SAS/STAT® User's Guide
The MCMC Procedure

2023.04*

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

SAS® Documentation
June 15, 2023

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

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2023. *SAS/STAT® User's Guide*. Cary, NC: SAS Institute Inc.

SAS/STAT® User's Guide

June 2023

Chapter 80
The MCMC Procedure

Contents

Overview: MCMC Procedure

The MCMC procedure is a general purpose Markov chain Monte Carlo (MCMC) simulation procedure that is designed to fit Bayesian models. Bayesian statistics is different from traditional statistical methods such as frequentist or classical methods. For a short introduction to Bayesian analysis and related basic concepts, see Chapter 8, "Introduction to Bayesian Analysis Procedures." Also see the section "A Bayesian Reading List" on page 178 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for a guide to Bayesian textbooks of varying degrees of difficulty.

In essence, Bayesian statistics treats parameters as unknown random variables, and it makes inferences based on the posterior distributions of the parameters. There are several advantages associated with this approach to statistical inference. Some of the advantages include its ability to use prior information and to directly answer specific scientific questions that can be easily understood. For further discussions of the relative advantages and disadvantages of Bayesian analysis, see the section "Bayesian Analysis: Advantages and Disadvantages" on page 154 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

It follows from Bayes' theorem that a posterior distribution is the product of the likelihood function and the prior distribution of the parameter. In all but the simplest cases, it is very difficult to obtain the posterior distribution directly and analytically. Often, Bayesian methods rely on simulations to generate sample from the desired posterior distribution and use the simulated draws to approximate the distribution and to make all of the inferences.

PROC MCMC is a flexible, simulation-based procedure that is suitable for fitting a wide range of Bayesian models. To use PROC MCMC, you need to specify a likelihood function for the data and a prior distribution for the parameters. If you are fitting hierarchical models, you can specify a hyperprior distribution or distributions for the random-effects parameters. PROC MCMC then obtains samples from the corresponding posterior distributions, produces summary and diagnostic statistics, and saves the posterior samples in an output data set that can be used for further analysis. Although PROC MCMC supports a suite of standard distributions, you can analyze data that have any likelihood, prior, or hyperprior, as long as these functions are programmable using the SAS DATA step functions. There are no constraints on how the parameters can enter the model, in either linear or any nonlinear functional form.

The MODEL statement in PROC MCMC can automatically model missing data, response variables, or covariates. The procedure treats the missing values as unknown parameters and incorporates the sampling of the missing values as part of the simulation.

PROC MCMC selects a sampling method for each parameter or a block of parameters. For example, when conjugacy is available, samples are drawn directly from the full conditional distribution by using standard random number generators. In other cases, PROC MCMC uses an adaptive blocked random walk Metropolis algorithm that uses a normal proposal distribution. You can also choose alternative sampling algorithms, such as the slice sampler.

PROC MCMC Compared with Other SAS Procedures

PROC MCMC is unlike most other SAS/STAT procedures in that the nature of the statistical inference is Bayesian. You specify prior distributions for the parameters with PRIOR statements, the likelihood function for the data with MODEL statements, and specify, if needed, the random effects with RANDOM statements. PROC MCMC derives inferences from simulation rather than through analytic or numerical methods. You should expect slightly different answers from each run for the same problem, unless the same random number seed is used. The model specification is similar to PROC NLIN, and PROC MCMC shares some of the syntax of PROC NLMIXED.

You can also perform a Bayesian analysis by using the BCHOICE, GENMOD, PHREG, LIFEREG, and FMM procedures for discrete choice models, generalized linear models, accelerated life failure models, Cox regression models, piecewise constant baseline hazard models (also known as piecewise exponential models), and finite mixture models. See Chapter 30, "The BCHOICE Procedure," Chapter 51, "The GENMOD Procedure," Chapter 92, "The PHREG Procedure," Chapter 76, "The LIFEREG Procedure," and Chapter 46, "The FMM Procedure."

You can fit Bayesian generalized linear mixed-effects models (GLMMs) by using the BGLIMM procedure. Although you can use both PROC BGLIMM and PROC MCMC to fit Bayesian GLMMs, PROC BGLIMM is designed specifically for this class of models and uses optimal sampling algorithms that PROC MCMC does not use, for the same model specifications. PROC BGLIMM enables you to fit Bayesian GLMMs that do not require customized specifications, such as nonstandard prior distributions. On the other hand, PROC MCMC supports a larger class of hierarchical models, such as nonlinear random-effects models, that PROC BGLIMM does not support.

Getting Started: MCMC Procedure

There are three examples in this "Getting Started" section: a simple linear regression, the Behrens-Fisher estimation problem, and a random-effects model. The regression model is chosen for its simplicity; the Behrens-Fisher problem illustrates some advantages of the Bayesian approach; and the random-effects model is one of the most prevalently used models.

Keep in mind that PARMS statements declare the parameters in the model, PRIOR statements declare the prior distributions, MODEL statements declare the likelihood for the data, and RANDOM statements declare the random effects. In most cases, you do not need to supply initial values. PROC MCMC advises you if it is unable to generate starting values for the Markov chain.

Simple Linear Regression

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcgs1.sas.)

This section illustrates some basic features of PROC MCMC by using a linear regression model. The model is as follows:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

for the observations $i = 1, 2, \ldots, n$.

The data set Sashelp.class, which is available in the Sashelp library, identifies the children and their observed heights (the variable Height) and weights (the variable Weight).

