pr > |t|	Lag	Variable	Shift
MU	2.95645	9.20752	0.32	0.7481	0	whi	0
MA1,1	-0.69305	0.25307	-2.74	0.0062	1	whi	0
NUM1	0.05036	0.01686	2.99	0.0028	0	whf	0
NUM2	0.06672	0.06939	0.96	0.3363	0	whc	0

Constant Estimate	2.956449
Variance Estimate	81.29645
Std Error Estimate	9.016455
AIC	148.9113
SBC	152.8942
Number of Residuals	20

Example 19.5: Polynomial Distributed Lags by Using %PDL

This example shows the use of the %PDL macro for polynomial distributed lag models. Simulated data is generated so that Y is a linear function of six lags of X, with the lag coefficients following a quadratic polynomial. The model is estimated by using a fourth-degree polynomial, both with and without endpoint constraints. The example uses simulated data generated from the following model:

$$y_t = 10 + \sum_{z=0}^{6} f(z)x_{t-z} + \epsilon$$

$$f(z) = -5z^2 + 1.5z$$

The LIST option prints the model statements added by the %PDL macro. The following statements generate simulated data as shown:

```
/*-------------------------------------------------------------*/
/*  Generate Simulated Data for a Linear Model with a PDL on X  */
/*        y = 10 + x(6,2) + e                                   */
/*        pdl(x) = -5.*(lg)**2 + 1.5*(lg) + 0.                  */
/*-------------------------------------------------------------*/
data pdl;
   pdl2=-5.; pdl1=1.5; pdl0=0;
   array zz(i) z0-z6;
   do i=1 to 7;
      z=i-1;
      zz=pdl2*z**2 + pdl1*z + pdl0;
      end;
   do n=-11 to 30;
      x  =10*ranuni(1234567)-5;
      pdl=z0*x + z1*xl1 + z2*xl2 + z3*xl3 + z4*xl4 + z5*xl5 + z6*xl6;
      e  =10*rannor(1234567);
      y  =10+pdl+e;
      if n>=1 then output;
      xl6=xl5; xl5=xl4; xl4=xl3; xl3=xl2; xl2=xl1; xl1=x;
      end;
run;

title1 'Polynomial Distributed Lag Example';
title3 'Estimation of PDL(6,4) Model-- No Endpoint Restrictions';

proc model data=pdl;
   parms int;                  /* declare the intercept parameter */
   %pdl( xpdl, 6, 4 )          /* declare the lag distribution */
   y = int + %pdl( xpdl, x );  /* define the model equation */
   fit y / list;               /* estimate the parameters */
run;
```

The LIST output for the model without endpoint restrictions is shown in Output 19.5.1. The first seven statements in the generated program are the polynomial expressions for lag parameters XPDL_L0 through XPDL_L6. The estimated parameters are INT, XPDL_0, XPDL_1, XPDL_2, XPDL_3, and XPDL_4.

Output 19.5.1 PROC MODEL Listing of Generated Program

Polynomial Distributed Lag Example

Estimation of PDL(6,4) Model-- No Endpoint Restrictions

The MODEL Procedure

Listing of Compiled Program Code

Stmt	Line:Col	Statement as Parsed
1	4781:14	XPDL_L0 = XPDL_0;
2	4781:14	XPDL_L1 = XPDL_0 + XPDL_1 + XPDL_2 + XPDL_3 + XPDL_4;
3	4781:14	XPDL_L2 = XPDL_0 + XPDL_1 * 2 + XPDL_2 * 2 ** 2 + XPDL_3 * 2 ** 3 + XPDL_4 * 2 ** 4;
4	4781:14	XPDL_L3 = XPDL_0 + XPDL_1 * 3 + XPDL_2 * 3 ** 2 + XPDL_3 * 3 ** 3 + XPDL_4 * 3 ** 4;
5	4781:14	XPDL_L4 = XPDL_0 + XPDL_1 * 4 + XPDL_2 * 4 ** 2 + XPDL_3 * 4 ** 3 + XPDL_4 * 4 ** 4;
6	4781:14	XPDL_L5 = XPDL_0 + XPDL_1 * 5 + XPDL_2 * 5 ** 2 + XPDL_3 * 5 ** 3 + XPDL_4 * 5 ** 4;
7	4781:14	XPDL_L6 = XPDL_0 + XPDL_1 * 6 + XPDL_2 * 6 ** 2 + XPDL_3 * 6 ** 3 + XPDL_4 * 6 ** 4;
8	4782:4	PRED.y = int + XPDL_L0 * x + XPDL_L1 * LAG1(x) + XPDL_L2 * LAG2(x) + XPDL_L3 * LAG3(x) + XPDL_L4 * LAG4(x) + XPDL_L5 * LAG5(x) + XPDL_L6 * LAG6(x);
8	4782:4	RESID.y = PRED.y - ACTUAL.y;
8	4782:4	ERROR.y = PRED.y - y;
9	4781:15	ESTIMATE XPDL_L0, XPDL_L1, XPDL_L2, XPDL_L3, XPDL_L4, XPDL_L5, XPDL_L6;
10	4781:15	_est0 = XPDL_L0;
11	4781:15	_est1 = XPDL_L1;
12	4781:15	_est2 = XPDL_L2;
13	4781:15	_est3 = XPDL_L3;
14	4781:15	_est4 = XPDL_L4;
15	4781:15	_est5 = XPDL_L5;
16	4781:14	_est6 = XPDL_L6;

The FIT results for the model without endpoint restrictions are shown in Output 19.5.2.

Output 19.5.2 PROC MODEL Results That Specify No Endpoint Restrictions

Polynomial Distributed Lag Example

Estimation of PDL(6,4) Model-- No Endpoint Restrictions

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
y	6	18	2070.8	115.0	10.7259	0.9998	0.9998

Output 19.5.2 *continued*

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
int	9.621969	2.3238	4.14	0.0006	
XPDL_0	0.084374	0.7587	0.11	0.9127	PDL(XPDL,6,4) parameter for (L)**0
XPDL_1	0.749956	2.0936	0.36	0.7244	PDL(XPDL,6,4) parameter for (L)**1
XPDL_2	-4.196	1.6215	-2.59	0.0186	PDL(XPDL,6,4) parameter for (L)**2
XPDL_3	-0.21489	0.4253	-0.51	0.6195	PDL(XPDL,6,4) parameter for (L)**3
XPDL_4	0.016133	0.0353	0.46	0.6528	PDL(XPDL,6,4) parameter for (L)**4

Portions of the output produced by the following PDL model with endpoints of the model restricted to zero are presented in Output 19.5.3.

```
title3 'Estimation of PDL(6,4) Model-- Both Endpoint Restrictions';

proc model data=pdl ;
   parms int;                    /* declare the intercept parameter */
   %pdl( xpdl, 6, 4, r=both )    /* declare the lag distribution */
   y = int + %pdl( xpdl, x );    /* define the model equation */
   fit y /list;                  /* estimate the parameters */
run;
```

Output 19.5.3 PROC MODEL Results Specifying Both Endpoint Restrictions

Polynomial Distributed Lag Example

Estimation of PDL(6,4) Model-- Both Endpoint Restrictions

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
y	4	20	449868	22493.4	150.0	0.9596	0.9535

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
int	17.08581	32.4032	0.53	0.6038	
XPDL_2	13.88433	5.4361	2.55	0.0189	PDL(XPDL,6,4) parameter for (L)**2
XPDL_3	-9.3535	1.7602	-5.31	<.0001	PDL(XPDL,6,4) parameter for (L)**3
XPDL_4	1.032421	0.1471	7.02	<.0001	PDL(XPDL,6,4) parameter for (L)**4

Note that XPDL_0 and XPDL_1 are not shown in the estimate summary. They were used to satisfy the endpoint restrictions analytically by the generated %PDL macro code. Their values can be determined by back substitution.

To estimate the PDL model with one or more of the polynomial terms dropped, specify the largest degree of the polynomial desired with the %PDL macro and use the DROP= option in the FIT statement to remove the unwanted terms. The dropped parameters should be set to 0. The following PROC MODEL statements demonstrate estimation with a PDL of degree 2 without the 0th order term.

```
title3 'Estimation of PDL(6,2) Model-- With XPDL_0 Dropped';

proc model data=pdl list;
   parms int;                    /* declare the intercept parameter */
   %pdl( xpdl, 6, 2 )            /* declare the lag distribution */
   y = int + %pdl( xpdl, x );    /* define the model equation */
   xpdl_0 =0;
   fit y drop=xpdl_0;            /* estimate the parameters */
run;
```

The results from this estimation are shown in Output 19.5.4.

Output 19.5.4 PROC MODEL Results That Specify %PDL(XPDL, 6, 2)

Polynomial Distributed Lag Example

Estimation of PDL(6,2) Model-- With XPDL_0 Dropped

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
y	3	21	2114.1	100.7	10.0335	0.9998	0.9998

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|	Label
int	9.536382	2.1685	4.40	0.0003	
XPDL_1	1.883315	0.3159	5.96	<.0001	PDL(XPDL,6,2) parameter for (L)**1
XPDL_2	-5.08827	0.0656	-77.56	<.0001	PDL(XPDL,6,2) parameter for (L)**2

Example 19.6: General Form Equations

Data for this example are generated. General form equations are estimated and forecast by using PROC MODEL. The system is a basic supply and demand model.

The following statements specify the form of the model:

```
title1 "General Form Equations for Supply-Demand Model";

proc model outmodel=model;
   var price quantity income unitcost;
   parms d0-d2 s0-s2;
   eq.demand=d0+d1*price+d2*income-quantity;
   eq.supply=s0+s1*price+s2*unitcost-quantity;
run;
```

Three data sets are used in this example. The first data set, HISTORY, is used to estimate the parameters of the model. The ASSUME data set is used to produce a forecast of PRICE and QUANTITY. Notice that the ASSUME data set does not need to contain the variables PRICE and QUANTITY. The HISTORY data set is shown as follows:

```
data history;
   input year income unitcost price quantity;
datalines;
1976    2221.87    3.31220     0.17903    266.714
1977    2254.77    3.61647     0.06757    276.049
1978    2285.16    2.21601     0.82916    285.858

   ... more lines ...
```

The ASSUME data set is shown as follows:

```
data assume;
   input year income unitcost;
datalines;
1986    2571.87    2.31220
1987    2609.12    2.45633
1988    2639.77    2.51647
1989    2667.77    1.65617
1990    2705.16    1.01601
;
```

The third data set, GOAL, used in a forecast of PRICE and UNITCOST as a function of INCOME and QUANTITY is as follows:

```
data goal;
   input year income quantity;
datalines;
1986    2571.87     371.4
1987    2721.08     416.5
1988    3327.05     597.3
1989    3885.85     764.1
1990    3650.98     694.3
;
```

The following statements fit the model to the HISTORY data set and solve the fitted model for the ASSUME data set.

```
proc model model=model outmodel=model;

/* estimate the model parameters */
   fit supply demand / data=history outest=est n2sls;
   instruments income unitcost year;
run;

/* produce forecasts for income and unitcost assumptions */
   solve price quantity / data=assume out=pq;
run;

title2 "Parameter Estimates for the System";
proc print data=est;
run;

title2 "Price Quantity Solution";
proc print data=pq;
run;
```

The model summary of the supply and demand model is shown as follows:

Output 19.6.1 Model Summary

General Form Equations for Supply-Demand Model

The MODEL Procedure

Model Summary	
Model Variables	4
Parameters	6
Equations	2
Number of Statements	3

Model Variables	price quantity income unitcost
Parameters	d0 d1 d2 s0 s1 s2
Equations	demand supply

The 2 Equations to Estimate	
supply =	F(s0(1), s1(price), s2(unitcost))
demand =	F(d0(1), d1(price), d2(income))
Instruments	1 income unitcost year

The estimation results are shown in Output 19.6.2 and the OUTEST= data set is show in Output 19.6.3. The output data set produced by the SOLVE statement is shown in Output 19.6.4.

Output 19.6.2 Output from the FIT Statement

General Form Equations for Supply-Demand Model

The MODEL Procedure

Nonlinear 2SLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
supply	3	7	3.3240	0.4749	0.6891		
demand	3	7	1.0829	0.1547	0.3933		

Nonlinear 2SLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
d0	-395.887	4.1841	-94.62	<.0001
d1	0.717328	0.5673	1.26	0.2466
d2	0.298061	0.00187	159.65	<.0001
s0	-107.62	4.1780	-25.76	<.0001
s1	201.5711	1.5977	126.16	<.0001
s2	102.2116	1.1217	91.12	<.0001

Output 19.6.3 Listing of OUTEST= Data Set Created in the FIT Statement

General Form Equations for Supply-Demand Model
Parameter Estimates for the System

Obs	_NAME_	_TYPE_	_STATUS_	_NUSED_	d0	d1	d2	s0	s1	s2
1		2SLS	0 Converged	10	-395.887	0.71733	0.29806	-107.620	201.571	102.212

Output 19.6.4 Listing of OUT= Data Set Created in the First SOLVE Statement

General Form Equations for Supply-Demand Model
Price Quantity Solution

Obs	_TYPE_	_MODE_	_ERRORS_	price	quantity	income	unitcost	year
1	PREDICT	SIMULATE	0	1.20473	371.552	2571.87	2.31220	1986
2	PREDICT	SIMULATE	0	1.18666	382.642	2609.12	2.45633	1987
3	PREDICT	SIMULATE	0	1.20154	391.788	2639.77	2.51647	1988
4	PREDICT	SIMULATE	0	1.68089	400.478	2667.77	1.65617	1989
5	PREDICT	SIMULATE	0	2.06214	411.896	2705.16	1.01601	1990

The following statements produce the goal-seeking solutions for PRICE and UNITCOST by using the GOAL dataset.

```
title2 "Price Unitcost Solution";

/* produce goal-seeking solutions for
     income and quantity assumptions*/
proc model model=model;
   solve price unitcost / data=goal out=pc;
run;

proc print data=pc;
run;
```

The output data set produced by the final SOLVE statement is shown in Output 19.6.5.

Output 19.6.5 Listing of OUT= Data Set Created in the Second SOLVE Statement

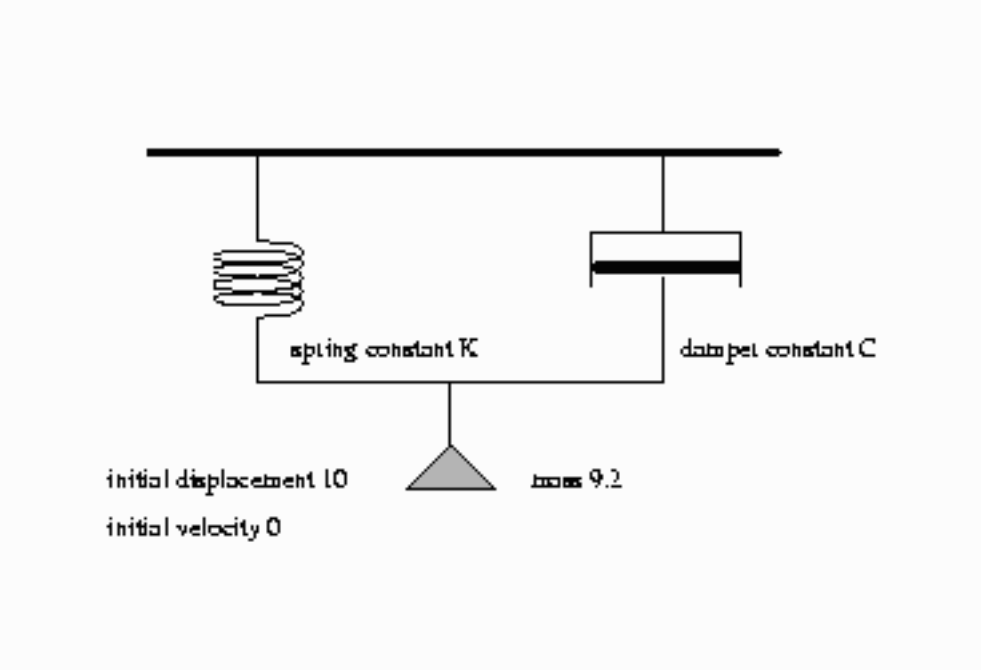
General Form Equations for Supply-Demand Model
Price Unitcost Solution

Obs	_TYPE_	_MODE_	_ERRORS_	price	quantity	income	unitcost	year
1	PREDICT	SIMULATE	0	0.99284	371.4	2571.87	2.72857	1986
2	PREDICT	SIMULATE	0	1.86594	416.5	2721.08	1.44798	1987
3	PREDICT	SIMULATE	0	2.12230	597.3	3327.05	2.71130	1988
4	PREDICT	SIMULATE	0	2.46166	764.1	3885.85	3.67395	1989
5	PREDICT	SIMULATE	0	2.74831	694.3	3650.98	2.42576	1990

Example 19.7: Spring and Damper Continuous System

This model simulates the mechanical behavior of a spring and damper system shown in Figure 19.99.

Figure 19.99 Spring and Damper System Model



A mass is hung from a spring with spring constant K. The motion is slowed by a damper with damper constant C. The damping force is proportional to the velocity, while the spring force is proportional to the displacement.

This is actually a continuous system; however, the behavior can be approximated by a discrete time model. We approximate the differential equation

$$\frac{\partial\, disp}{\partial\, time} = velocity$$

with the difference equation

$$\frac{\Delta\, disp}{\Delta\, time} = velocity$$

This is rewritten as

$$\frac{disp - \mathrm{LAG}(disp)}{dt} = velocity$$

where *dt* is the time step used. In PROC MODEL, this is expressed with the program statement

```
disp = lag(disp) + vel * dt;
```

or

```
dert.disp = vel;
```

The first statement is simply a computing formula for Euler's approximation for the integral

$$disp = \int velocity\ dt$$

If the time step is small enough with respect to the changes in the system, the approximation is good. Although PROC MODEL does not have the variable step-size and error-monitoring features of simulators designed for continuous systems, the procedure is a good tool to use for less challenging continuous models.

The second form instructs the MODEL procedure to do the integration for you.

This model is unusual because there are no exogenous variables, and endogenous data are not needed. Although you still need a SAS data set to count the simulation periods, no actual data are brought in.

Since the variables DISP and VEL are lagged, initial values specified in the VAR statement determine the starting state of the system. The mass, time step, spring constant, and damper constant are declared and initialized by a CONTROL statement as shown in the following statements:

```
title1 'Simulation of Spring-Mass-Damper System';

/*- Data to drive the simulation time periods ---*/
data one;
   do n=1 to 100;
      output;
   end;
run;

proc model data=one outmodel=spring;
   var      force -200  disp  10  vel  0  accel -20  time 0;
   control  mass   9.2  c    1.5  dt  .1  k      20;
   force = -k * disp -c * vel;
   disp  = lag(disp) + vel * dt;
   vel   = lag(vel) + accel * dt;
   accel = force / mass;
   time  = lag(time) + dt;
run;
```

The displacement scale is zeroed at the point where the force of gravity is offset, so the acceleration of the gravity constant is omitted from the force equation. The control variable C and K represent the damper and the spring constants respectively.