The equation of interest is as follows:

$$\text{Weight}_i = \beta_0 + \beta_1 \text{Height}_i + \epsilon_i$$

The observation errors, ϵ_i, are assumed to be independent and identically distributed with a normal distribution with mean zero and variance σ^2.

$$\text{Weight}_i \sim \text{normal}(\beta_0 + \beta_1 \text{Height}_i, \sigma^2)$$

The likelihood function for each of the Weight, which is specified in the MODEL statement, is as follows:

$$p(\text{Weight}|\beta_0, \beta_1, \sigma^2, \text{Height}_i) = \phi(\beta_0 + \beta_1 \text{Height}_i, \sigma^2)$$

where $p(\cdot|\cdot)$ denotes a conditional probability density and ϕ is the normal density. There are three parameters in the likelihood: β_0, β_1, and σ^2. You use the PARMS statement to indicate that these are the parameters in the model.

Suppose you want to use the following three prior distributions on each of the parameters:

$$\begin{aligned} \pi(\beta_0) &= \phi(0, \text{var} = 1e6) \\ \pi(\beta_1) &= \phi(0, \text{var} = 1e6) \\ \pi(\sigma^2) &= f_{i\Gamma}(\text{shape} = 3/10, \text{scale} = 10/3) \end{aligned}$$

where $\pi(\cdot)$ indicates a prior distribution and $f_{i\Gamma}$ is the density function for the inverse gamma distribution. The normal priors on β_0 and β_1 have large variances, expressing your lack of knowledge about the regression coefficients. The priors correspond to an equal-tail 95% credible intervals of approximately (-2000, 2000) for β_0 and β_1. Priors of this type are often called *vague* or *diffuse* priors. See the section "Prior Distributions" on page 150 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more information. Typically diffuse prior distributions have little influence on the posterior distribution and are appropriate when stronger prior information about the parameters is not available.

A frequently used prior for the variance parameter σ^2 is the inverse gamma distribution. See Table 80.22 in the section "Standard Distributions" on page 6275 for the density definition. The inverse gamma distribution is a conjugate prior (see the section "Conjugate Sampling" on page 6272) for the variance parameter in a normal distribution. Also see the section "Gamma and Inverse Gamma Distributions" on page 6332 for typical usages of the gamma and inverse gamma prior distributions. With a shape parameter of 3/10 and a

scale parameter of 10/3, this prior corresponds to an equal-tail 95% credible interval of (1.7, 1E6), with the mode at 2.5641 for σ^2. Alternatively, you can use any other prior distribution with positive support on this variance component. For example, you can use the gamma prior.

According to Bayes' theorem, the likelihood function and prior distributions determine the posterior (joint) distribution of β_0, β_1, and σ^2 as follows:

$$\pi(\beta_0, \beta_1, \sigma^2|\text{Weight}, \text{Height}) \quad \propto \quad \pi(\beta_0)\pi(\beta_1)\pi(\sigma^2)\, p(\text{Weight}|\beta_0, \beta_1, \sigma^2, \text{Height})$$

You do not need to know the form of the posterior distribution when you use PROC MCMC. PROC MCMC automatically obtains samples from the desired posterior distribution, which is determined by the prior and likelihood you supply.

The following statements fit this linear regression model with diffuse prior information:

```
ods graphics on;
proc mcmc data=sashelp.class outpost=classout nmc=10000 thin=2 seed=246810;
   parms beta0 0 beta1 0;
   parms sigma2 1;
   prior beta0 beta1 ~ normal(mean = 0, var = 1e6);
   prior sigma2 ~ igamma(shape = 3/10, scale = 10/3);
   mu = beta0 + beta1*height;
   model weight ~ n(mu, var = sigma2);
run;
ods graphics off;
```

When ODS Graphics is enabled, diagnostic plots, such as the trace and autocorrelation function plots of the posterior samples, are displayed. For more information about ODS Graphics, see Chapter 24, "Statistical Graphics Using ODS."

The PROC MCMC statement invokes the procedure and specifies the input data set Class. The output data set Classout contains the posterior samples for all of the model parameters. The NMC= option specifies the number of posterior simulation iterations. The THIN= option controls the thinning of the Markov chain and specifies that one of every 2 samples is kept. Thinning is often used to reduce the correlations among posterior sample draws. In this example, 5,000 simulated values are saved in the Classout data set. The SEED= option specifies a seed for the random number generator, which guarantees the reproducibility of the random stream. For more information about Markov chain sample size, burn-in, and thinning, see the section "Burn-In, Thinning, and Markov Chain Samples" on page 162 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

The PARMS statements identify the three parameters in the model: beta0, beta1, and sigma2. Each statement also forms a block of parameters, where the parameters are updated simultaneously in each iteration. In this example, beta0 and beta1 are sampled jointly, conditional on sigma2; and sigma2 is sampled conditional on fixed values of beta0 and beta1. In simple regression models such as this, you expect the parameters beta0 and beta1 to have high posterior correlations, and placing them both in the same block improves the mixing of the chain—that is, the efficiency that the posterior parameter space is explored by the Markov chain. For more information, see the section "Blocking of Parameters" on page 6266. The PARMS statements also assign initial values to the parameters (see the section "Initial Values of the Markov Chains" on page 6273). The regression parameters are given 0 as their initial values, and the scale parameter sigma2 starts at value 1. If you do not provide initial values, PROC MCMC chooses starting values for every parameter.