The model is simulated three times, and the simulation results are written to output data sets. The first run uses the original initial conditions specified in the VAR statement. In the second run, the initial displacement is doubled; the results show that the period of the motion is unaffected by the amplitude. In the third run, the DERT. syntax is used to do the integration. Notice that the path of the displacement is close to the old path, indicating that the original time step is short enough to yield an accurate solution. These simulations are performed by the following statements:

```
proc model data=one model=spring;
   title2 "Simulation of the model for the base case";
   control run '1';
   solve / out=a;
run;

   title2 "Simulation of the model with twice the initial displacement";
   control run '2';
   var disp 20;
   solve / out=b;
run;

data two;
   do time = 0 to 10 by .2; output;end;
run;

title2 "Simulation of the model using the dert. syntax";
proc model data=two;
   var      force -200  disp  10  vel  0  accel -20  time 0;
   control  mass   9.2  c    1.5  dt  .1  k      20;
   control run '3' ;
   force = -k * disp -c * vel;
   dert.disp  = vel ;
   dert.vel   = accel;
   accel = force / mass;
   solve / out=c;
   id time ;
run;
```

The output SAS data sets that contain the solution results are merged and the displacement time paths for the three simulations are plotted. The three runs are identified on the plot as 1, 2, and 3. The following statements produce Output 19.7.1 through Output 19.7.5.

```
data p;
   set a b c;
run;

title2 'Overlay Plot of All Three Simulations';
proc sgplot data=p;
   series x=time y=disp / group=run lineattrs=(pattern=1);
   xaxis values=(0 to 10 by 1);
   yaxis values=(-20 to 20 by 10);
run;
```

Output 19.7.1 Model Summary

Simulation of Spring-Mass-Damper System
Simulation of the model for the base case

The MODEL Procedure

Model Summary	
Model Variables	5
Control Variables	5
Equations	5
Number of Statements	6
Program Lag Length	1

Model Variables	force(-200) disp(10) vel(0) accel(-20) time(0)
Control Variables	mass(9.2) c(1.5) dt(0.1) k(20) run(1)
Equations	force disp vel accel time

Output 19.7.2 Printed Output Produced by PROC MODEL SOLVE Statements

Simulation of Spring-Mass-Damper System
Simulation of the model for the base case

The MODEL Procedure
Dynamic Simultaneous Simulation

Data Set Options	
DATA=	ONE
OUT=	A

Solution Summary	
Variables Solved	5
Simulation Lag Length	1
Solution Method	NEWTON
CONVERGE=	1E-8
Maximum CC	8.68E-15
Maximum Iterations	1
Total Iterations	99
Average Iterations	1

Observations Processed	
Read	100
Lagged	1
Solved	99
First	2
Last	100

Variables Solved For	force disp vel accel time

Output 19.7.3 Printed Output Produced by PROC MODEL SOLVE Statements

Simulation of Spring-Mass-Damper System
Simulation of the model with twice the initial displacement

The MODEL Procedure
Dynamic Simultaneous Simulation

Data Set Options	
DATA=	ONE
OUT=	B

Solution Summary	
Variables Solved	5
Simulation Lag Length	1
Solution Method	NEWTON
CONVERGE=	1E-8
Maximum CC	2.64E-14
Maximum Iterations	1
Total Iterations	99
Average Iterations	1

Observations Processed	
Read	100
Lagged	1
Solved	99
First	2
Last	100

Variables Solved For	force disp vel accel time

Output 19.7.4 Printed Output Produced by PROC MODEL SOLVE Statements

Simulation of Spring-Mass-Damper System
Simulation of the model using the dert. syntax

The MODEL Procedure
Simultaneous Simulation

Data Set Options	
DATA=	TWO
OUT=	C

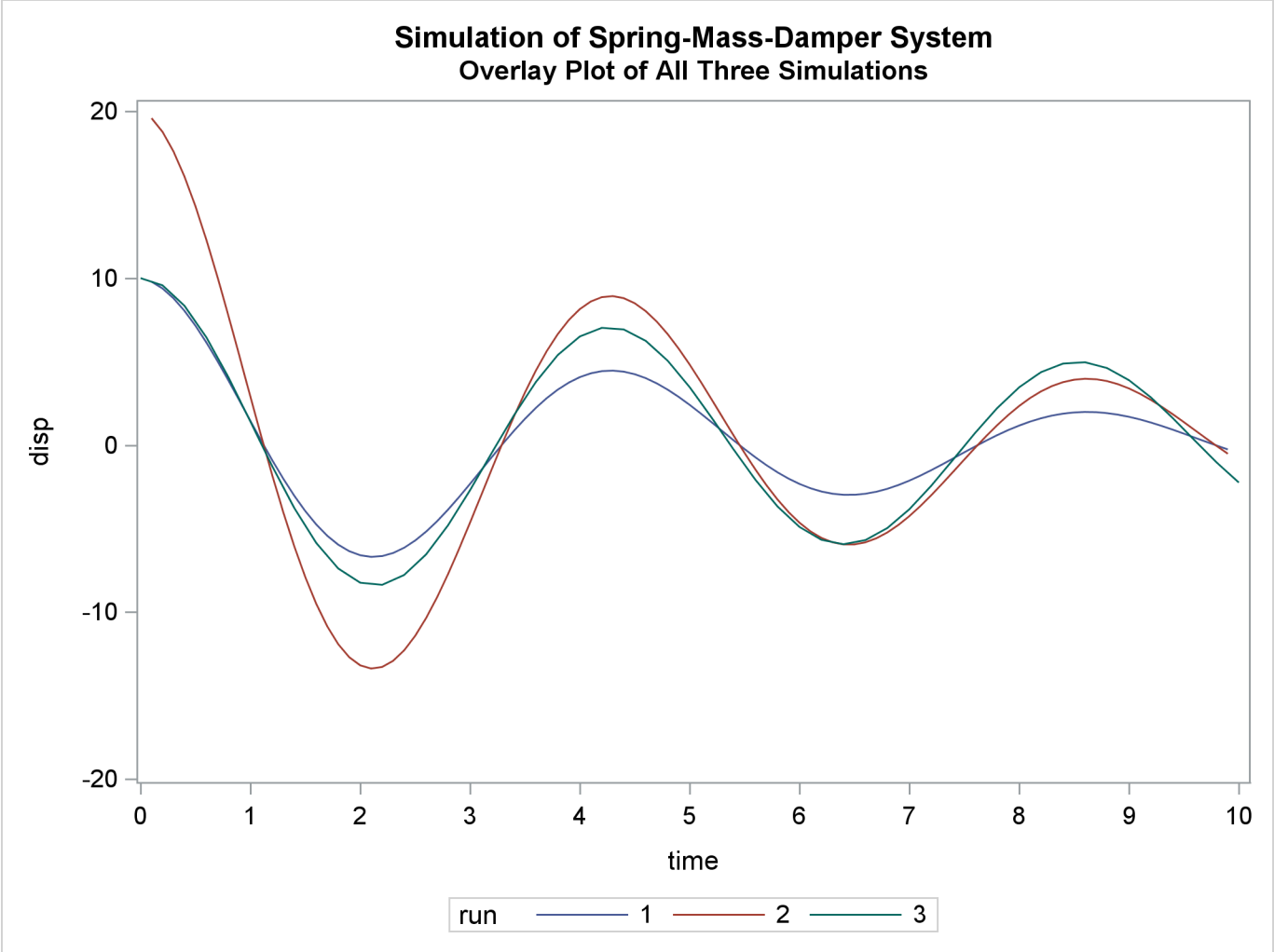
Solution Summary	
Variables Solved	4
Solution Method	NEWTON
Maximum Iterations	0

Output 19.7.4 *continued*

Observations Processed	
Read	51
Solved	51

Variables Solved For	force disp vel accel
ODE's	dert.disp dert.vel
Auxiliary Equations	force accel

Output 19.7.5 Overlay Plot of Three Simulations



Example 19.8: Nonlinear FIML Estimation

The data and model for this example were obtained from Bard (1974, pp. 133–138). The example is a two-equation econometric model used by Bodkin and Klein to fit U.S production data for the years 1909–1949.

The model is the following:

$$g_1 = c_1 10^{c_2 z_4}(c_5 z_1^{-c_4} + (1-c_5)z_2^{-c_4})^{-c_3/c_4} - z_3 = 0$$

$$g_2 = [c_5/(1-c_5)](z_1/z_2)^{(-1-c_4)} - z_5 = 0$$

where z_1 is capital input, z_2 is labor input, z_3 is real output, z_4 is time in years with 1929 as year zero, and z_5 is the ratio of price of capital services to wage scale. The c_i's are the unknown parameters. z_1 and z_2 are considered endogenous variables. A FIML estimation is performed by using the following statements:

```
data bodkin;
   input z1 z2 z3 z4 z5;
datalines;
1.33135 0.64629 0.4026 -20 0.24447
1.39235 0.66302 0.4084 -19 0.23454
1.41640 0.65272 0.4223 -18 0.23206

   ... more lines ...


title1 "Nonlinear FIML Estimation";

proc model data=bodkin;
   parms c1-c5;
   endogenous z1 z2;
   exogenous z3 z4 z5;

   eq.g1 = c1 * 10 **(c2 * z4) * (c5*z1**(-c4)+
           (1-c5)*z2**(-c4))**(-c3/c4) - z3;
   eq.g2 = (c5/(1-c5))*(z1/z2)**(-1-c4) -z5;

   fit g1 g2 / fiml ;
run;
```

When FIML estimation is selected, the log likelihood of the system is output as the objective value. The results of the estimation are shown in Output 19.8.1.

Output 19.8.1 FIML Estimation Results for U.S. Production Data

Nonlinear FIML Estimation

The MODEL Procedure

Nonlinear FIML Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
g1	4	37	0.0529	0.00143	0.0378		
g2	1	40	0.0173	0.000431	0.0208		

Output 19.8.1 *continued*

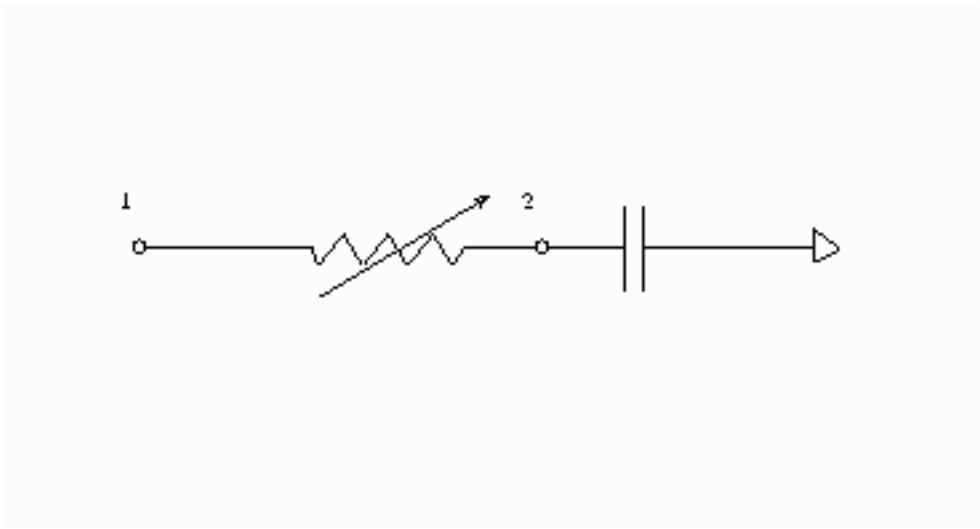
Nonlinear FIML Parameter Estimates				
Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
c1	0.58395	0.0218	26.76	<.0001
c2	0.005877	0.000673	8.74	<.0001
c3	1.3636	0.1148	11.87	<.0001
c4	0.473688	0.2699	1.75	0.0873
c5	0.446748	0.0596	7.49	<.0001

Number of Observations		Statistics for System	
Used	41	Log Likelihood	110.7773
Missing	0		

Example 19.9: Circuit Estimation

Consider the nonlinear circuit shown in Figure 19.100.

Figure 19.100 Nonlinear Resistor Capacitor Circuit



The theory of electric circuits is governed by Kirchhoff's laws: the sum of the currents flowing to a node is zero, and the net voltage drop around a closed loop is zero. In addition to Kirchhoff's laws, there are relationships between the current I through each element and the voltage drop V across the elements. For the circuit in Figure 19.100, the relationships are

$$C\frac{dV}{dt} = I$$

for the capacitor and

$$V = (R_1 + R_2(1 - \exp(-V)))I$$

for the nonlinear resistor. The following differential equation describes the current at node 2 as a function of time and voltage for this circuit:

$$C\frac{d\,V_2}{dt} - \frac{V_1 - V_2}{R_1 + R_2(1 - \exp(-V))} = 0$$

This equation can be written in the form

$$\frac{d\,V_2}{dt} = \frac{V_1 - V_2}{(R_1 + R_2(1 - \exp(-V)))C}$$

Consider the following data.

```
data circ;
   input v2 v1 time@@;
datalines;
-0.00007 0.0 0.0000000001 0.00912 0.5 0.0000000002
 0.03091 1.0 0.0000000003 0.06419 1.5 0.0000000004
 0.11019 2.0 0.0000000005 0.16398 2.5 0.0000000006
 0.23048 3.0 0.0000000007 0.30529 3.5 0.0000000008
 0.39394 4.0 0.0000000009 0.49121 4.5 0.0000000010
 0.59476 5.0 0.0000000011 0.70285 5.0 0.0000000012
 0.81315 5.0 0.0000000013 0.90929 5.0 0.0000000014
 1.01412 5.0 0.0000000015 1.11386 5.0 0.0000000016
 1.21106 5.0 0.0000000017 1.30237 5.0 0.0000000018
 1.40461 5.0 0.0000000019 1.48624 5.0 0.0000000020
 1.57894 5.0 0.0000000021 1.66471 5.0 0.0000000022
;
```

You can estimate the parameters in the preceding equation by using the following SAS statements:

```
title1 'Circuit Model Estimation Example';

proc model data=circ mintimestep=1.0e-23;
   parm R2 2000  R1 4000 C 5.0e-13;
   dert.v2 = (v1-v2)/((r1 + r2*(1-exp( -(v1-v2)))) * C);
   fit v2;
run;
```

The results of the estimation are shown in Output 19.9.1.

Output 19.9.1 Circuit Estimation

Circuit Model Estimation Example

The MODEL Procedure

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
R2	3002.471	1517.1	<------	Biased
R1	4984.842	1466.8	<------	Biased
C	5E-13	0	<------	Biased

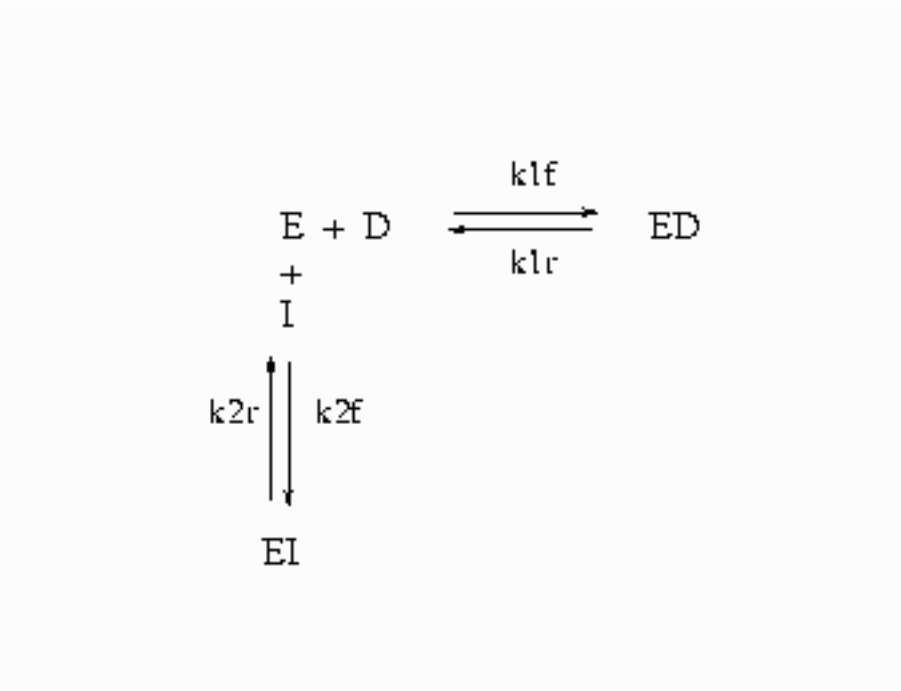
Note: The model was singular. Some estimates are marked 'Biased'.

In this case, the model equation is such that there is linear dependency that causes biased results and inflated variances. The Jacobian matrix is singular or nearly singular, but eliminating one of the parameters is not a solution in this case.

Example 19.10: Systems of Differential Equations

The following is a simplified reaction scheme for the competitive inhibitors with recombinant human renin (Morelock et al. 1995).

Figure 19.101 Competitive Inhibition of Recombinant Human Renin



In Figure 19.101, E=enzyme, D=probe, and I=inhibitor.

The differential equations that describe this reaction scheme are as follows:

$$\frac{dD}{dt} = k1r*ED - k1f*E*D$$

$$\frac{dED}{dt} = k1f*E*D - k1r*ED$$

$$\frac{dE}{dt} = k1r*ED - k1f*E*D + k2r*EI - k2f*E*I$$

$$\frac{dEI}{dt} = k2f*E*I - k2r*EI$$

$$\frac{dI}{dt} = k2r*EI - k2f*E*I$$

For this system, the initial values for the concentrations are derived from equilibrium considerations (as a function of parameters) or are provided as known values.

The experiment used to collect the data was carried out in two ways; preincubation (type='disassoc') and no preincubation (type='assoc'). The data also contain repeated measurements. The data contain values for fluorescence F, which is a function of concentration. Since there are no direct data for the concentrations, all the differential equations are simulated dynamically.

The SAS statements used to fit this model are as follows:

```
title1 'Systems of Differential Equations Example';

proc sort data=fit;
   by type time;
run;

%let k1f = 6.85e6  ;
%let k1r = 3.43e-4 ;
%let k2f = 1.8e7  ;
%let k2r = 2.1e-2 ;

%let qf  = 2.1e8  ;
%let qb  = 4.0e9  ;

%let dt  = 5.0e-7 ;
%let et  = 5.0e-8 ;
%let it  = 8.05e-6 ;

proc model data=fit;

   parameters qf  = 2.1e8
              qb  = 4.0e9
              k2f = 1.8e5
              k2r = 2.1e-3
              l   = 0;

              k1f = 6.85e6;
              k1r = 3.43e-4;

      /* Initial values for concentrations */
   control dt 5.0e-7
           et 5.0e-8
           it 8.05e-6;

      /* Association initial values ---------------*/
   if type = 'assoc' and time=0 then do;
      ed = 0;
      /* solve quadratic equation ----------*/
      a = 1;
      b = -(&it+&et+(k2r/k2f));
      c = &it*&et;
      ei = (-b-(((b**2)-(4*a*c))**.5))/(2*a);
      d = &dt-ed;
      i = &it-ei;
      e = &et-ed-ei;
      end;
```

```
      /* Disassociation initial values ----------*/
   if type = 'disassoc' and time=0 then do;
      ei = 0;
      a = 1;
      b = -(&dt+&et+(&k1r/&k1f));
      c = &dt*&et;
      ed = (-b-(((b**2)-(4*a*c))**.5))/(2*a);
      d = &dt-ed;
      i = &it-ei;
      e = &et-ed-ei;
   end;

   if time ne 0 then do;

      dert.d = k1r* ed  - k1f *e *d;

      dert.ed = k1f* e *d - k1r*ed;

      dert.e = k1r* ed - k1f* e * d  + k2r * ei - k2f * e *i;

      dert.ei = k2f* e *i - k2r * ei;

      dert.i = k2r * ei - k2f* e *i;

      end;

   /* L - offset between curves  */
   if type = 'disassoc' then
      F = (qf*(d-ed)) + (qb*ed) -L;
   else
      F = (qf*(d-ed)) + (qb*ed);

   fit F / method=marquardt;
run;
```

This estimation requires the repeated simulation of a system of 41 differential equations (5 base differential equations and 36 differential equations to compute the partials with respect to the parameters).

The results of the estimation are shown in Output 19.10.1.