The PRIOR statements specify prior distributions for the parameters. The parameters beta0 and beta1 both share the same prior—a normal prior with mean 0 and variance 1e6. The parameter sigma2 has an inverse gamma distribution with a shape parameter of 3/10 and a scale parameter of 10/3. For a list of standard distributions that PROC MCMC supports, see the section "Standard Distributions" on page 6275.

The MU assignment statement calculates the expected value of Weight as a linear function of Height. The MODEL statement uses the shorthand notation, `n`, for the normal distribution to indicate that the response variable, Weight, is normally distributed with parameters mu and sigma2. The functional argument MEAN= in the normal distribution is optional, but you have to indicate whether sigma2 is a variance (VAR=), a standard deviation (SD=), or a precision (PRECISION=) parameter. See Table 80.2 in the section "MODEL Statement" on page 6240 for distribution specifications.

The distribution parameters can contain expressions. For example, you can write the MODEL statement as follows:

```
model weight ~ n(beta0 + beta1*height, var = sigma2);
```

Before you do any posterior inference, it is essential that you examine the convergence of the Markov chain (see the section "Assessing Markov Chain Convergence" on page 162 in Chapter 8, "Introduction to Bayesian Analysis Procedures"). You cannot make valid inferences if the Markov chain has not converged. A very effective convergence diagnostic tool is the trace plot. Although PROC MCMC produces graphs at the end of the procedure output (see Figure 80.5), you should visually examine the convergence graph first.

The first table that PROC MCMC produces is the "Number of Observations" table, as shown in Figure 80.1. This table lists the number of observations read from the DATA= data set and the number of observations used in the analysis.

Figure 80.1 Observation Information

The MCMC Procedure

Number of Observations Read	19
Number of Observations Used	19

The "Parameters" table, shown in Figure 80.2, lists the names of the parameters, the blocking information, the sampling method used, the starting values, and the prior distributions. For more information about blocking information, see the section "Blocking of Parameters" on page 6266; for more information about starting values, see the section "Initial Values of the Markov Chains" on page 6273. The first block, which consists of the parameters beta0 and beta1, uses a random walk Metropolis algorithm. The second block, which consists of the parameter sigma2, is updated via its full conditional distribution in conjugacy. You should check this table to ensure that you have specified the parameters correctly, especially for complicated models.

Figure 80.2 Parameter Information

Parameters

Block	Parameter	Sampling Method	Initial Value	Prior Distribution
1	**beta0**	N-Metropolis	0	normal(mean = 0, var = 1e6)
	beta1		0	normal(mean = 0, var = 1e6)
2	**sigma2**	Conjugate	1.0000	igamma(shape = 3/10, scale = 10/3)

For each posterior distribution, PROC MCMC also reports summary and interval statistics (posterior means,

standard deviations, and 95% highest posterior density credible intervals), as shown in Figure 80.3. For more information about posterior statistics, see the section "Summary Statistics" on page 175 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

Figure 80.3 MCMC Summary and Interval Statistics

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta0	5000	-142.8	33.4337	-210.8	-81.6694
beta1	5000	3.8924	0.5333	2.9056	4.9545
sigma2	5000	137.3	51.1034	59.2363	236.3

By default, PROC MCMC computes the effective sample sizes (ESSs) as a convergence diagnostic test to help you determine whether the chain has converged. The ESSs are shown in Figure 80.4. For details and interpretations of ESS and additional convergence diagnostics, see the section "Assessing Markov Chain Convergence" on page 162 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