Output 19.10.1 Kinetics Estimation

Systems of Differential Equations Example

The MODEL Procedure

Nonlinear OLS Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq
f	5	797	2525.0	3.1681	1.7799	0.9980	0.9980

Output 19.10.1 *continued*

Nonlinear OLS Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
qf	2.0413E8	681443	299.55	<.0001
qb	4.2263E9	9133195	462.74	<.0001
k2f	6451186	866998	7.44	<.0001
k2r	0.007808	0.00103	7.55	<.0001
l	-5.76974	0.4138	-13.94	<.0001

Example 19.11: Monte Carlo Simulation

This example illustrates how the form of the error in a ODE model affects the results from a static and dynamic estimation. The differential equation studied is

$$\frac{dy}{dt} = a - ay$$

The analytical solution to this differential equation is

$$y = 1 - \exp(-at)$$

The first data set contains errors that are strictly additive and independent. The data for this estimation are generated by the following DATA step:

```
data drive1;
   a = 0.5;
   do iter=1 to 100;
      do time = 0 to 50;
         y = 1 - exp(-a*time) + 0.1 *rannor(123);
         output;
      end;
   end;
run;
```

The second data set contains errors that are cumulative in form.

```
data drive2;
   a = 0.5;
   yp = 1.0 + 0.01 *rannor(123);
   do iter=1 to 100;
      do time = 0 to 50;
         y = 1 - exp(-a)*(1 - yp);
         yp = y + 0.01 *rannor(123);
         output;
      end;
   end;
run;
```

The following statements perform the 100 static estimations for each data set:

```
title1 'Monte Carlo Simulation of ODE';

proc model data=drive1 noprint;
   parm a 0.5;
   dert.y = a - a * y;
   fit y / outest=est;
   by iter;
run;
```

Similar statements are used to produce 100 dynamic estimations with a fixed and an unknown initial value. The first value in the data set is used to simulate an error in the initial value. The following PROC UNIVARIATE statements process the estimations:

```
proc univariate data=est noprint;
   var a;
   output out=monte mean=mean p5=p5 p95=p95;
run;

proc print data=monte;
run;
```

The results of these estimations are summarized in Table 19.6.

Table 19.6 Monte Carlo Summary, A=0.5

Estimation Type	Additive Error			Cumulative Error		
	mean	p95	p5	mean	p95	p5
static	0.77885	1.03524	0.54733	0.57863	1.16112	0.31334
dynamic fixed	0.48785	0.63273	0.37644	3.8546E24	8.88E10	-51.9249
dynamic unknown	0.48518	0.62452	0.36754	641704.51	1940.42	-25.6054

For this example model, it is evident that the static estimation is the least sensitive to misspecification.

Example 19.12: Cauchy Distribution Estimation

In this example a nonlinear model is estimated by using the Cauchy distribution. Then a simulation is done for one observation in the data.

The following DATA step creates the data for the model.

```
/* Generate a Cauchy distributed Y */
data c;
   format date monyy.;
   call streaminit(156789);
   do t=0 to 20 by 0.1;
      date=intnx('month','01jun90'd,(t*10)-1);
      x=rand('normal');
      e=rand('cauchy') + 10 ;
      y=exp(4*x)+e;
      output;
   end;
run;
```

The model to be estimated is

$$\begin{aligned} y &= e^{-a\,x} + \epsilon \\ \epsilon &\sim \text{Cauchy}(nc) \end{aligned}$$

That is, the residuals of the model are distributed as a Cauchy distribution with noncentrality parameter nc.

The log likelihood for the Cauchy distribution is

$$\text{ll} = -\log \pi(1 + (x - nc)^2)$$

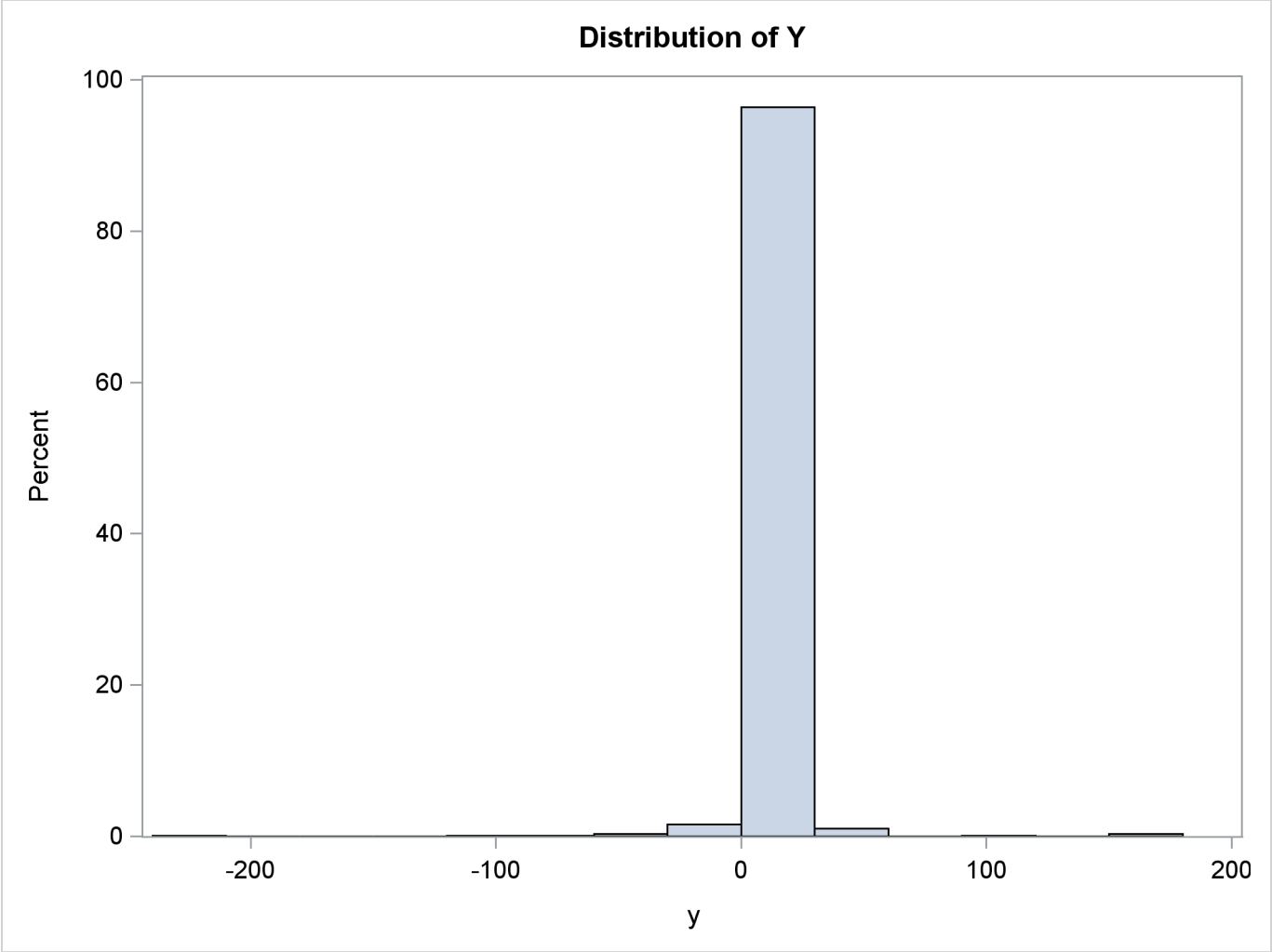
The following SAS statements specify the model and the log-likelihood function.

```
title1 'Cauchy Distribution';

proc model data=c ;
   dependent y;
   parm a -2 nc 4;
   y=exp(-a*x);

       /* Likelihood function for the residuals */
   obj = log(constant('pi')*(1+(-resid.y-nc)**2));

   errormodel y ~ general(obj) cdf=cauchy(nc);

   fit y / outsn=s1 method=marquardt;
   solve y / sdata=s1 data=c(obs=1) random=1000
             seed=256789 out=out1;
run;

title 'Distribution of Y';
proc sgplot data=out1;
   histogram y;
run;
```

The FIT statement uses the OUTSN= option to output the $\mathbf{\Sigma}$ matrix for residuals from the normal distribution. The $\mathbf{\Sigma}$ matrix is 1×1 and has value 1.0 because it is a correlation matrix. The OUTS= matrix is the scalar 2989.0. Because the distribution is univariate (no covariances), the OUTS= option would produce the same simulation results. The simulation is performed by using the SOLVE statement.

The distribution of y is shown in the following output.

Output 19.12.1 Distribution of Y



Example 19.13: Switching Regression Example

Take the usual linear regression problem

$$y = \mathbf{X}\beta + u$$

where Y denotes the n column vector of the dependent variable, $\mathbf{X}$ denotes the $(n \times k)$ matrix of independent variables, β denotes the k column vector of coefficients to be estimated, n denotes the number of observations (i =1, 2, . . . , n), and k denotes the number of independent variables.

You can take this basic equation and split it into two regimes, where the *i*th observation on y is generated by one regime or the other:

$$y_i = \sum_{j=1}^{k} \beta_{1j} X_{ji} + u_{1i} = x_i'\beta_1 + u_{1i}$$
$$y_i = \sum_{j=1}^{k} \beta_{2j} X_{ji} + u_{2i} = x_i'\beta_2 + u_{2i}$$

where x_{hi} and x_{hj} are the *i*th and *j*th observations, respectively, on x_h. The errors, u_{1i} and u_{2i}, are assumed to be distributed normally and independently with mean zero and constant variance. The variance for the first regime is σ_1^2, and the variance for the second regime is σ_2^2. If $\sigma_1^2 \neq \sigma_2^2$ and $\beta_1 \neq \beta_2$, the regression system given previously is thought to be switching between the two regimes.

The problem is to estimate β_1, β_2, σ_1, and σ_2 without knowing *a priori* which of the n values of the dependent variable, y, was generated by which regime. If it is known *a priori* which observations belong to which regime, a simple Chow test can be used to test $\sigma_1^2 = \sigma_2^2$ and $\beta_1 = \beta_2$.

Using Goldfeld and Quandt's D-method for switching regression, you can solve this problem. Assume that observations exist on some exogenous variables $z_{1i}, z_{2i}, \ldots, z_{pi}$, where z determines whether the *i*th observation is generated from one equation or the other. The equations are given as follows:

$$y_i = x_i'\beta_1 + u_{1i} \quad \text{if} \sum_{j=1}^{p} \pi_j z_{ji} \leq 0$$
$$y_i = x_i'\beta_2 + u_{2i} \quad \text{if} \sum_{j=1}^{p} \pi_j z_{ji} > 0$$

where π_j are unknown coefficients to be estimated. Define $d(z_i)$ as a continuous approximation to a step function. Replacing the unit step function with a continuous approximation by using the cumulative normal integral enables a more practical method that produces consistent estimates.

$$d(z_i) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\sum \pi_j z_{ji}} exp\left[-\frac{1}{2}\frac{\xi^2}{\sigma^2}\right] d\xi$$

D is the n dimensional diagonal matrix consisting of $d(z_i)$:

$$\mathbf{D} = \begin{bmatrix} d(z_1) & 0 & 0 & 0 \\ 0 & d(z_2) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & d(z_n) \end{bmatrix}$$

The parameters to estimate are now the k β_1's, the k β_2's, σ_1^2, σ_2^2, p π's, and the σ introduced in the $d(z_i)$ equation. The σ can be considered as given *a priori*, or it can be estimated, in which case, the estimated magnitude provides an estimate of the success in discriminating between the two regimes (Goldfeld and Quandt 1976). Given the preceding equations, the model can be written as:

$$Y = (\mathbf{I} - \mathbf{D})\mathbf{X}\beta_1 + \mathbf{D}\mathbf{X}\beta_2 + W$$

where $W = (\mathbf{I} - \mathbf{D})U_1 + \mathbf{D}U_2$, and W is a vector of unobservable and heteroscedastic error terms. The covariance matrix of W is denoted by $\mathbf{\Omega}$, where $\mathbf{\Omega} = (\mathbf{I} - \mathbf{D})^2\sigma_1^2 + \mathbf{D}^2\sigma_2^2$. The maximum likelihood parameter estimates maximize the following log-likelihood function.

$$\begin{aligned}\log L &= -\frac{n}{2}\log 2\pi - \frac{1}{2}\log \mid \mathbf{\Omega} \mid - \\ &\quad \frac{1}{2} * \left[[Y - (\mathbf{I} - \mathbf{D})\mathbf{X}\beta_1 - \mathbf{D}\mathbf{X}\beta_2]' \, \mathbf{\Omega}^{-1} \, [Y - (\mathbf{I} - \mathbf{D})\mathbf{X}\beta_1 - \mathbf{D}\mathbf{X}\beta_2]\right]\end{aligned}$$

As an example, you now can use this switching regression likelihood to develop a model of housing starts as a function of changes in mortgage interest rates. The data for this example are from the U.S. Census Bureau and cover the period from January 1973 to March 1999. The hypothesis is that there are different coefficients on your model based on whether the interest rates are going up or down.

So the model for z_i is

$$z_i = p * (\mathrm{rate}_i - \mathrm{rate}_{i-1})$$

where rate_i is the mortgage interest rate at time i and p is a scale parameter to be estimated.

The regression model is

$$\begin{aligned}\mathrm{starts}_i &= \mathrm{intercept}_1 + \mathrm{ar1} * \mathrm{starts}_{i-1} + \mathrm{djf1} * \mathrm{decjanfeb} \qquad z_i < 0 \\ \mathrm{starts}_i &= \mathrm{intercept}_2 + \mathrm{ar2} * \mathrm{starts}_{i-1} + \mathrm{djf2} * \mathrm{decjanfeb} \qquad z_i >= 0\end{aligned}$$

where starts_i is the number of housing starts at month i and $\mathrm{decjanfeb}$ is a dummy variable that indicates that the current month is one of December, January, or February.

This model is written by using the following SAS statements:

```
title1 'Switching Regression Example';

proc model data=switch;
   parms sig1=10 sig2=10 int1 b11 b13 int2 b21 b23 p;
   bounds 0.0001 < sig1 sig2;

   decjanfeb = ( month(date) = 12 | month(date) <= 2 );

   a = p*dif(rate);       /* Upper bound of integral */
   d = probnorm(a);       /* Normal CDF as an approx of switch */

                          /* Regime 1 */
   y1 = int1 + zlag(starts)*b11 + decjanfeb *b13 ;
                          /* Regime 2 */
   y2 = int2 + zlag(starts)*b21 + decjanfeb *b23 ;
                          /* Composite regression equation */
   starts  = (1 - d)*y1 +  d*y2;

                         /* Resulting log-likelihood function */
   logL = (1/2)*( (log(2*3.1415)) +
        log( (sig1**2)*((1-d)**2)+(sig2**2)*(d**2) )
      + (resid.starts*( 1/( (sig1**2)*((1-d)**2)+
```

```
        (sig2**2)*(d**2) ) )*resid.starts) ) ;

    errormodel starts ~ general(logL);

    fit starts / method=marquardt converge=1.0e-5;

      /* Test for significant differences in the parms */
    test int1 = int2 ,/ lm;
    test b11 = b21 ,/ lm;
    test b13 = b23 ,/ lm;
    test sig1 = sig2 ,/ lm;

run;
```

Four TEST statements are added to test the hypothesis that the parameters are the same in both regimes. The parameter estimates and ANOVA table from this run are shown in Output 19.13.1.

Output 19.13.1 Parameter Estimates from the Switching Regression

Switching Regression Example

The MODEL Procedure

Nonlinear Liklhood Summary of Residual Errors

Equation	DF Model	DF Error	SSE	MSE	Root MSE	R-Square	Adj R-Sq	Label
starts	9	304	85878.0	282.5	16.8075	0.7806	0.7748	Housing Starts

Nonlinear Liklhood Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
sig1	15.47484	0.9476	16.33	<.0001
sig2	19.77808	1.2710	15.56	<.0001
int1	32.82221	5.9083	5.56	<.0001
b11	0.73952	0.0444	16.64	<.0001
b13	-15.4556	3.1912	-4.84	<.0001
int2	42.73348	6.8159	6.27	<.0001
b21	0.734117	0.0478	15.37	<.0001
b23	-22.5184	4.2985	-5.24	<.0001
p	25.94712	8.5205	3.05	0.0025

The test results shown in Output 19.13.2 suggest that the variance of the housing starts, SIG1 and SIG2, are significantly different in the two regimes. The tests also show a significant difference in the AR term on the housing starts.

Output 19.13.2 Test Results for Switching Regression

Test Results

Test	Type	Statistic	Pr > ChiSq	Label
Test0	L.M.	1.00	0.3185	int1 = int2
Test1	L.M.	15636	<.0001	b11 = b21
Test2	L.M.	1.45	0.2280	b13 = b23
Test3	L.M.	4.39	0.0361	sig1 = sig2

Example 19.14: Simulating from a Mixture of Distributions

This example illustrates how to perform a multivariate simulation by using models that have different error distributions. Three models are used. The first model has *t* distributed errors. The second model is a GARCH(1,1) model with normally distributed errors. The third model has a noncentral Cauchy distribution.