Figure 80.4 MCMC Convergence Diagnostics

The MCMC Procedure

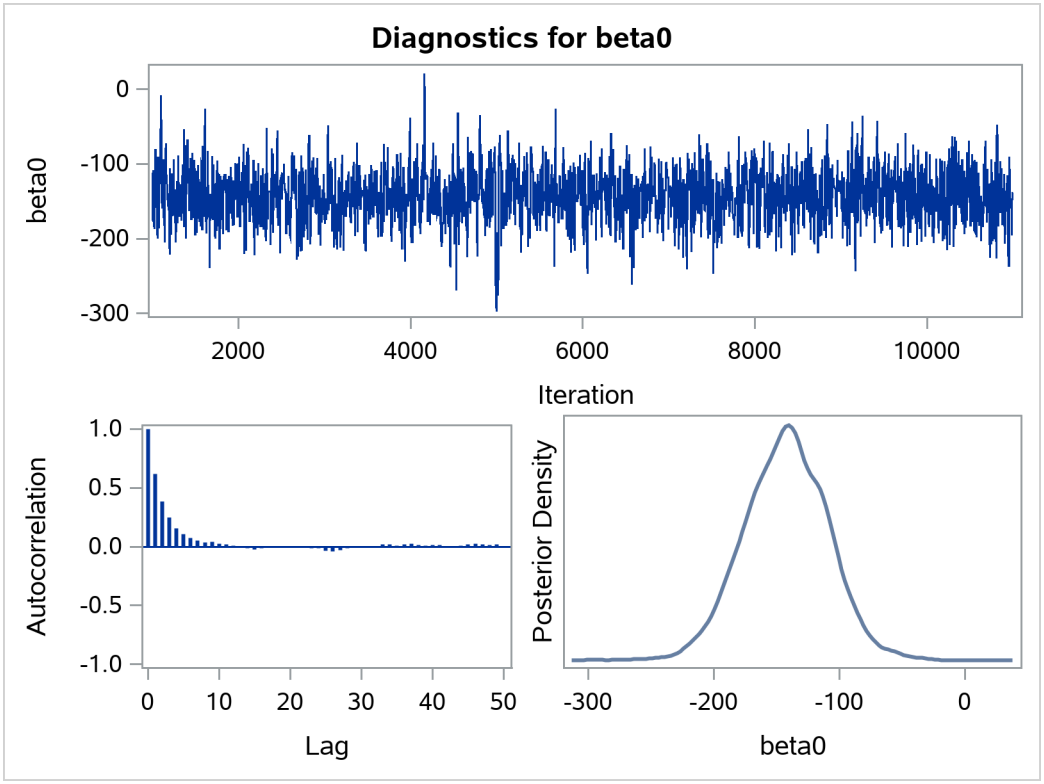
Effective Sample Sizes

Parameter	ESS	Autocorrelation Time	Efficiency
beta0	1102.2	4.5366	0.2204
beta1	1119.0	4.4684	0.2238
sigma2	2910.0	1.7182	0.5820

PROC MCMC produces a number of graphs, shown in Figure 80.5, which also aid convergence diagnostic checks. With the trace plots, there are two important aspects to examine. First, you want to check whether the mean of the Markov chain has stabilized and appears constant over the graph. Second, you want to check whether the chain has good mixing and is "dense," in the sense that it quickly traverses the support of the distribution to explore both the tails and the mode areas efficiently. The plots show that the chains appear to have reached their stationary distributions.

Next, you should examine the autocorrelation plots, which indicate the degree of autocorrelation for each of the posterior samples. High correlations usually imply slow mixing. Finally, the kernel density plots estimate the posterior marginal distributions for each parameter.

Figure 80.5 Diagnostic Plots for β_0, β_1 and σ^2



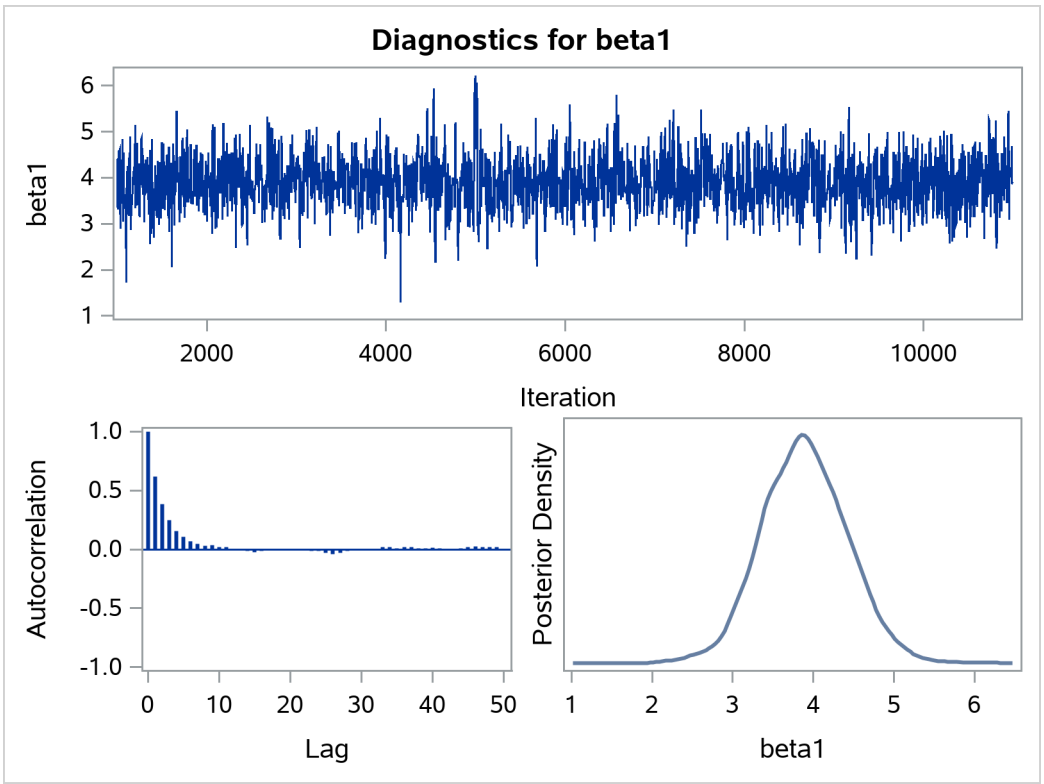
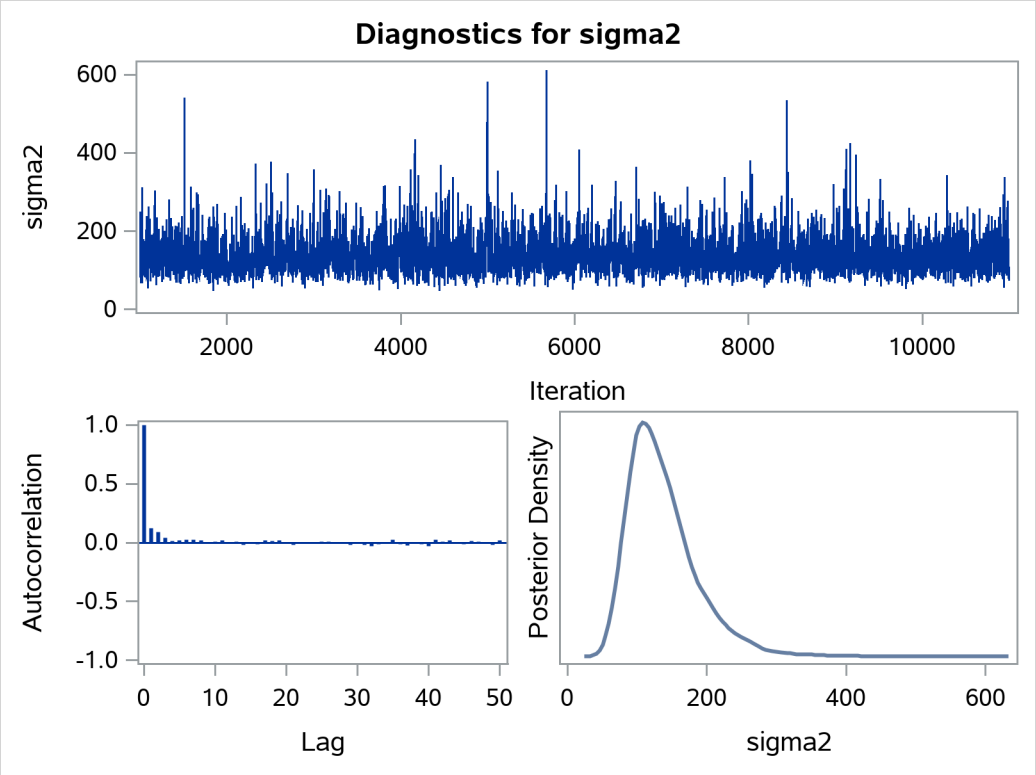