The following SAS statements generate the data for this example. The *t* and the CAUCHY data sets use a common seed so that those two series are correlated.

```
/* set distribution parameters */
%let df = 7.5;
%let sig1 = .5;
%let var2 = 2.5;

data t;
   format date monyy.;
   do date='1jun2001'd  to '1nov2002'd;
              /* t-distribution with df,sig1 */
      t = .05 * date + 5000  + &sig1*tinv(ranuni(1234),&df);
      output;
   end;
run;

data normal;
   format date monyy.;
   le = &var2;
   lv = &var2;
   do date='1jun2001'd  to '1nov2002'd;
              /* Normal with GARCH error structure */
      v = 0.0001 + 0.2 * le**2 + .75 * lv;
      e = sqrt( v) * rannor(12345) ;
      normal = 25 + e;
      le = e;
      lv = v;
      output;
   end;
run;

data cauchy;
   format date monyy.;
   PI = 3.1415926;
   do date='1jun2001'd  to '1nov2002'd;
      cauchy = -4 + tan((ranuni(1234) - 0.5) * PI);
      output;
   end;
run;
```

Since the multivariate joint likelihood is unknown, the models must be estimated separately. The residuals for each model are saved by using the OUT= option. Also, each model is saved by using the OUTMODEL= option. The ID statement is used to provide a variable in the residual data set to merge by. The XLAG function is used to model the GARCH(1,1) process. The XLAG function returns the lag of the first argument if it is nonmissing, otherwise it returns the second argument.

```
title1 't-distributed Errors Example';

proc model data=t outmod=tModel;
   parms df 10 vt 4;
   t = a * date + c;
   errormodel t ~ t( vt, df );
   fit t / out=tresid;
   id date;
run;

title1 'GARCH-distributed Errors Example';

proc model data=normal outmodel=normalModel;
   normal = b0 ;
   h.normal = arch0 + arch1 * xlag(resid.normal **2 , mse.normal)
            + GARCH1 * xlag(h.normal, mse.normal);

   fit normal /fiml out=nresid;
   id date;
run;

title1 'Cauchy-distributed Errors Example';

proc model data=cauchy outmod=cauchyModel;
   parms nc = 1;
   /* nc is noncentrality parm to Cauchy dist */
   cauchy = nc;
   obj = log(1+resid.cauchy**2 * 3.1415926);
   errormodel cauchy ~ general(obj) cdf=cauchy(nc);

   fit cauchy / out=cresid;
   id date;
run;
```

The simulation requires a covariance matrix created from normal residuals. The following DATA step statements use the inverse CDFs of the *t* and Cauchy distributions to convert the residuals to the normal distribution. The CORR procedure is used to create a correlation matrix that uses the converted residuals.

```
/* Merge and normalize the 3 residual data sets */
data c; merge tresid nresid cresid; by date;
   t = probit(cdf("T", t/sqrt(0.2789), 16.58 ));
   cauchy = probit(cdf("CAUCHY", cauchy, -4.0623));
run;

proc corr data=c out=s;
   var t normal cauchy;
run;
```

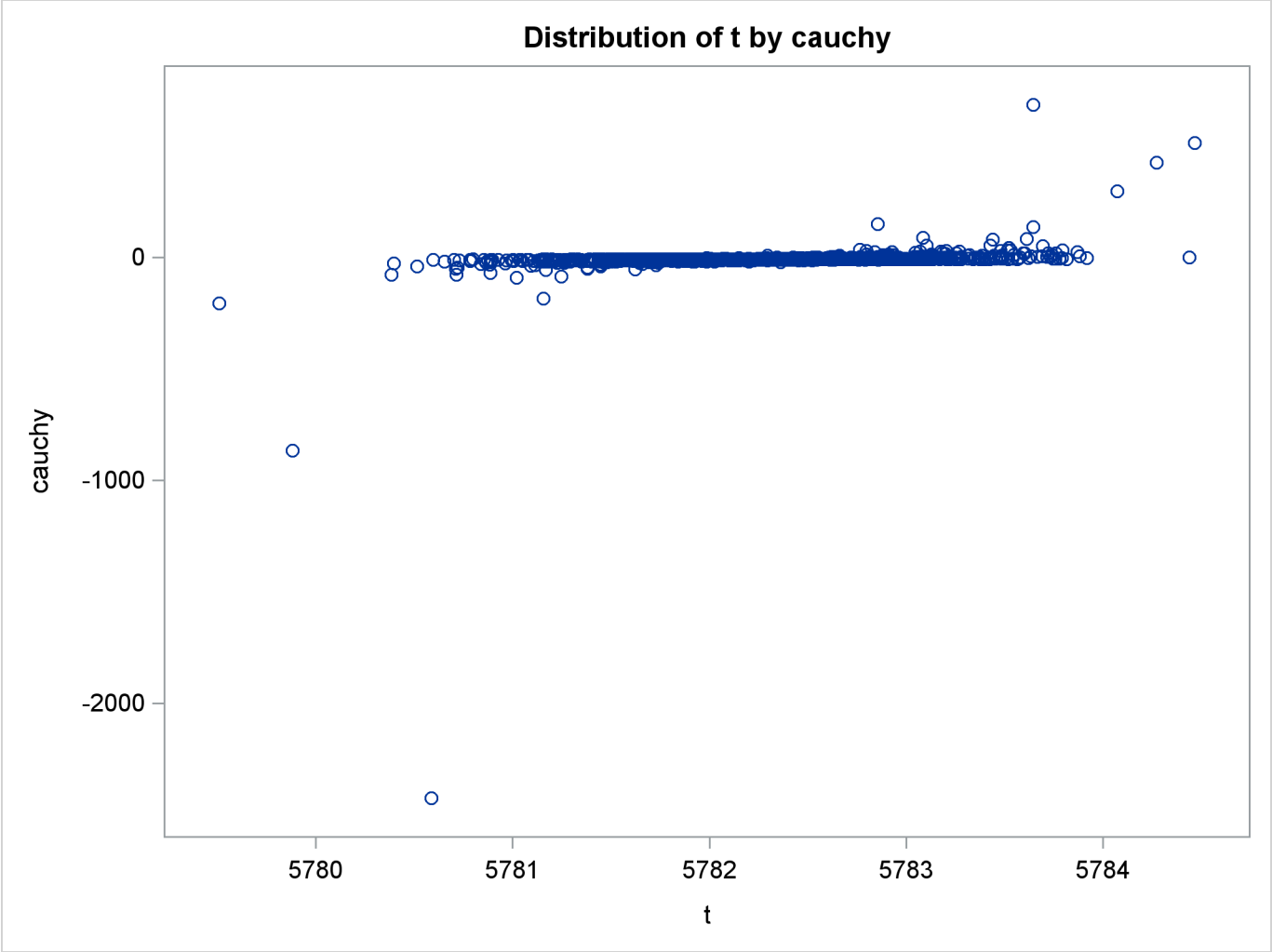
Now the models can be simulated together by using the MODEL procedure SOLVE statement. The data set created by the CORR procedure is used as the correlation matrix.

```
title1 'Simulating Equations with Different Error Distributions';

/* Create one observation driver data set */
data sim; merge t normal cauchy; by date;
data sim; set sim(firstobs = 519 );

proc model data=sim model=( tModel normalModel cauchyModel );
   errormodel t ~ t( vt, df );
   errormodel cauchy ~ cauchy(nc);
   solve t cauchy normal / random=2000 seed=1962 out=monte
                           sdata=s(where=(_type_="CORR"));
run;
```

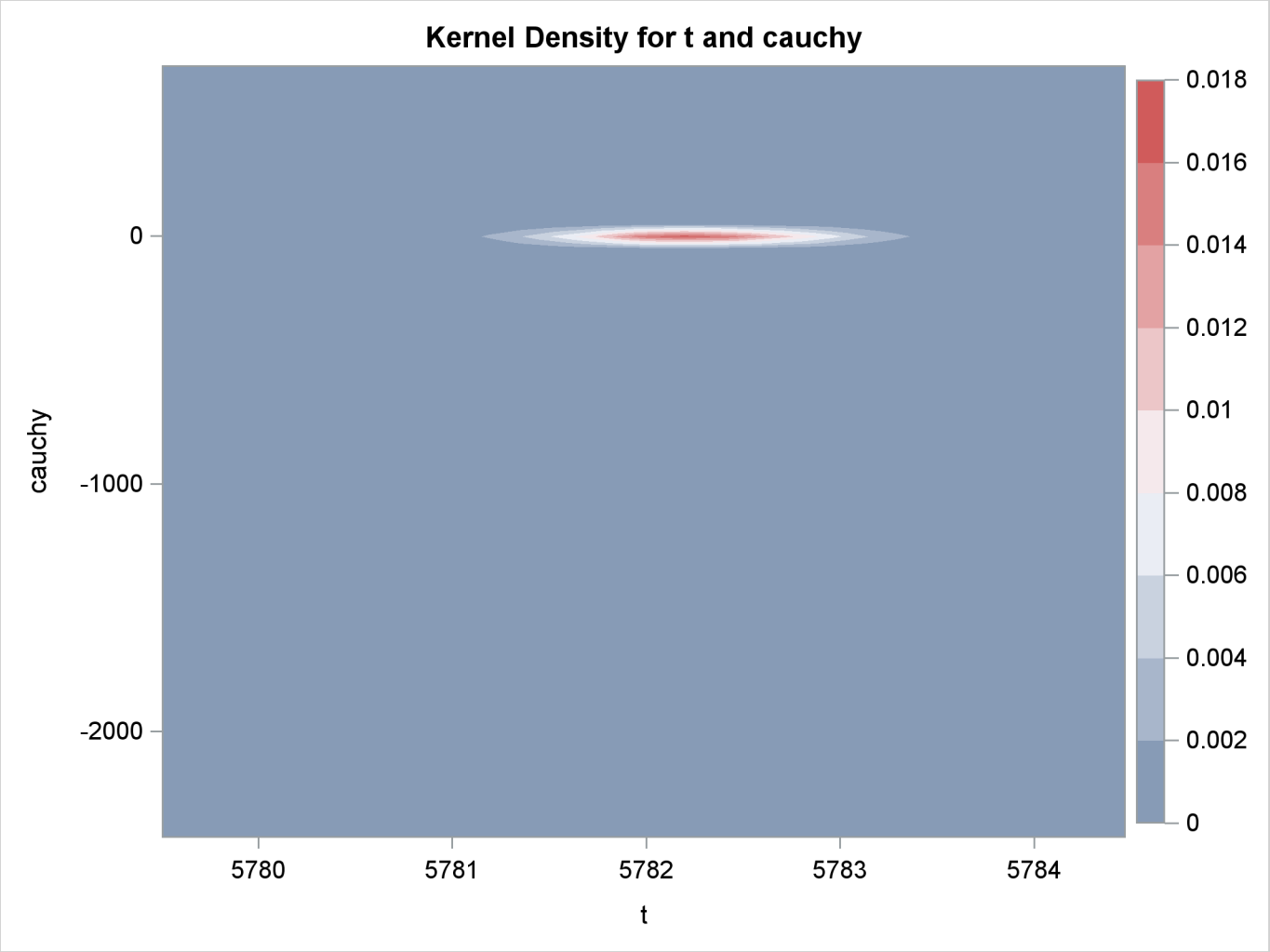
An estimation of the joint density of the *t* and Cauchy distribution is created by using the KDE procedure. Bounds are placed on the Cauchy dimension because of its fat tail behavior. The joint PDF is shown in Output 19.14.1.

```
title "T and Cauchy Distribution";

proc kde data=monte;
   univar t       / out=t_dens;
   univar cauchy  / out=cauchy_dens;
   bivar t cauchy / out=density
                    plots=all;
run;
```

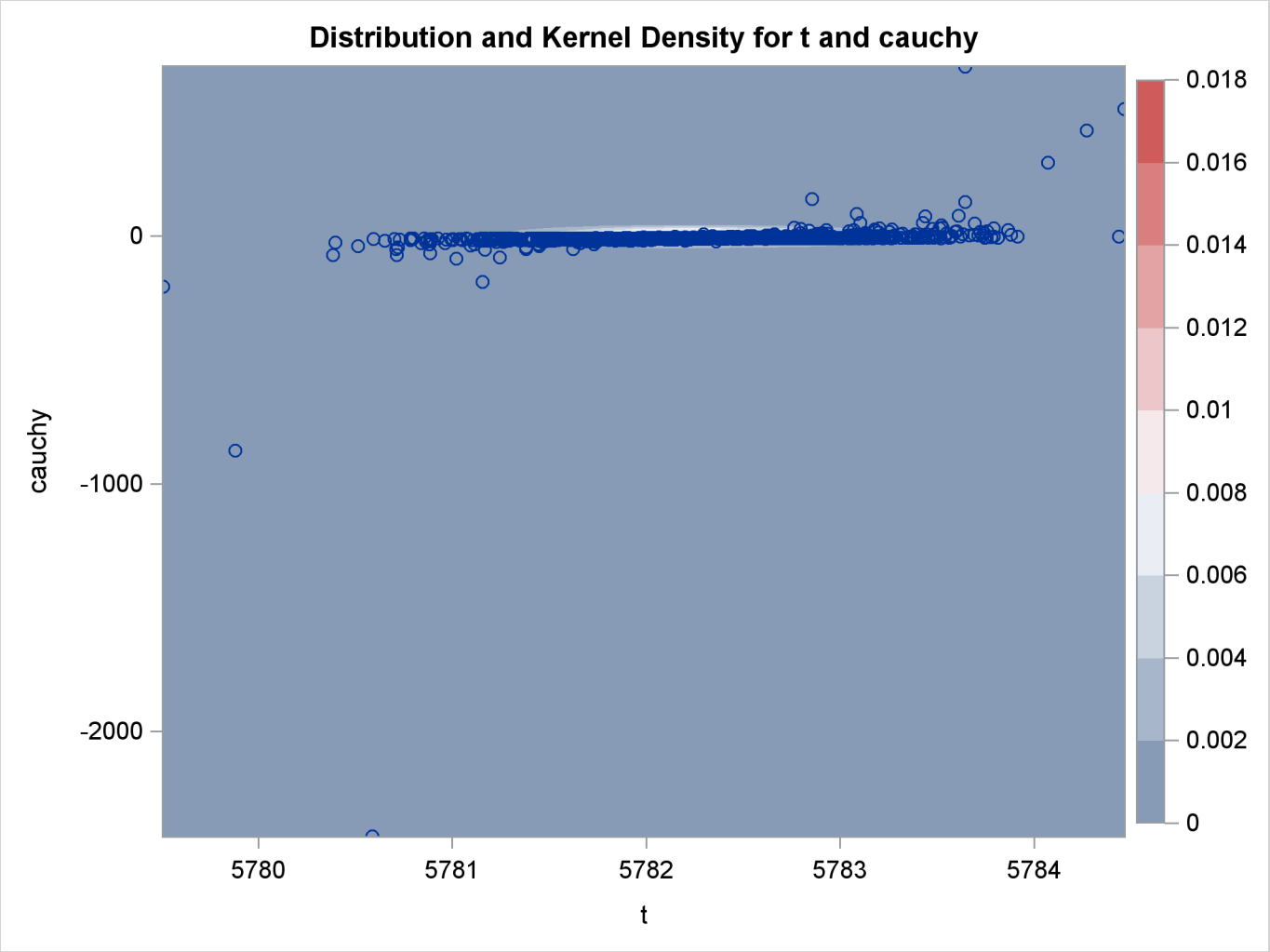
Output 19.14.1 Bivariate Density of *t* and Cauchy, Distribution of *t* by Cauchy



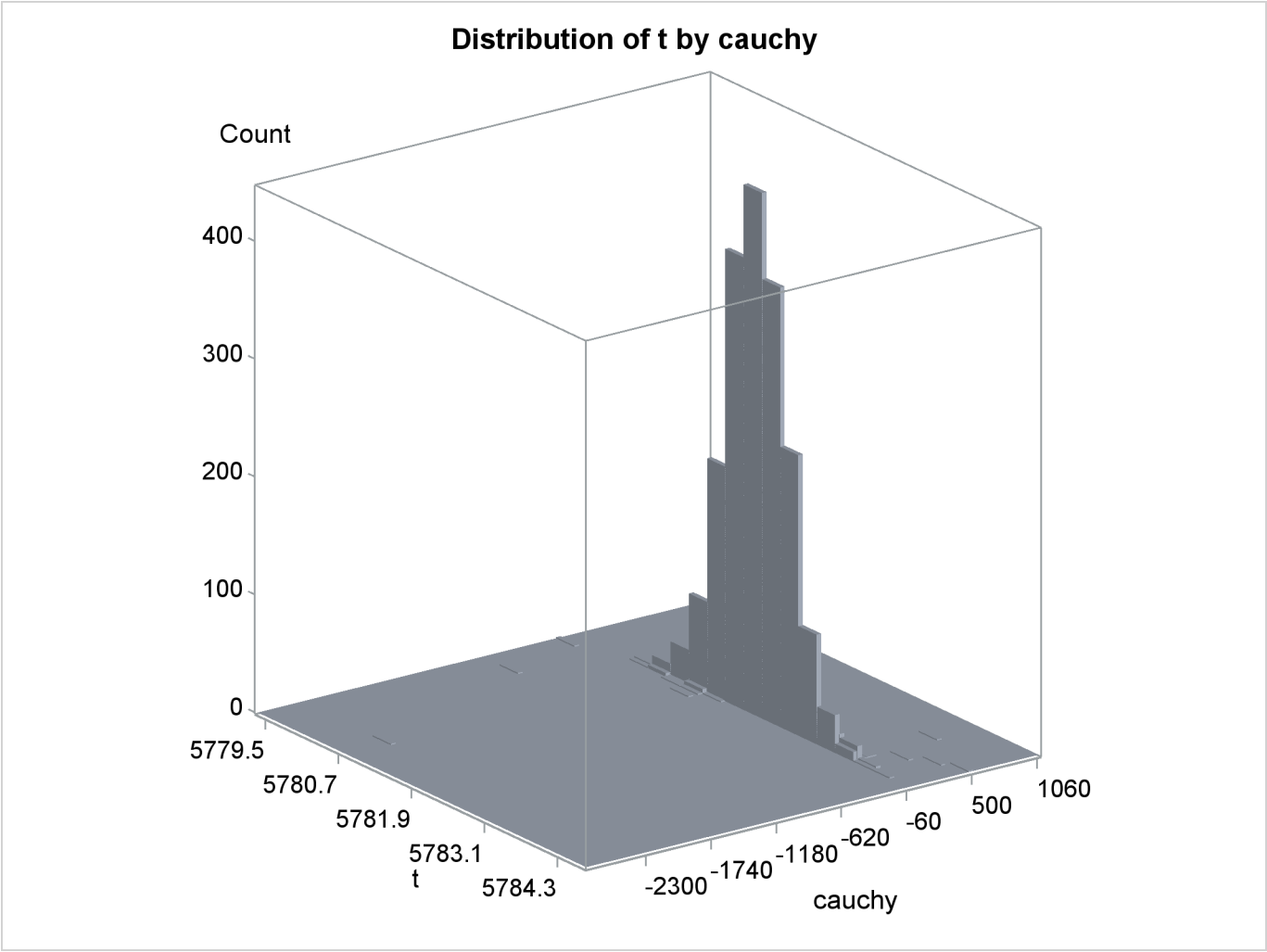
Output 19.14.2 Bivariate Density of *t* and Cauchy, Kernel Density for *t* and Cauchy



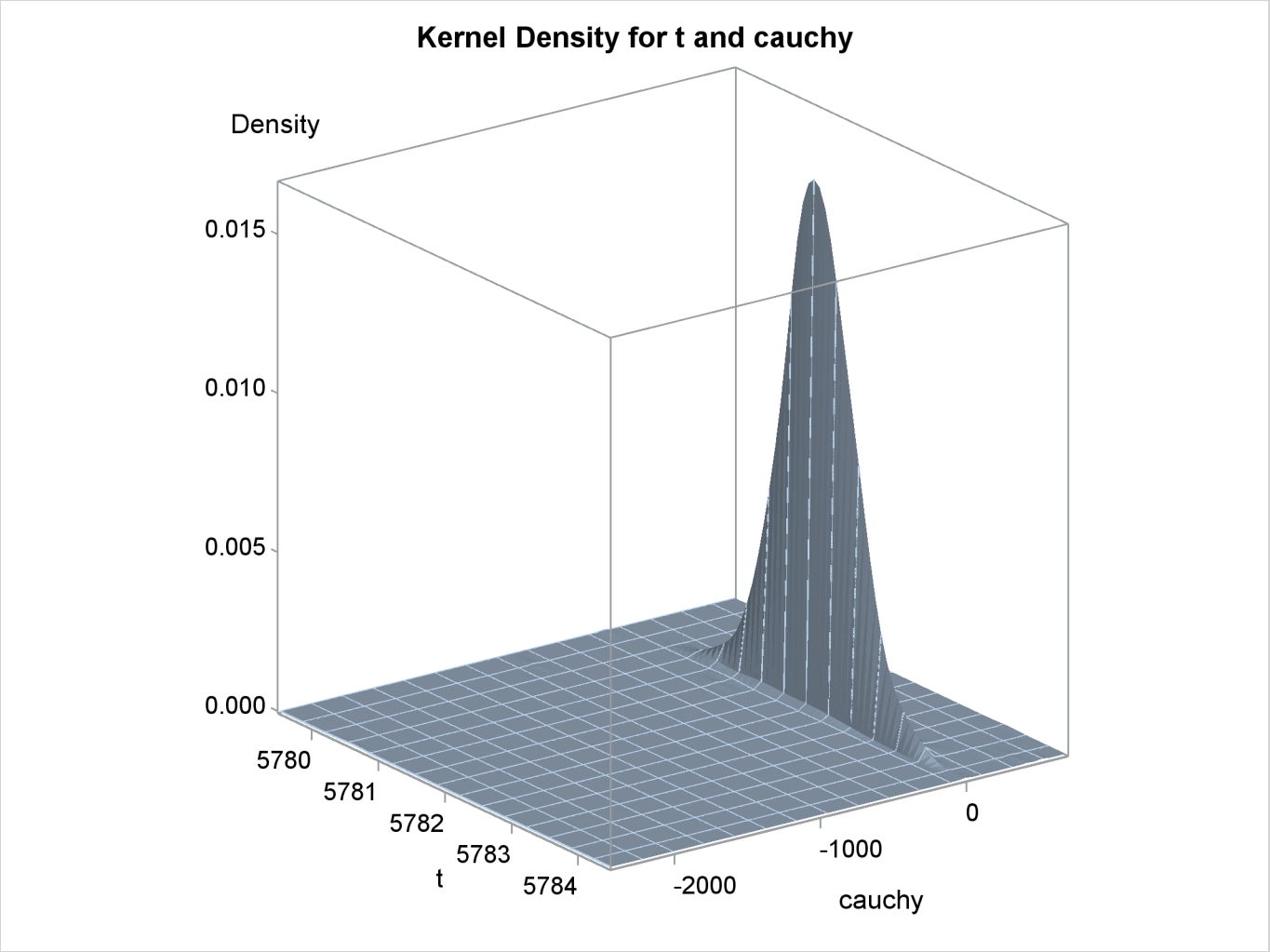
Output 19.14.3 Bivariate Density of *t* and Cauchy, Distribution and Kernel Density for *t* and Cauchy



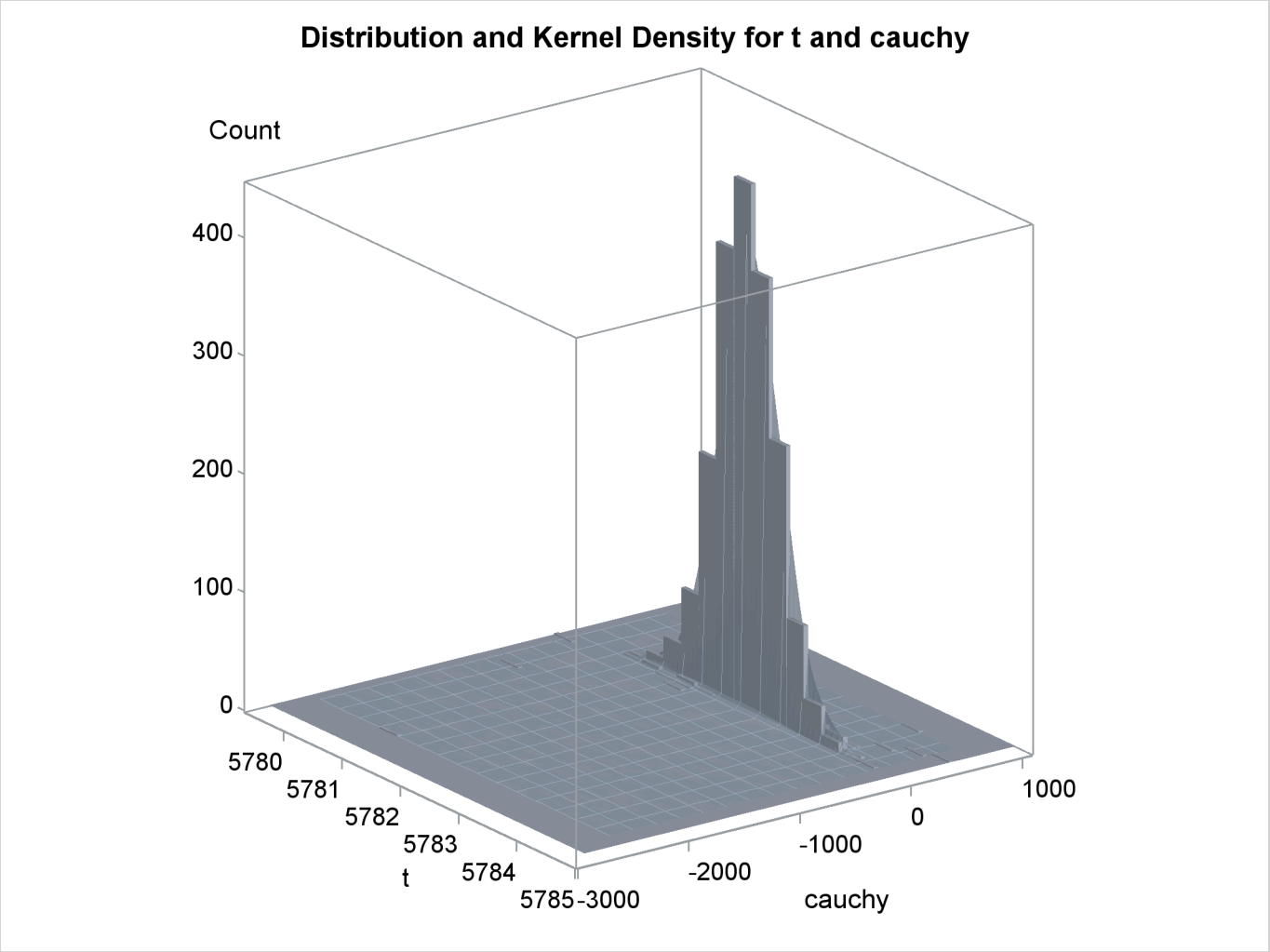
Output 19.14.4 Bivariate Density of *t* and Cauchy, Distribution of *t* by Cauchy



Output 19.14.5 Bivariate Density of *t* and Cauchy, Kernel Density for *t* and Cauchy



Output 19.14.6 Bivariate Density of *t* and Cauchy, Distribution and Kernel Density for *t* and Cauchy



Example 19.15: Simulated Method of Moments—Simple Linear Regression

This example illustrates how to use SMM to estimate a simple linear regression model for the following process:

$$y = a + bx + \epsilon,\ \epsilon \sim iid\ N(0, s^2)$$

In the following SAS statements, $ysim$ is simulated, and the first moment and the second moment of $ysim$ are compared with those of the observed endogenous variable y.

```
title "Simple regression model";

data regdata;
   do i=1 to 500;
      x = rannor( 1013 );
      Y = 2 + 1.5 * x + 1.5 * rannor( 1013 );
      output;
   end;
run;
```

```
proc model data=regdata;
   parms a b s;
   instrument x;

   ysim = (a+b*x) + s * rannor( 8003 );
   y = ysim;
   eq.ysq = y*y - ysim*ysim;

   fit y ysq / gmm ndraw;
   bound s > 0;
run;
```

Alternatively, the MOMENT statement can be used to specify the moments using the following syntax:

```
proc model data=regdata;
   parms a b s;
   instrument x;

   ysim = (a+b*x) + s * rannor( 8003 );
   y = ysim;
   moment y = (2);

   fit y / gmm ndraw;
   bound s > 0;
run;
```

The output of the MODEL procedure is shown in Output 19.15.1:

Output 19.15.1 PROC MODEL Output

Simple regression model

The MODEL Procedure

Model Summary	
Model Variables	1
Parameters	3
Equations	2
Number of Statements	4

Model Variables	Y
Parameters	a b s
Equations	ysq Y

The 2 Equations to Estimate	
Y =	F(a(1), b(x), s)
ysq =	F(a, b, s)
Instruments	1 x

Output 19.15.1 *continued*

Nonlinear GMM Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
a	2.065983	0.0657	31.45	<.0001
b	1.511075	0.0565	26.73	<.0001
s	1.483358	0.0498	29.78	<.0001

Example 19.16: Simulated Method of Moments—AR(1) Process

This example illustrates how to use SMM to estimate an AR(1) regression model for the following process:

$$
\begin{aligned}
y_t &= a + bx_t + u_t \\
u_t &= \alpha u_{t-1} + \epsilon_t \\
\epsilon_t &\sim iid\ N(0, s^2)
\end{aligned}
$$

In the following SAS statements, $ysim$ is simulated by using this model, and the endogenous variable y is set to be equal to $ysim$. The MOMENT statement creates two more moments for the estimation. One is the second moment, and the other is the first-order autocovariance. The NPREOBS=10 option instructs PROC MODEL to run the simulation 10 times before $ysim$ is compared to the first observation of y. Because the initial zlag(u) is zero, the first $ysim$ is $a + b * x + s *$ rannor(8003). Without the NPREOBS option, this $ysim$ is matched with the first observation of y. With NPREOBS, this $ysim$ and the next nine $ysim$ are thrown away, and the moment match starts with the eleventh $ysim$ with the first observation of y. This way, the initial values do not exert a large influence on the simulated endogenous variables.

```
%let nobs=500;
data ardata;
   lu =0;
   do i=-10 to &nobs;
      x = rannor( 1011 );
      e = rannor( 1011 );
      u = .6 * lu + 1.5 * e;
      Y = 2 + 1.5 * x + u;
      lu = u;
      if i > 0 then output;
   end;
run;

title1 'Simulated Method of Moments for AR(1) Process';

proc model data=ardata ;
   parms a b s 1 alpha .5;
   instrument x;

   u = alpha * zlag(u) + s * rannor( 8003 );
   ysim = a + b * x + u;
   y = ysim;
   moment y = (2) lag1(1);
```

```
   fit y / gmm npreobs=10 ndraw=10;
   bound s > 0, 1 > alpha > 0;
run;
```

The output of the MODEL procedure is shown in Output 19.16.1:

Output 19.16.1 PROC MODEL Output

Simulated Method of Moments for AR(1) Process

The MODEL Procedure

Model Summary	
Model Variables	1
Parameters	4
Equations	3
Number of Statements	8
Program Lag Length	1

Model Variables	Y
Parameters(Value)	a b s(1) alpha(0.5)
Equations	_moment_2 _moment_1 Y

The 3 Equations to Estimate	
_moment_2 =	F(a, b, s, alpha)
_moment_1 =	F(a, b, s, alpha)
Y =	F(a(1), b(x), s, alpha)
Instruments	1 x

Nonlinear GMM Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
a	1.632798	0.1038	15.73	<.0001
b	1.513197	0.0698	21.67	<.0001
s	1.427888	0.0984	14.52	<.0001
alpha	0.543985	0.0809	6.72	<.0001

Example 19.17: Simulated Method of Moments—Stochastic Volatility Model

This example illustrates how to use SMM to estimate a stochastic volatility model as in Andersen and Sorensen (1996):

$$\begin{aligned} y_t &= \sigma_t z_t \\ log(\sigma_t^2) &= a + b\, log(\sigma_{t-1}^2) + s u_t \\ (z_t, u_t) &\sim iid\ N(0, I_2) \end{aligned}$$

This model is widely used in modeling the return process of stock prices and foreign exchange rates. This is called the stochastic volatility model because the volatility is stochastic as the random variable u_t appears in the volatility equation. The following SAS statements use three moments: absolute value, the second-order

moment, and absolute value of the first-order autoregressive moment. Note the ADJSMMV option in the FIT statement to request the SMM covariance adjustment for the parameter estimates. Although these moments have closed form solution as shown by Andersen and Sorensen (1996), the simulation approach significantly simplifies the moment conditions.

```
%let nobs=1000;
data _tmpdata;
   a = -0.736; b=0.9; s=0.363;
   ll=sqrt( exp(a/(1-b)));;
   do i=-10 to &nobs;
      u = rannor( 101 );
      z = rannor( 101 );
      lnssq = a+b*log(ll**2) +s*u;
      st = sqrt(exp(lnssq));
      ll = st;
      y = st * z;
      if i > 0 then output;
   end;
run;

title1 'Simulated Method of Moments for Stochastic Volatility Model';

proc model data=_tmpdata ;
   parms a b .5 s 1;
   instrument / intonly;

   u = rannor( 8801 );
   z = rannor( 9701 );
   lsigmasq = xlag(sigmasq,exp(a));
   lnsigmasq = a + b * log(lsigmasq) + s * u;
   sigmasq = exp( lnsigmasq );

   ysim = sqrt(sigmasq) * z;
   eq.m1 = abs(y) - abs(ysim);
   eq.m2 = y**2 - ysim**2;
   eq.m5 = abs(y*lag(y))-abs(ysim*lag(ysim));

   fit m1 m2 m5 / gmm npreobs=10 ndraw=10 adjsmmv;
   bound s > 0, 1 > b > 0;
run;
```

The output of the MODEL procedure is shown in Output 19.17.1.

Output 19.17.1 PROC MODEL Output

Simulated Method of Moments for Stochastic Volatility Model

The MODEL Procedure

Model Summary	
Parameters	3
Equations	3
Number of Statements	10
Program Lag Length	1

Output 19.17.1 *continued*

Parameters(Value)	a b(0.5) s(1)
Equations	m1 m2 m5

The 3 Equations to Estimate	
m1 =	F(a, b, s)
m2 =	F(a, b, s)
m5 =	F(a, b, s)
Instruments	1

Nonlinear GMM Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
a	-2.2299	1.1357	-1.96	0.0499
b	0.695469	0.1554	4.47	<.0001
s	0.747779	0.1648	4.54	<.0001

Example 19.18: Duration Data Model with Unobserved Heterogeneity

All of the previous three models actually have closed-form moment conditions, so the simulation approach is not necessarily required for the estimation. This example illustrates how to use SMM to estimate a model for which there is no closed-form solution for the moments and thus the traditional GMM method does not apply. The model is the duration data model with unobserved heterogeneity in Gourieroux and Monfort (1993):

$$\begin{aligned} y_i &= -exp(-bx_i - \sigma u_i)log(v_i) \\ u_i &\sim N(0,1) \quad v_i \sim U_{[0,1]} \end{aligned}$$

The SAS statements are:

```
title1 'SMM for Duration Model with Unobserved Heterogeneity';

%let nobs=1000;
data durationdata;
   b=0.9; s=0.5;
   do i=1 to &nobs;
      u = rannor( 1011 );
      v = ranuni( 1011 );
      x = 2 * ranuni( 1011 );
      y = -exp(-b * x + s * u) * log(v);
      output;
   end;
run;

proc model data=durationdata;
   parms b .5 s 1;
   instrument x;
```

```
   u = rannor( 1011 );
   v = ranuni( 1011 );
   y = -exp(-b * x + s * u) * log(v);

   moment y = (2 3 4);
   fit y / gmm ndraw=10 ;* maxiter=500;
   bound s > 0, b > 0;
run;
```

The output of the MODEL procedure is shown in Output 19.18.1.

Output 19.18.1 PROC MODEL Output

SMM for Duration Model with Unobserved Heterogeneity

The MODEL Procedure

Model Summary	
Model Variables	1
Parameters	2
Equations	4
Number of Statements	9

Model Variables	y
Parameters(Value)	b(0.5) s(1)
Equations	_moment_3 _moment_2 _moment_1 y

The 4 Equations to Estimate	
_moment_3 =	F(b, s)
_moment_2 =	F(b, s)
_moment_1 =	F(b, s)
y =	F(b, s)
Instruments	1 x

Nonlinear GMM Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
b	0.92983	0.0331	28.08	<.0001
s	0.341825	0.0608	5.62	<.0001

Example 19.19: EMM Estimation of a Stochastic Volatility Model

The efficient method of moments (EMM) (Bansal et al. 1993, 1995; Gallant and Tauchen 2001), can be considered a variant of SMM. The idea is to match the efficiency of the maximum likelihood (ML) estimation with the flexibility of the SMM procedure. ML itself can be interpreted as a method of moments procedure, where the *score vector*, the vector of derivatives of the log-likelihood function with respect to the parameters, provides the exactly identifying moment conditions. EMM employs an auxiliary (or pseudo) model that closely matches the true model. The score vector of the auxiliary model provides the moment conditions in the SMM step.

This example uses the SMM feature of PROC MODEL to estimate the simple stochastic volatility (SV) model of Example 19.17 with the EMM method.

Suppose that your data are the time series $\{y_1, y_2, \ldots, y_n\}$, and the model that you want to estimate, or the structural model, is characterized by the vector of parameters $\boldsymbol{\theta}$. For the SV model, $\boldsymbol{\theta}$ is given by (a, b, s).

The first step of the EMM method is to fit the data with an auxiliary model (or score generator) that has transition density $f(y_t|Y_{t-1}, \boldsymbol{\eta})$, parameterized by the pseudo parameter $\boldsymbol{\eta}$, where $Y_{t-1} = \{y_{t-1}, \ldots, y_1\}$. The auxiliary model must approximate the true data-generating process as closely as possible and be such that ML estimation is feasible.

The only identification requirement is that the dimension of the pseudo parameter $\boldsymbol{\eta}$ be greater than or equal to that of the structural parameter $\boldsymbol{\theta}$.

Andersen, Chung, and Sorensen (1999) showed that the GARCH(1,1) is an appropriate auxiliary model that leads to a good performance of the EMM estimator for the SV model.

The analytical expression for the GARCH(1,1) model with mean zero is

$$\begin{aligned} y_t &= \sigma_t z_t \\ \sigma_t^2 &= \omega + \alpha y_{t-1} + \beta \sigma_{t-1}^2 \end{aligned}$$

The pseudo parameter vector $\boldsymbol{\eta}$ is given by (ω, α, β).

One advantage of such a class of models is that the conditional density of y_t is Gaussian—that is,

$$f(y_t|Y_{t-1}, \boldsymbol{\eta}) \propto \frac{1}{\sigma_t} \exp\left(-\frac{y_t^2}{2\sigma_t^2}\right)$$

Therefore the score vector can easily be computed analytically.

The AUTOREG procedure provides the ML estimates, $\hat{\boldsymbol{\eta}}_n$. The estimates are stored in the garchest data set.

```
title1 'Efficient Method of Moments for Stochastic Volatility Model';

/* estimate GARCH(1,1) model */
proc autoreg data=svdata(keep=y)
             outest=garchest
             noprint covout;
   model y =  / noint garch=(q=1,p=1,type=nonneg);
run;
```

If the pseudo model is close enough to the structural model, in a suitable sense, Gallant and Long (1997) showed that a consistent estimator of the asymptotic covariance matrix of the sample pseudo-score vector can be obtained from the formula

$$\hat{\mathbf{V}}_n = \frac{1}{n} \sum_{t=1}^{n} s_f(Y_t, \hat{\boldsymbol{\eta}}_n) s_f(Y_t, \hat{\boldsymbol{\eta}}_n)'$$

where $s_f(Y_t, \hat{\boldsymbol{\eta}}_n) = (\partial/\partial \boldsymbol{\eta}_n) \log f(y_t|Y_{t-1}, \hat{\boldsymbol{\eta}}_n)$ denotes the score function of the auxiliary model computed at the ML estimates.

The ML estimates of the GARCH(1,1) model are used in the following SAS statements to compute the variance-covariance matrix $\hat{\mathbf{V}}_n$.

```
/* compute the V matrix */
data vvalues;
   set scores;

   array score{*} dlldw dllda dlldb;
   array v_t{*} v_t_1-v_t_6;
   array v{*} v_1-v_6;

   /* compute external product of score vector */
   do i=1 to 3;
      do j=i to 3;
         v_t{j*(j-1)/2 + i} = score{i}*score{j};
      end;
   end;

   /* average them over t */
   do s=1 to 6;
      v{s}+ v_t{s}/&nobs;
   end;
run;
```

The $\hat{\mathbf{V}}$ matrix must be formatted to be used with the VDATA= option of the MODEL procedure. See the section "VDATA= Input data set" on page 1224 for more information about the VDATA= data set.

```
/* Create a VDATA dataset acceptable to PROC MODEL */

/* Transpose the last obs in the dataset */
proc transpose data=vvalues(firstobs=&nobs keep=v_1-v_6)
               out=tempv;
run;

/* Add eq and inst labels */
data vhat;
   set tempv(drop=_name_);
   value = col1;
   drop col1;
   input _type_ $ eq_row $ eq_col $ inst_row $ inst_col $; *$;
   datalines;
      gmm m1 m1 1 1  /* intcpt is the only inst we use */
      gmm m1 m2 1 1
      gmm m2 m2 1 1
      gmm m1 m3 1 1
      gmm m2 m3 1 1
      gmm m3 m3 1 1
    ;
```

The last step of the EMM procedure is to estimate $\boldsymbol{\theta}$ by using SMM, where the moment conditions are given by the scores of the auxiliary model.