Figure 80.5 *continued*



In regression models such as this, you expect the posterior estimates to be very similar to the maximum likelihood estimators with noninformative priors on the parameters, The REG procedure produces the following fitted model (code not shown):

$$\text{Weight} = -143.0 + 3.9 \times \text{Height}$$

These are very similar to the means show in Figure 80.3. With PROC MCMC, you can carry out informative analysis that uses specifications to indicate prior knowledge on the parameters. Informative analysis is likely to produce different posterior estimates, which are the result of information from both the likelihood and the prior distributions. Incorporating additional information in the analysis is one major difference between the classical and Bayesian approaches to statistical inference.

The Behrens-Fisher Problem

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcgs2.sas.)

One of the famous examples in the history of statistics is the Behrens-Fisher problem (Fisher 1935). Consider the situation where there are two independent samples from two different normal distributions:

$$y_{11}, y_{12}, \ldots, y_{1n_1} \sim \text{normal}(\mu_1, \sigma_1^2)$$
$$y_{21}, y_{22}, \ldots, y_{2n_2} \sim \text{normal}(\mu_2, \sigma_2^2)$$

Note that $n_1 \neq n_2$. When you do not want to assume that the variances are equal, testing the hypothesis $H_0 : \mu_1 = \mu 2$ is a difficult problem in the classical statistics framework, because the distribution under H_0 is not known. Within the Bayesian framework, this problem is straightforward because you can estimate the posterior distribution of $\mu_1 - \mu_2$ while taking into account the uncertainties in all of parameters by treating them as random variables.

Suppose you have the following set of data:

```
title 'The Behrens-Fisher Problem';

data behrens;
   input y ind @@;
   datalines;
121 1  94 1 119 1 122 1 142 1 168 1 116 1
172 1 155 1 107 1 180 1 119 1 157 1 101 1
145 1 148 1 120 1 147 1 125 1 126 2 125 2
130 2 130 2 122 2 118 2 118 2 111 2 123 2
126 2 127 2 111 2 112 2 121 2
;
```

The response variable is y, and the ind variable is the group indicator, which takes two values: 1 and 2. There are 19 observations that belong to group 1 and 14 that belong to group 2.

The likelihood functions for the two samples are as follows:

$$p(y_{1i}|\mu_1, \sigma_1^2) = \phi(y_{1i}; \mu_1, \sigma_1^2) \quad \text{for} \quad i = 1, \ldots, 19$$
$$p(y_{2j}|\mu_2, \sigma_2^2) = \phi(y_{2j}; \mu_2, \sigma_2^2) \quad \text{for} \quad j = 1, \ldots, 14$$

Berger (1985) showed that a uniform prior on the support of the location parameter is a noninformative prior. The distribution is invariant under location transformations—that is, $\theta = \mu + c$. You can use this prior for the mean parameters in the model:

$$\pi(\mu_1) \propto 1$$
$$\pi(\mu_2) \propto 1$$

In addition, Berger (1985) showed that a prior of the form $1/\sigma^2$ is noninformative for the scale parameter, and it is invariant under scale transformations (that is $\tau = c\sigma^2$). You can use this prior for the variance

parameters in the model:

$$\begin{aligned}\pi(\sigma_1^2) &\propto 1/\sigma_1^2\\ \pi(\sigma_2^2) &\propto 1/\sigma_2^2\end{aligned}$$

The log densities of the prior distributions on σ_1^2 and σ_2^2 are:

$$\begin{aligned}\log(\pi(\sigma_1^2)) &= -\log(\sigma_1^2)\\ \log(\pi(\sigma_2^2)) &= -\log(\sigma_2^2)\end{aligned}$$