Given a fixed value of the parameter vector $\boldsymbol{\theta}$ and an arbitrarily large T, one can simulate a series $\{\hat{y}_1(\boldsymbol{\theta}), \hat{y}_2(\boldsymbol{\theta}), \ldots, \hat{y}_T(\boldsymbol{\theta})\}$ from the structural model. The EMM estimator is the value $\hat{\boldsymbol{\theta}}_n$ that minimizes the quantity

$$m_T(\boldsymbol{\theta}, \hat{\boldsymbol{\eta}}_n)' \hat{\mathbf{V}}_n^{-1} m_T(\boldsymbol{\theta}, \hat{\boldsymbol{\eta}}_n)$$

where

$$m_T(\boldsymbol{\theta}, \hat{\boldsymbol{\eta}}_n) = \frac{1}{T}\sum_{k=1}^{T} s_f(\hat{Y}_k(\boldsymbol{\theta}), \hat{\boldsymbol{\eta}}_n)$$

is the sample moment condition evaluated at the fixed estimated pseudo parameter $\hat{\boldsymbol{\eta}}_n$. Note that the target function depends on the parameter $\boldsymbol{\theta}$ only through the simulated series $\hat{y}_k$.

The following statements generate a data set that contains $T = 20,000$ replicates of the estimated pseudo parameter $\hat{\boldsymbol{\eta}}_n$ and that is then input to the MODEL procedure. The EMM estimates are found by using the SMM option of the FIT statement. The $\hat{\mathbf{V}}_n$ matrix computed above serves as weighting matrix by using the VDATA= option, and the scores of the GARCH(1,1) auxiliary model evaluated at the ML estimates are the moment conditions in the GMM step.

Since the number of structural parameters to estimate (3) is equal to the number of moment equations (3) times the number of instruments (1), the model is exactly identified and the objective function has value zero at the minimum.

For simplicity, the starting values are set to the true values of the parameters.

```
/* USE SMM TO FIND EMM ESTIMATES */

/* Generate dataset of length T */
data emm;
   set garchest(where=(_type_="PARM") rename=(_ah_0=w _ah_1=a _gh_1=b _mse_=mse)
               keep=_type_ _ah_0 _ah_1 _gh_1 _mse_);
   do i=1 to 20000;
      output;
   end;
   drop i;
run;

title2 'EMM estimates';
/* Find the EMM estimates */
proc model data=emm maxiter=1000 plot=none;
   parms aa -0.736 bb 0.9 ss 0.363;
   instruments _exog_ / intonly;

   /* Describe the structural model */
   u = rannor( 8801 );
   z = rannor( 9701 );
   lsigmasq = xlag(sigmasq,exp(aa));
   lnsigmasq = aa + bb * log(lsigmasq) + ss * u;
   sigmasq = exp( lnsigmasq );
   ysim = sqrt(sigmasq) * z;

   /* Compute scores of GARCH(1,1) */
   /* derivative of loglik wrt sigma-sq */
   ysim2 = ysim*ysim;
   lagvar = w + a*xlag(ysim2,mse) + xlag(lagvar,0)*b;
   var = lagvar + mse*b**_n_;
   dlldv = (-1 + ysim2/var)/var/2;

   /* arch 0 */
```

```
   dvdw = b*xlag(dvdw,0) + 1;
   dlldw = dlldv*dvdw;

   /* arch 1 */
   dvda = b*xlag(dvda,0) + xlag(ysim2,mse);
   dllda = dlldv*dvda;

   /* garch 1 */
   currdvdb = w + a*xlag(ysim2,mse);
   dvdb = - b*b*xlag2(dvdb,0) + 2*b*xlag(dvdb,0) + xlag(currdvdb,0);
   dlldb = dlldv*(dvdb + _n_*b**(_n_-1)*mse);

   /* Use scores of the GARCH model as moment conditions */
   eq.m1 = dlldw;
   eq.m2 = dllda;
   eq.m3 = dlldb;

   /* Fit scores using SMM and estimated Vhat */
   fit m1 m2 m3 / gmm npreobs=10 ndraw=1 /* smm options */
                  vdata=vhat /* use estimated Vhat */
                  kernel=(bart,0,) /* turn smoothing off */;
   bounds ss > 0, 0 < bb < 1;
quit;
```

The output of the MODEL procedure is shown in Output 19.19.1.

Output 19.19.1 PROC MODEL Output

Efficient Method of Moments for Stochastic Volatility Model
EMM estimates

The MODEL Procedure

Model Summary	
Parameters	3
Equations	3
Number of Statements	21

Parameters(Value)	aa(-0.736) bb(0.9) ss(0.363)
Equations	m1 m2 m3

The 3 Equations to Estimate	
m1 =	F(aa, bb, ss)
m2 =	F(aa, bb, ss)
m3 =	F(aa, bb, ss)
Instruments	1

Nonlinear GMM Parameter Estimates

Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
aa	-0.49702	0.0101	-49.40	<.0001
bb	0.930294	0.00137	677.59	<.0001
ss	0.316689	0.00395	80.14	<.0001

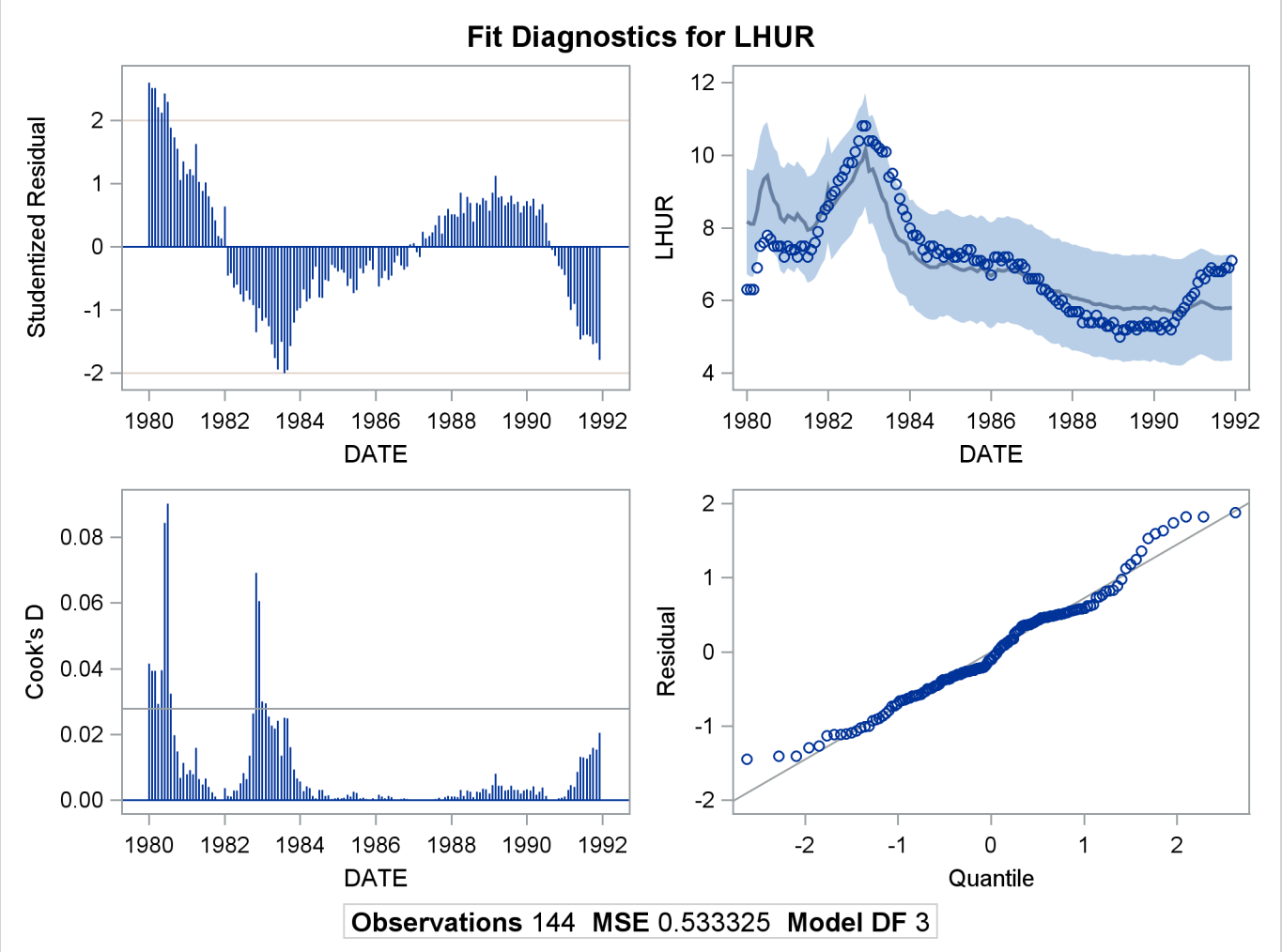
Example 19.20: Illustration of ODS Graphics

This example illustrates graphical output from PROC MODEL. This is a continuation of the section "Nonlinear Regression Analysis" on page 1071. For information about the graphics available in the MODEL procedure, see the section "ODS Graphics" on page 1231.

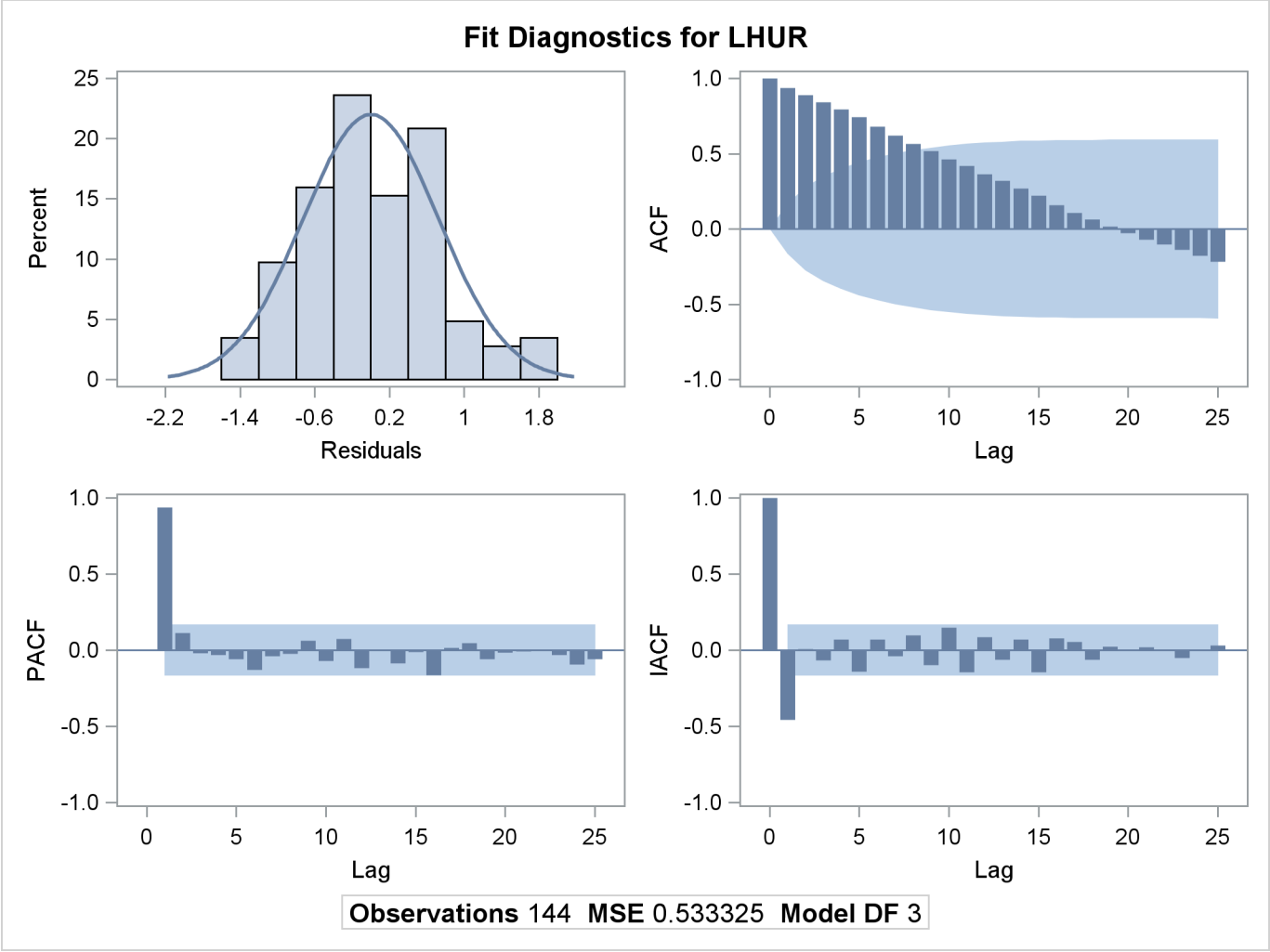
The following statements show how to generate ODS Graphics plots with the MODEL procedure. The plots are displayed in Output 19.20.1 and Output 19.20.2. Note that the variable date in the ID statement is used to define the horizontal tick mark values when appropriate.

```
title1 'Example of Graphical Output from PROC MODEL';

proc model data=sashelp.citimon;
   lhur = 1/(a * ip + b) + c;
   fit lhur;
   id date;
run;
```

Output 19.20.1 Diagnostics Plots



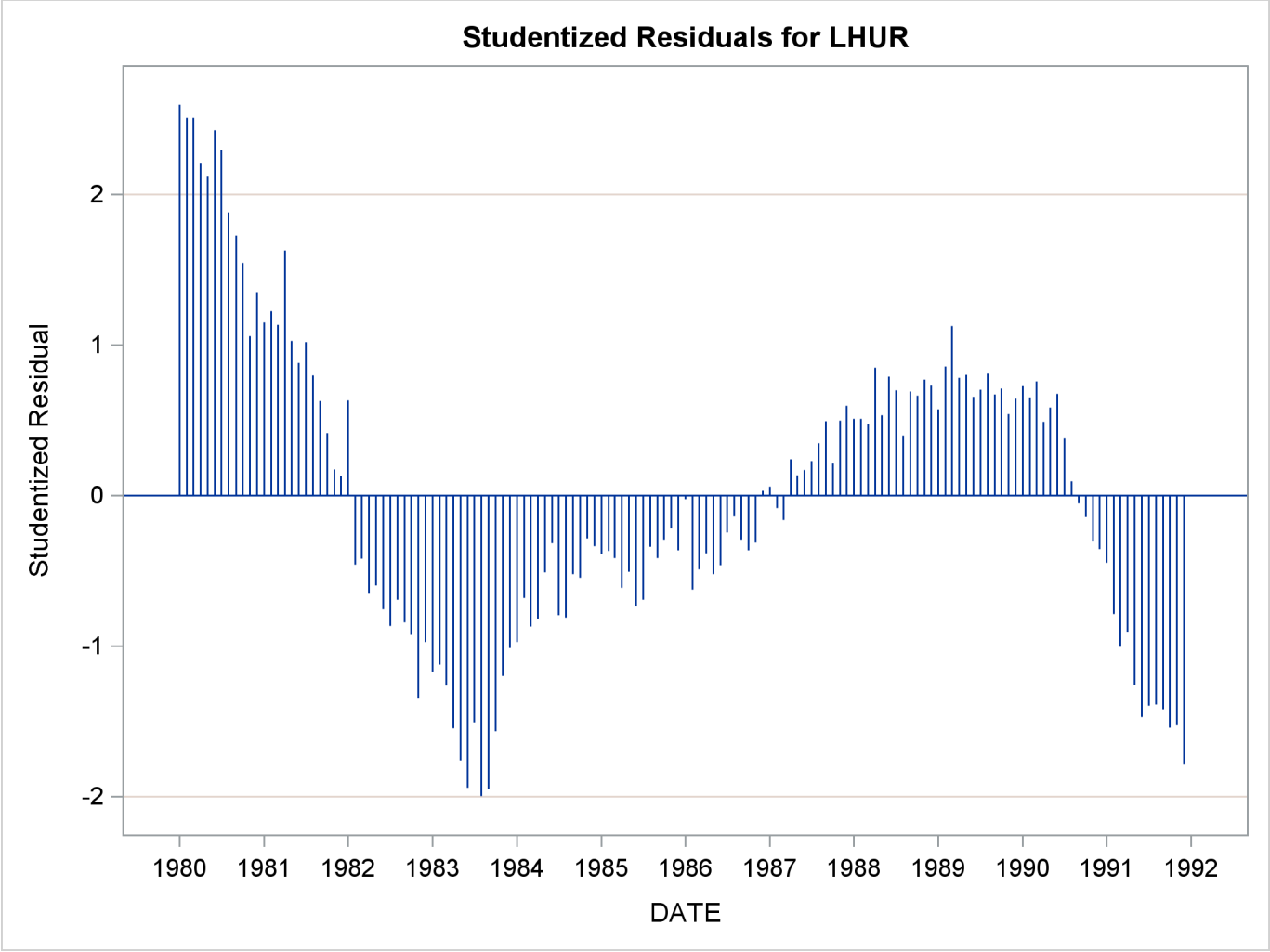
Output 19.20.2 Diagnostics Plots



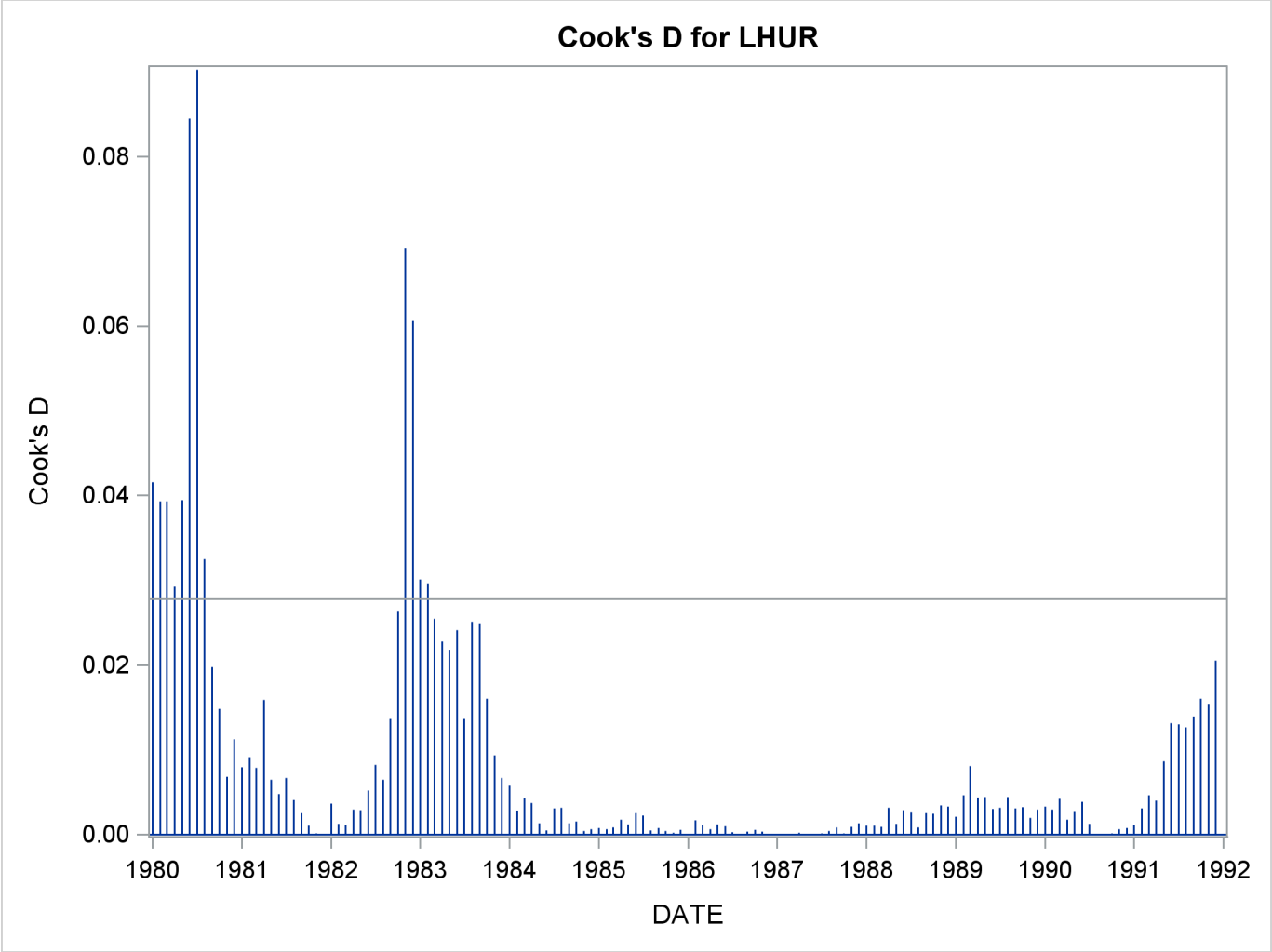
You can also obtain the plots in the diagnostics panel as separate graphs by specifying the PLOTS(UNPACK) option. These plots are displayed in Output 19.20.3 through Output 19.20.10.

```
title1 'Unpacked Graphical Output from PROC MODEL';

proc model data=sashelp.citimon plots(unpack);
   lhur = 1/(a * ip + b) + c;
   fit lhur;
   id date;
run;
```

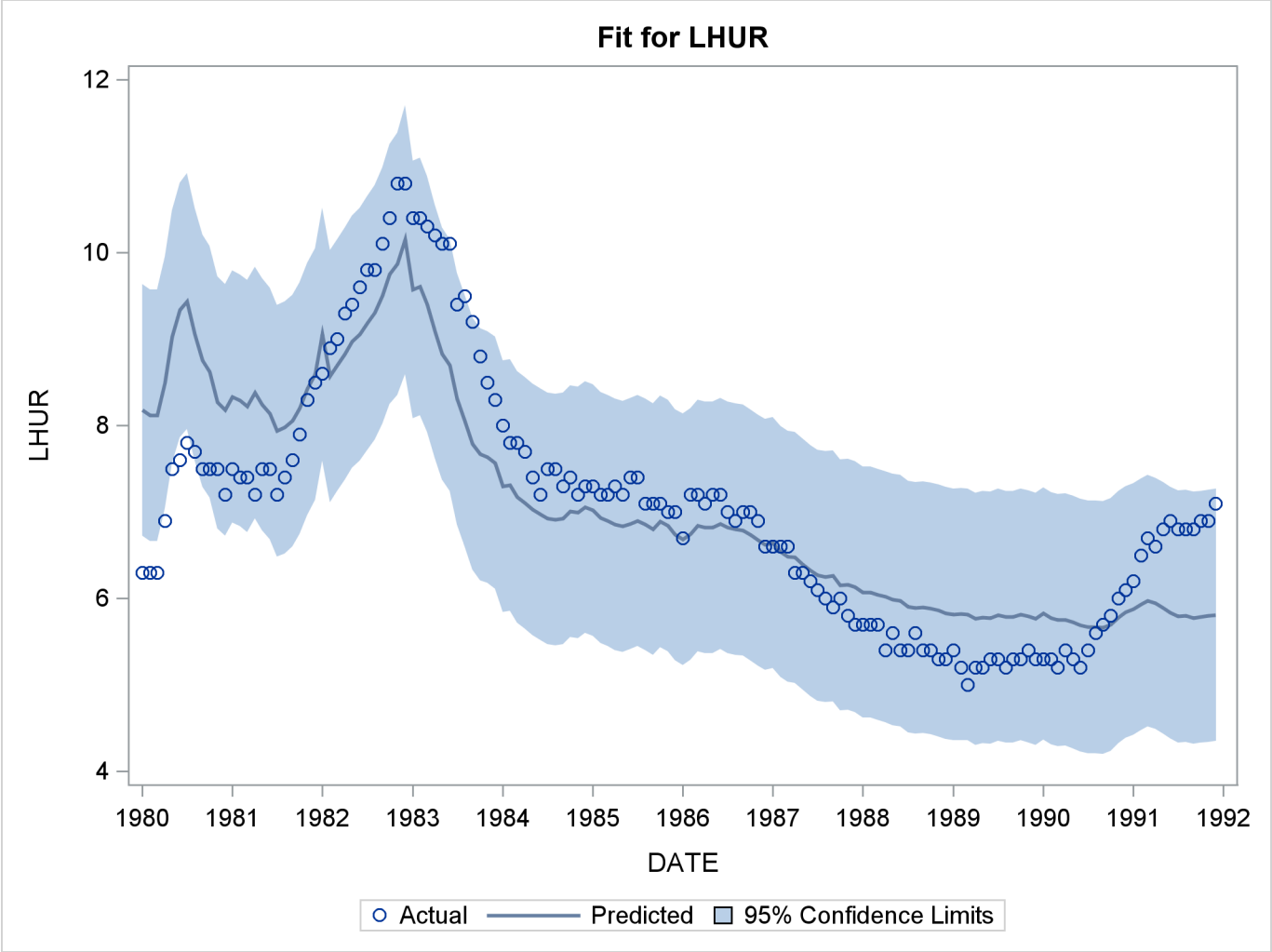
Output 19.20.3 Studentized Residuals Plot



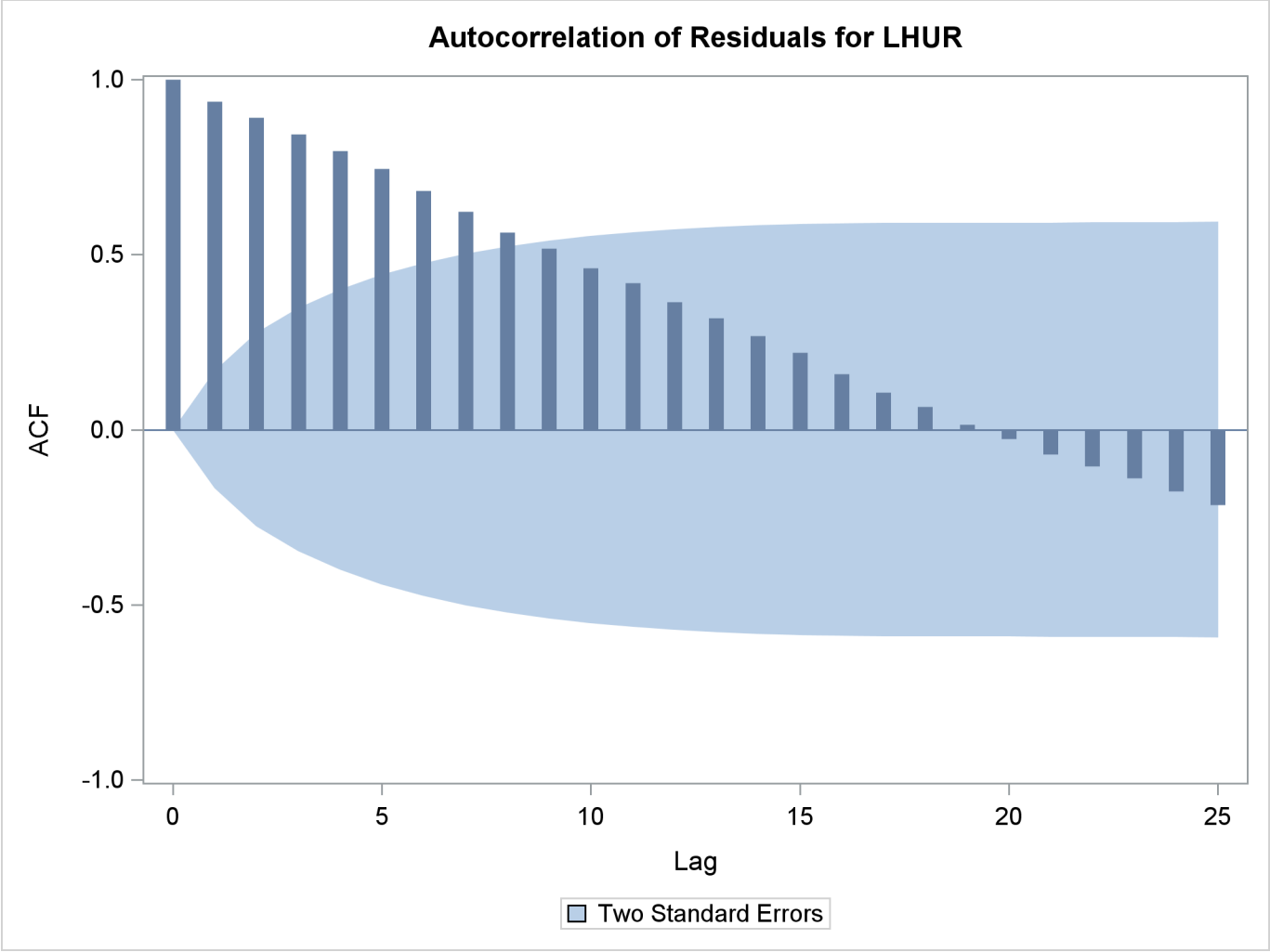
Output 19.20.4 Cook's D Plot



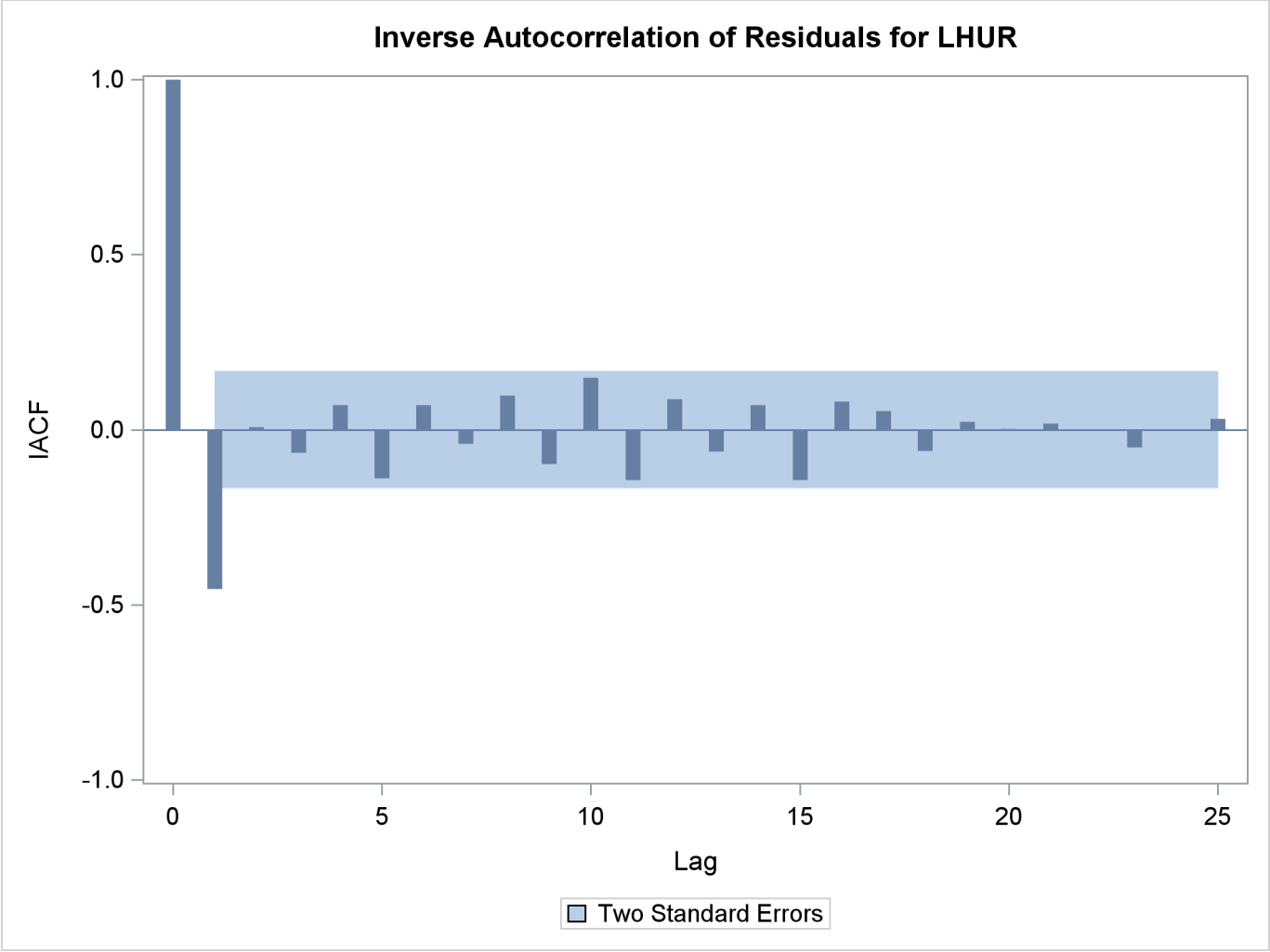
Output 19.20.5 Predicted versus Actual Plot



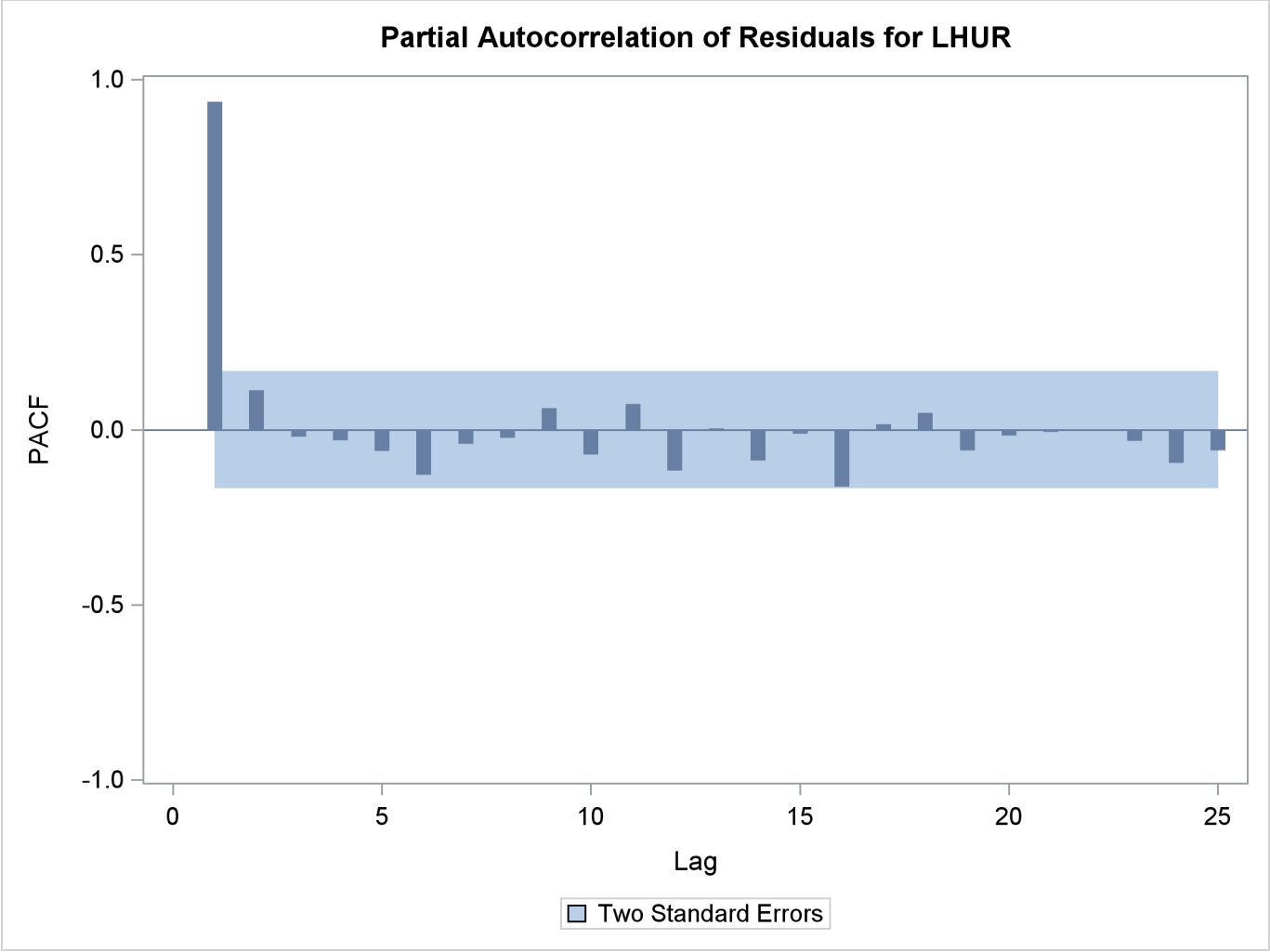
Output 19.20.6 Autocorrelation of Residuals Plot



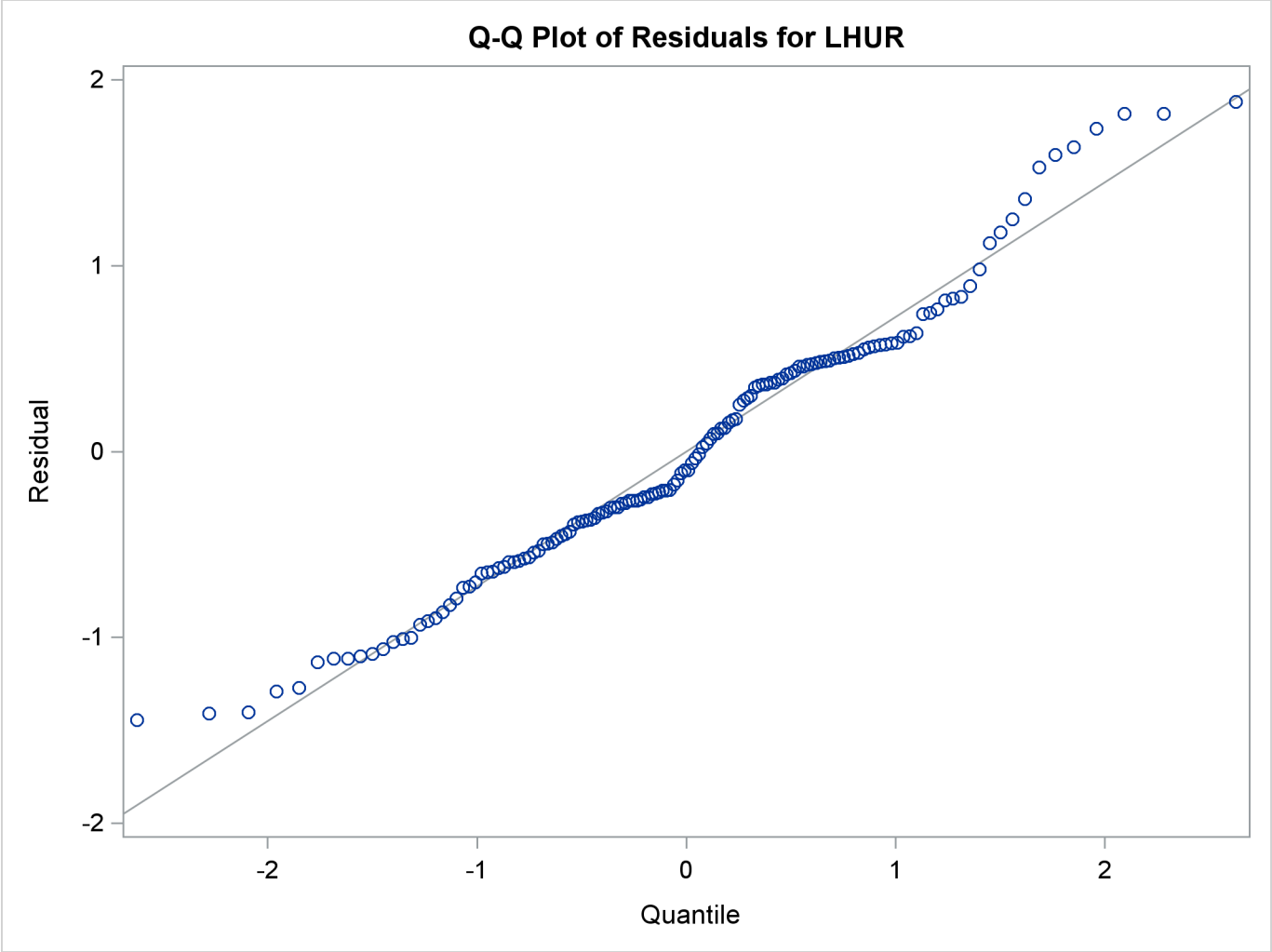
Output 19.20.7 Partial Autocorrelation of Residuals Plot