The following statements generate posterior samples of $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$, and the difference in the means: $\mu_1 - \mu_2$:

```
proc mcmc data=behrens outpost=postout seed=123
          nmc=40000 monitor=(_parms_ mudif)
          statistics(alpha=0.01);
   ods select PostSumInt;
   parm mu1 0 mu2 0;
   parm sig21 1;
   parm sig22 1;
   prior mu: ~ general(0);
   prior sig21 ~ general(-log(sig21), lower=0);
   prior sig22 ~ general(-log(sig22), lower=0);
   mudif = mu1 - mu2;
   if ind = 1 then do;
      mu = mu1;
      s2 = sig21;
   end;
   else do;
      mu = mu2;
      s2 = sig22;
   end;
   model y ~ normal(mu, var=s2);
run;
```

The PROC MCMC statement specifies an input data set (Behrens), an output data set containing the posterior samples (Postout), a random number seed, and the simulation size. The MONITOR= option specifies a list of symbols, which can be either parameters or functions of the parameters in the model, for which inference is to be done. The symbol _parms_ is a shorthand for all model parameters—in this case, mu1, mu2, sig21, and sig22. The symbol mudif is defined in the program as the difference between μ_1 and μ_2.

The global suboption ALPHA=0.01 in the STATISTICS= option specifies 99% highest posterior density (HPD) credible intervals for all parameters.

The ODS SELECT statement displays the summary statistics and interval statistics tables while excluding all other output. For a complete list of ODS tables that PROC MCMC can produce, see the sections "Displayed Output" on page 6355 and "ODS Table Names" on page 6359.

The PARMS statements assign the parameters mu1 and mu2 to the same block, and sig21 and sig22 each to their own separate blocks. There are a total of three blocks. The PARMS statements also assign an initial value to each parameter.

The PRIOR statements specify prior distributions for the parameters. Because the priors are all nonstandard (uniform on the real axis for μ_1 and μ_2 and $1/\sigma^2$ for σ_1^2 and σ_2^2), you must use the GENERAL function here. The argument in the GENERAL function is an expression for the log of the distribution, up to an additive constant. This distribution can have any functional form, as long as it is programmable using SAS functions and expressions. The function specifies a distribution on the log scale, not on the original scale. The log of the prior on mu1 and mu2 is 0, and the log of the priors on sig21 and sig22 are –log(sig21) and –log(sig22) respectively. See the section "Specifying a New Distribution" on page 6290 for more information about how to specify an arbitrary distribution. The LOWER= option indicates that both variance terms must be strictly positive.

The MUDIF assignment statement calculates the difference between mu1 and mu2. The IF-ELSE statements enable different y's to have different mean and variance, depending on their group indicator ind. The MODEL statement specifies the normal likelihood function for each observation in the model.

Figure 80.6 displays the posterior summary and interval statistics.

Figure 80.6 Posterior Summary and Interval Statistics

The Behrens-Fisher Problem

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	99% HPD Interval	
mu1	40000	134.8	6.0092	119.1	152.3
mu2	40000	121.4	1.9119	116.1	126.6
sig21	40000	685.0	255.3	260.0	1580.5
sig22	40000	51.1811	23.8675	14.2322	136.0
mudif	40000	13.3730	6.3095	-3.3609	30.7938

The mean difference has a posterior mean value of 13.37, and the lower endpoints of the 99% credible intervals are negative. This suggests that the mean difference is positive with a high probability. However, if you want to estimate the probability that $\mu_1 - \mu_2 > 0$, you can do so as follows.

The following statements produce Figure 80.7:

```
proc format;
   value diffmt low-0 = 'mu1 - mu2 <= 0' 0<-high = 'mu1 - mu2 > 0';
run;

proc freq data = postout;
   tables mudif /nocum;
   format mudif diffmt.;
run;
```

The sample estimate of the posterior probability that $\mu_1 - \mu_2 > 0$ is 0.98. This example illustrates an advantage of Bayesian analysis. You are not limited to making inferences based on model parameters only. You can accurately quantify uncertainties with respect to any function of the parameters, and this allows for flexibility and easy interpretations in answering many scientific questions.

Figure 80.7 Estimated Probability of $\mu_1 - \mu_2 > 0$.

The Behrens-Fisher Problem

The FREQ Procedure

mudif	Frequency	Percent
mu1 - mu2 <= 0	753	1.88
mu1 - mu2 > 0	39247	98.12

Random-Effects Model

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcgs3.sas.)

This example illustrates how you can fit a normal likelihood random-effects model in PROC MCMC. PROC MCMC offers you the ability to model beyond the normal likelihood (see "Example 80.7: Logistic Regression Random-Effects Model" on page 6401, "Example 80.8: Nonlinear Poisson Regression Multilevel Random-Effects Model" on page 6403, and "Example 80.16: Piecewise Exponential Frailty Model" on page 6449).

Consider a scenario in which data are collected in groups and you want to model group-specific effects. You can use a random-effects model (sometimes also known as a variance-components model):

$$y_{ij} = \beta_0 + \beta_1 x_{ij} + \gamma_i + e_{ij}, \quad e_{ij} \sim \text{normal}(0, \sigma^2)$$

where $i = 1, 2, \ldots, I$ is the group index and $j = 1, 2, \ldots, n_i$ indexes the observations in the ith group. In the regression model, the fixed effects β_0 and β_1 are the intercept and the coefficient for variable x_{ij}, respectively. The random effect γ_i is the mean for the ith group, and e_{ij} are the error term.

Consider the following SAS data set:

```
title 'Random-Effects Model';

data heights;
   input Family G$ Height @@;
   datalines;
1 F 67   1 F 66   1 F 64   1 M 71   1 M 72   2 F 63
2 F 63   2 F 67   2 M 69   2 M 68   2 M 70   3 F 63
3 M 64   4 F 67   4 F 66   4 M 67   4 M 67   4 M 69
;
```

The response variable Height measures the heights (in inches) of 18 individuals. The covariate x is the gender (variable G), and the individuals are grouped according to Family (group index). Since the variable G is a character variable and PROC MCMC does not support a CLASS statement, you need to create the corresponding design matrix. In this example, the design matrix for a factor variable of level 2 (M and F) can be constructed using the following statement:

```
data input;
   set heights;
   if g eq 'F' then gf = 1;
   else gf = 0;
   drop g;
run;
```

The data set variable gf is a numeric variable and can be used in the regression model in PROC MCMC.

In data sets with factor variables that have more levels, you can consider using PROC TRANSREG to construct the design matrix. See the section "Create Design Matrix" on page 6303 for more information.

To model the data, you can assume that Height is normally distributed:

$$y_{ij} \sim \text{normal}(\mu_{ij}, \sigma^2), \quad \mu_{ij} = \beta_0 + \beta_1 \text{gf}_{ij} + \gamma_i$$

The priors on the parameters β_0, β_1, γ_i are also assumed to be normal:

$$\begin{aligned} \beta_0, \beta_1 &\sim \text{normal}(0, \text{var} = 1e5) \\ \gamma_i &\sim \text{normal}(0, \text{var} = \sigma_\gamma^2) \end{aligned}$$

Priors on the variance terms, σ^2 and σ_γ^2, are inverse gamma:

$$\sigma^2, \sigma_\gamma^2 \sim \text{igamma}(\text{shape} = 0.01, \text{scale} = 0.01)$$

The inverse gamma distribution is a conjugate prior for the variance in the normal likelihood and the variance in the prior distribution of the random effect.

The following statements fit a linear random-effects model to the data and produce the output shown in Figure 80.9 and Figure 80.10:

```
ods graphics on;
proc mcmc data=input outpost=postout nmc=50000 seed=7893 plots=trace;
   ods select Parameters REparameters PostSumInt tracepanel;
   parms b0 0 b1 0 s2 1 s2g 1;

   prior b: ~ normal(0, var = 10000);
   prior s: ~ igamma(0.01, scale = 0.01);
   random gamma ~ normal(0, var = s2g) subject=family monitor=(gamma);
   mu = b0 + b1 * gf + gamma;
   model height ~ normal(mu, var = s2);
run;
ods graphics off;
```

Some of the statements are very similar to those shown in the previous two examples. The ODS GRAPHICS ON statement enables ODS Graphics. The PROC MCMC statement specifies the input and output data sets, the simulation size, and a random number seed. The ODS SELECT statement displays the model parameter and random-effects parameter information tables, summary statistics table, the interval statistics table, and the trace plots.

The PARMS statement lumps all four model parameters in a single block. They are b0 (overall intercept), b1 (main effect for gf), s2 (variance of the likelihood function), and s2g (variance of the random effect). If a random walk Metropolis sampler is the only applicable sampler for all parameters, then these four parameters

are updated in a single block. However, because PROC MCMC updates the parameters s2 and s2g via conjugacy, these parameters are separated into individual blocks. (See the Block column in "Parameters" table in Figure 80.8.)

The PRIOR statements specify priors for all the parameters. The notation b: is a shorthand for all symbols that start with the letter 'b'. In this example, b: includes b0 and b1. Similarly, s: stands for both s2 and s2g. This shorthand notation can save you some typing, and it keeps your statements tidy.

The RANDOM statement specifies a single random effect to be gamma, and specifies that it has a normal prior centered at 0 with variance s2g. The SUBJECT= argument in the RANDOM statement defines a group index (family) in the model, where all observations from the same family should have the same group indicator value. The MONITOR= option outputs analysis for all the random-effects parameters.

Finally, the MU assignment statement calculates the expected value of the height of the model. The calculation includes the random-effects term gamma. The MODEL statement specifies the likelihood function for height.

The "Parameters" and "Random-Effects Parameters" tables, shown in Figure 80.8, contain information about the model parameters and the four random-effects parameters.

Figure 80.8 Model and Random-Effects Parameter Information

Random-Effects Model

The MCMC Procedure

Parameters

Block	Parameter	Sampling Method	Initial Value	Prior Distribution
1	s2	Conjugate	1.0000	igamma(0.01, scale = 0.01)
2	s2g	Conjugate	1.0000	igamma(0.01, scale = 0.01)
3	b0	N-Metropolis	0	normal(0, var = 10000)
	b1		0	normal(0, var = 10000)

Random Effect Parameters

Parameter	Sampling Method	Subject	Number of Subjects	Subject Values	Prior Distribution
gamma	N-Metropolis	Family	4	1 2 3 4	normal(0, var = s2g)

The posterior summary and interval statistics for the model parameters and the random-effects parameters are shown in Figure 80.9.

Figure 80.9 Posterior Summary and Interval Statistics

Random-Effects Model

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
b0	50000	68.3094	1.4598	65.5446	70.8492
b1	50000	-3.5358	0.9524	-5.5319	-1.7448
s2	50000	4.1449	1.9730	1.3930	7.9364
s2g	50000	5.9587	28.4772	0.00155	22.0497
gamma_1	50000	1.0855	1.5801	-1.0753	4.6908
gamma_2	50000	0.1550	1.3862	-2.5341	3.0563
gamma_3	50000	-1.2087	1.6501	-5.1118	1.0319
gamma_4	50000	0.2603	1.4094	-2.2312	3.2988

Trace plots for all the parameters are shown in Figure 80.10. The mixing looks very reasonable, suggesting convergence.