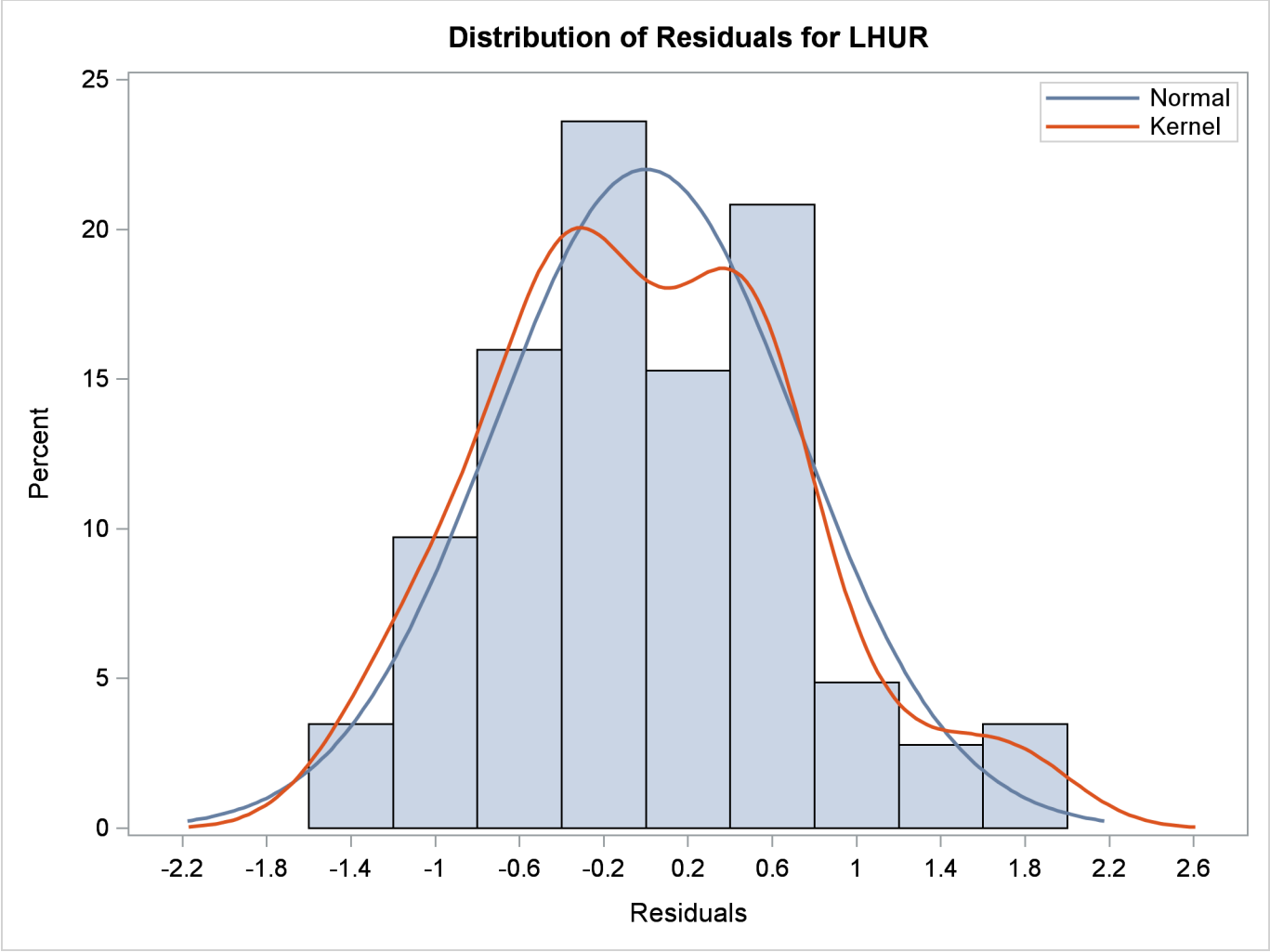
Output 19.20.8 Inverse Autocorrelation of Residuals Plot



Output 19.20.9 Q-Q Plot of Residuals



Output 19.20.10 Histogram of Residuals



References

Aiken, R. C. (1985), *Stiff Computation*, New York: Oxford University Press.

Amemiya, T. (1974), "The Nonlinear Two-Stage Least-Squares Estimator," *Journal of Econometrics*, 2, 105–110.

Amemiya, T. (1977), "The Maximum Likelihood Estimator and the Nonlinear Three-Stage Least Squares Estimator in the General Nonlinear Simultaneous Equation Model," *Econometrica*, 45, 955–968.

Amemiya, T. (1985), *Advanced Econometrics*, Cambridge, MA: Harvard University Press.

Andersen, T. G., Chung, H.-J., and Sorensen, B. E. (1999), "Efficient Method of Moments Estimation of a Stochastic Volatility Model: A Monte Carlo Study," *Journal of Econometrics*, 91, 61–87.

Andersen, T. G. and Sorensen, B. E. (1996), "GMM Estimation of a Stochastic Volatility Model: A Monte Carlo Study," *Journal of Business and Economic Statistics*, 14, 328–352.

Andrews, D. W. K. (1991), "Heteroscedasticity and Autocorrelation Consistent Covariance Matrix Estimation," *Econometrica*, 59, 817–858.

Andrews, D. W. K. and Monahan, J. C. (1992), "Improved Heteroscedasticity and Autocorrelation Consistent Covariance Matrix Estimator," *Econometrica*, 60, 953–966.

Bansal, R., Gallant, A. R., Hussey, R., and Tauchen, G. E. (1993), "Computational Aspects of Nonparametric Simulation Estimation," in D. A. Belsey, ed., *Computational Techniques for Econometrics and Economic Analysis*, 3–22, Boston: Kluwer Academic.

Bansal, R., Gallant, A. R., Hussey, R., and Tauchen, G. E. (1995), "Nonparametric Estimation of Structural Models for High-Frequency Currency Market Data," *Journal of Econometrics*, 66, 251–287.

Bard, Y. (1974), *Nonlinear Parameter Estimation*, New York: Academic Press.

Bates, D. M. and Watts, D. G. (1981), "A Relative Offset Orthogonality Convergence Criterion for Nonlinear Least Squares," *Technometrics*, 23, 179–183.

Belsley, D. A., Kuh, E., and Welsch, R. E. (1980), *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, New York: John Wiley & Sons.

Binkley, J. K. and Nelson, G. (1984), "Impact of Alternative Degrees of Freedom Corrections in Two and Three Stage Least Squares," *Journal of Econometrics*, 24, 223–233.

Bowden, R. J. and Turkington, D. A. (1984), *Instrumental Variables*, Cambridge: Cambridge University Press.

Bratley, P., Fox, B. L., and Niederreiter, H. (1992), "Implementation and Tests of Low-Discrepancy Sequences," *ACM Transactions on Modeling and Computer Simulation*, 2, 195–213.

Breusch, T. S. and Pagan, A. R. (1979), "A Simple Test for Heteroscedasticity and Random Coefficient Variation," *Econometrica*, 47, 1287–1294.

Breusch, T. S. and Pagan, A. R. (1980), "The Lagrange Multiplier Test and Its Applications to Model Specification in Econometrics," *Review of Econometric Studies*, 47, 239–253.

Byrne, G. D. and Hindmarsh, A. C. (1975), "A Polyalgorithm for the Numerical Solution of ODEs," *ACM Transactions on Mathematical Software*, 1, 71–96.

Calzolari, G. and Panattoni, L. (1988), "Alternative Estimators of FIML Covariance Matrix: A Monte Carlo Study," *Econometrica*, 56, 701–714.

Chan, K. C., Karolyi, G. A., Longstaff, F. A., and Sanders, A. B. (1992), "An Empirical Comparison of Alternate Models of the Short-Term Interest Rate," *Journal of Finance*, 47, 1209–1227.

Christensen, L. R., Jorgenson, D. W., and Lau, L. J. (1975), "Transcendental Logarithmic Utility Functions," *American Economic Review*, 65, 367–383.

Dagenais, M. G. (1978), "The Computation of FIML Estimates as Iterative Generalized Least Squares Estimates in Linear and Nonlinear Simultaneous Equation Models," *Econometrica*, 46, 1351–1362.

Davidian, M. and Giltinan, D. M. (1995), *Nonlinear Models for Repeated Measurement Data*, New York: Chapman & Hall.

Davidson, R. and MacKinnon, J. G. (1993), *Estimation and Inference in Econometrics*, New York: Oxford University Press.

Duffie, D. and Singleton, K. J. (1993), "Simulated Moments Estimation of Markov Models of Asset Prices," *Econometrica*, 61, 929–952.

Dulmage, A. L. and Mendelsohn, N. F. (1958), "Coverings of Bipartite Graphs," *Canadian Journal of Mathematics*, 10, 517–534.

Fair, R. C. (1984), *Specification, Estimation, and Analysis of Macroeconometric Models*, Cambridge, MA: Harvard University Press.

Ferson, W. E. and Foerster, S. R. (1994), "Finite Sample Properties of the Generalized Method of Moments in Tests of Conditional Asset Pricing Models," *Journal of Financial Economics*, 36, 29–56, previously released as Working Paper No. 77, University of Washington.

Fox, B. L. (1986), "Algorithm 647: Implementation and Relative Efficiency of Quasirandom Sequence Generators," *ACM Transactions on Mathematical Software*, 12, 362–276.

Gallant, A. R. (1987), *Nonlinear Statistical Models*, New York: John Wiley & Sons.

Gallant, A. R. and Holly, A. (1980), "Statistical Inference in an Implicit, Nonlinear, Simultaneous Equation Model in the Context of Maximum Likelihood Estimation," *Econometrica*, 48, 697–720.

Gallant, A. R. and Jorgenson, D. W. (1979), "Statistical Inference for a System of Simultaneous, Nonlinear, Implicit Equations in the Context of Instrumental Variables Estimation," *Journal of Econometrics*, 11, 275–302.

Gallant, A. R. and Long, J. R. (1997), "Estimating Stochastic Differential Equations Efficiently by Minimum Chi-Squared," *Biometrika*, 84, 125–141.

Gallant, A. R. and Tauchen, G. E. (2001), "Efficient Method of Moments," Working Paper.
URL http://www.econ.duke.edu/~get/wpapers/ee.pdf

Gill, P. E., Murray, W., and Wright, M. H. (1981), *Practical Optimization*, New York: Academic Press.

Godfrey, L. G. (1978a), "Testing against General Autoregressive and Moving Average Error Models When the Regressors Include Lagged Dependent Variables," *Econometrica*, 46, 1293–1301.

Godfrey, L. G. (1978b), "Testing for Higher Order Serial Correlation in Regression Equations When the Regressors Include Lagged Dependent Variables," *Econometrica*, 46, 1303–1310.

Goldfeld, S. M. and Quandt, R. E. (1972), *Nonlinear Methods in Econometrics*, Amsterdam: North-Holland.

Goldfeld, S. M. and Quandt, R. E. (1973a), "The Estimation of Structural Shifts by Switching Regressions," *Annals of Economic and Social Measurement*, 2, 475–486.

Goldfeld, S. M. and Quandt, R. E. (1973b), "A Markov Model for Switching Regressions," *Journal of Econometrics*, 3–16.

Goldfeld, S. M. and Quandt, R. E. (1976), *Studies in Nonlinear Estimation*, Cambridge, MA: Ballinger.

Goodnight, J. H. (1979), "A Tutorial on the Sweep Operator," *American Statistician*, 33, 149–158.

Gourieroux, C. and Monfort, A. (1993), "Simulation Based Inference: A Survey with Special Reference to Panel Data Models," *Journal of Econometrics*, 59, 5–33.

Greene, W. H. (1993), *Econometric Analysis*, 2nd Edition, New York: Macmillan.

Greene, W. H. (2004), *Econometric Analysis*, New York: Macmillan.

Gregory, A. W. and Veall, M. R. (1985), "On Formulating Wald Tests for Nonlinear Restrictions," *Econometrica*, 53, 1465–1468.

Grunfeld, Y. and Griliches, Z. (1960), "Is Aggregation Necessarily Bad?" *Review of Economics and Statistics*, 113–134.

Hansen, L. P. (1982), "Large Sample Properties of Generalized Method of Moments Estimators," *Econometrica*, 50, 1029–1054.

Hansen, L. P. (1985), "A Method for Calculating Bounds on the Asymptotic Covariance Matrices of Generalized Method of Moments Estimators," *Journal of Econometrics*, 30, 203–238.

Hatanaka, M. (1978), "On the Efficient Estimation Methods for the Macro-economic Models Nonlinear in Variables," *Journal of Econometrics*, 8, 323–356.

Hausman, J. A. (1978), "Specification Tests in Econometrics," *Econometrica*, 46, 1251–1271.

Hausman, J. A. and Taylor, W. E. (1982), "A Generalized Specification Test," *Economics Letters*, 8, 239–245.

Henze, N. and Zirkler, B. (1990), "A Class of Invariant Consistent Tests for Multivariate Normality," *Communications in Statistics—Theory and Methods*, 19, 3595–3617.

Johnston, J. (1984), *Econometric Methods*, 3rd Edition, New York: McGraw-Hill.

Jorgenson, D. W. and Laffont, J. (1974), "Efficient Estimation of Nonlinear Simultaneous Equations with Additive Disturbances," *Annals of Social and Economic Measurement*, 3, 615–640.

Joy, C., Boyle, P. P., and Tan, K. S. (1996), "Quasi-Monte Carlo Methods in Numerical Finance," *Management Science*, 42, 926–938.

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.

Lee, B. and Ingram, B. (1991), "Simulation Estimation of Time Series Models," *Journal of Econometrics*, 47, 197–205.

MacKinnon, J. G. and White, H. (1985), "Some Heteroskedasticity-Consistent Covariance Matrix Estimators with Improved Finite Sample Properties," *Journal of Econometrics*, 29, 305–325.

Maddala, G. S. (1977), *Econometrics*, New York: McGraw-Hill.

Mardia, K. V. (1970), "Measures of Multivariate Skewness and Kurtosis with Applications," *Biometrika*, 57, 519–530.

Mardia, K. V. (1974), "Applications of Some Measures of Multivariate Skewness and Kurtosis in Testing Normality and Robustness Studies," *Indian Journal of Statistics*, 36, 115–128.

Matis, J. H., Miller, T. H., and Allen, D. M. (1991), *Metal Ecotoxicology: Concepts and Applications*, Chelsea, MI: Lewis Publishers.

Matsumoto, M. and Nishimura, T. (1998), "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-random Number Generator," *ACM Transactions on Modeling and Computer Simulation*, 8, 3–30.

McFadden, D. (1989), "A Method of Simulated Moments for Estimation of Discrete Response Models without Numerical Integration," *Econometrica*, 57, 995–1026.

McNeil, A., Frey, R., and Embrechts, P. (2005), *Quantitative Risk Management: Concepts, Techniques, and Tools*, Princeton, NJ: Princeton University Press.

Messer, K. and White, H. (1994), "A Note on Computing the Heteroskedasticity Consistent Covariance Matrix Using Instrumental Variable Techniques," *Oxford Bulletin of Economics and Statistics*, 46, 181–184.

Mikhail, W. M. (1975), "A Comparative Monte Carlo Study of the Properties of Economic Estimators," *Journal of the American Statistical Association*, 70, 94–104.

Miller, D. M. (1984), "Reducing Transformation Bias in Curve Fitting," *American Statistician*, 38, 124–126.

Morelock, M. M., Pargellis, C. A., Graham, E. T., Lamarre, D., and Jung, G. (1995), "Time-Resolved Ligand Exchange Reactions: Kinetic Models for Competitive Inhibitors with Recombinant Human Renin," *Journal of Medical Chemistry*, 38, 1751–1761.

Nelsen, R. B. (1999), *An Introduction to Copulas*, New York: Springer-Verlag.

Newey, W. K. and West, D. W. (1987), "A Simple, Positive Semi-definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix," *Econometrica*, 55, 703–708.

Noble, B. and Daniel, J. W. (1977), *Applied Linear Algebra*, Englewood Cliffs, NJ: Prentice-Hall.

Ortega, J. M. and Rheinbolt, W. C. (1970), *Iterative Solution of Nonlinear Equations in Several Variables*, Burlington, MA: Academic Press.

Pakes, A. and Pollard, D. (1989), "Simulation and the Asymptotics of Optimization Estimators," *Econometrica*, 57, 1027–1057.

Parzen, E. (1957), "On Consistent Estimates of the Spectrum of a Stationary Time Series," *Annals of Mathematical Statistics*, 28, 329–348.

Pearlman, J. G. (1980), "An Algorithm for the Exact Likelihood of a High-Order Autoregressive–Moving Average Process," *Biometrika*, 67, 232–233.

Petzold, L. R. (1982), "Differential/Algebraic Equations Are Not ODEs," *SIAM Journal on Scientific and Statistical Computing*, 3, 367–384.

Phillips, C. B. and Park, J. Y. (1988), "On Formulating Wald Tests of Nonlinear Restrictions," *Econometrica*, 56, 1065–1083.

Pindyck, R. S. and Rubinfeld, D. L. (1981), *Econometric Models and Econometric Forecasts*, 2nd Edition, New York: McGraw-Hill.

Pothen, A. and Fan, C.-J. (1990), "Computing the Block Triangular Form of a Sparse Matrix," *ACM Transactions on Mathematical Software*, 16, 303–324.

Rebonato, R. and Jäckel, P. (1999), "The Most General Methodology for Creating Valid Correlation Matrix for Risk Management and Option Pricing Purposes," *Journal of Risk*, 2, 17–27.

Savin, N. E. and White, K. J. (1978), "Testing for Autocorrelation with Missing Observations," *Econometrica*, 46, 59–67.

Sobol, I. M. (1994), *A Primer for the Monte Carlo Method*, Boca Raton, FL: CRC Press.

Srivastava, V. and Giles, D. E. A. (1987), *Seemingly Unrelated Regression Equation Models*, New York: Marcel Dekker.

Theil, H. (1971), *Principles of Econometrics*, New York: John Wiley & Sons.

Thursby, J. (1982), "Misspecification, Heteroscedasticity, and the Chow and Goldfield-Quandt Test," *Review of Economics and Statistics*, 64, 314–321.

Venzon, D. J. and Moolgavkar, S. H. (1988), "A Method for Computing Profile-Likelihood Based Confidence Intervals," *Applied Statistics*, 37, 87–94.

White, H. (1980), "A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity," *Econometrica*, 48, 817–838.

Wu, D. M. (1973), "Alternative Tests of Independence between Stochastic Regressors and Disturbances," *Econometrica*, 41, 733–750.

Subject Index

Syntax Index