Figure 80.10 Plots for b_1 and Log of the Posterior Density

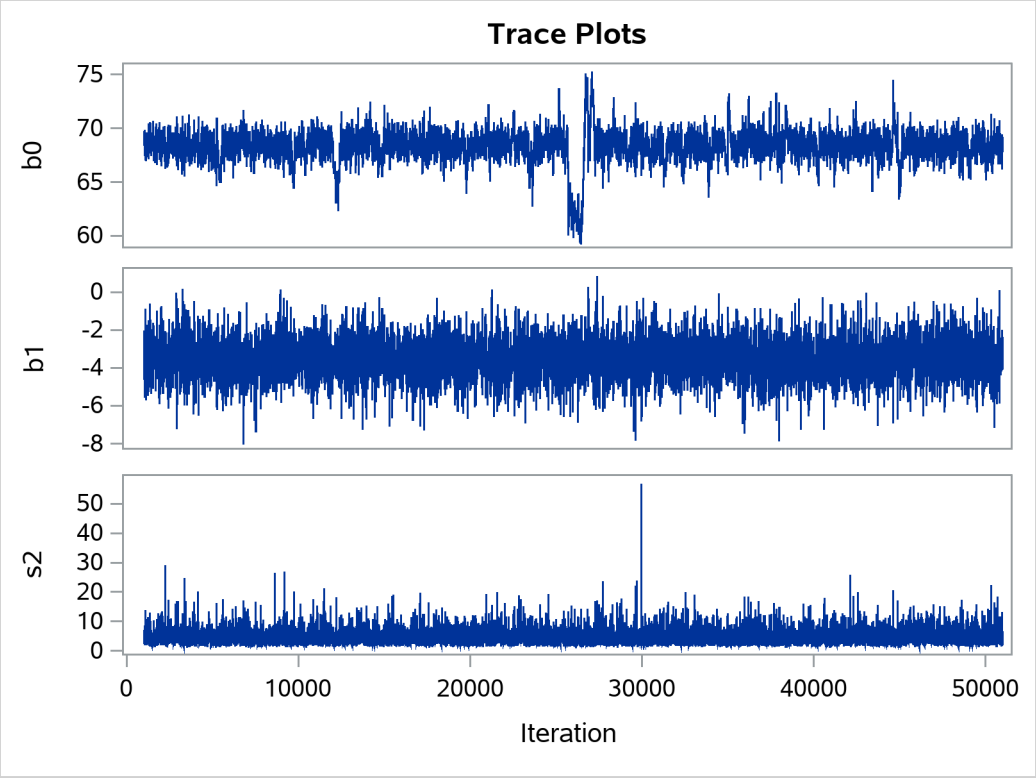
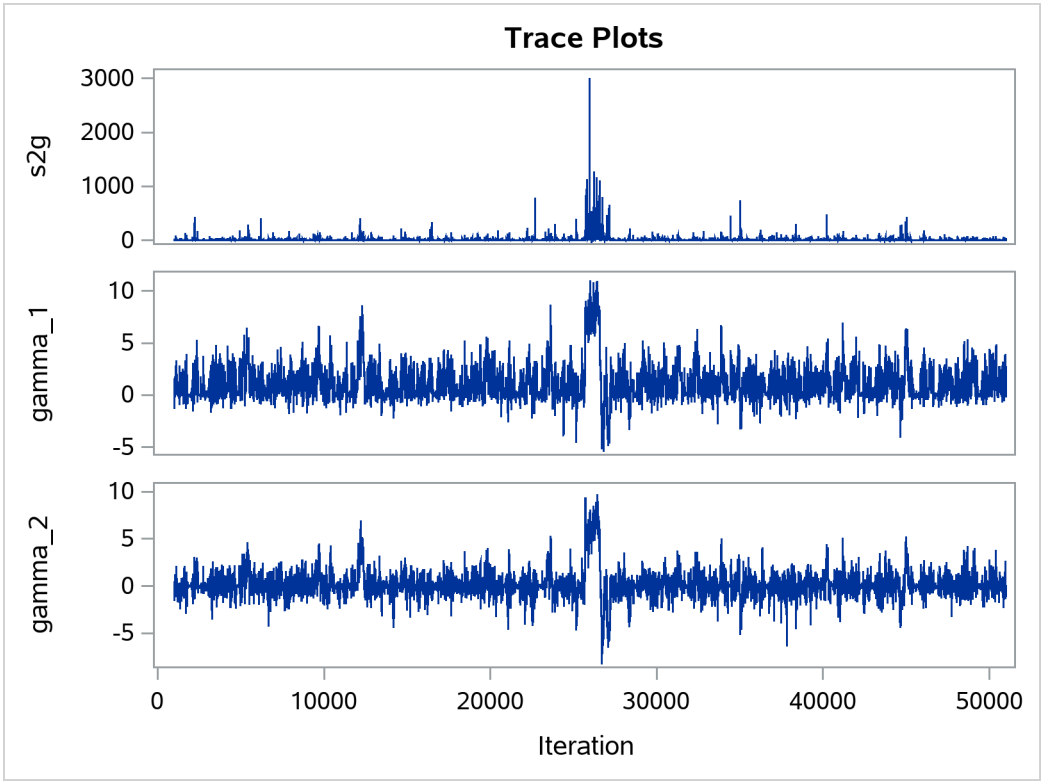
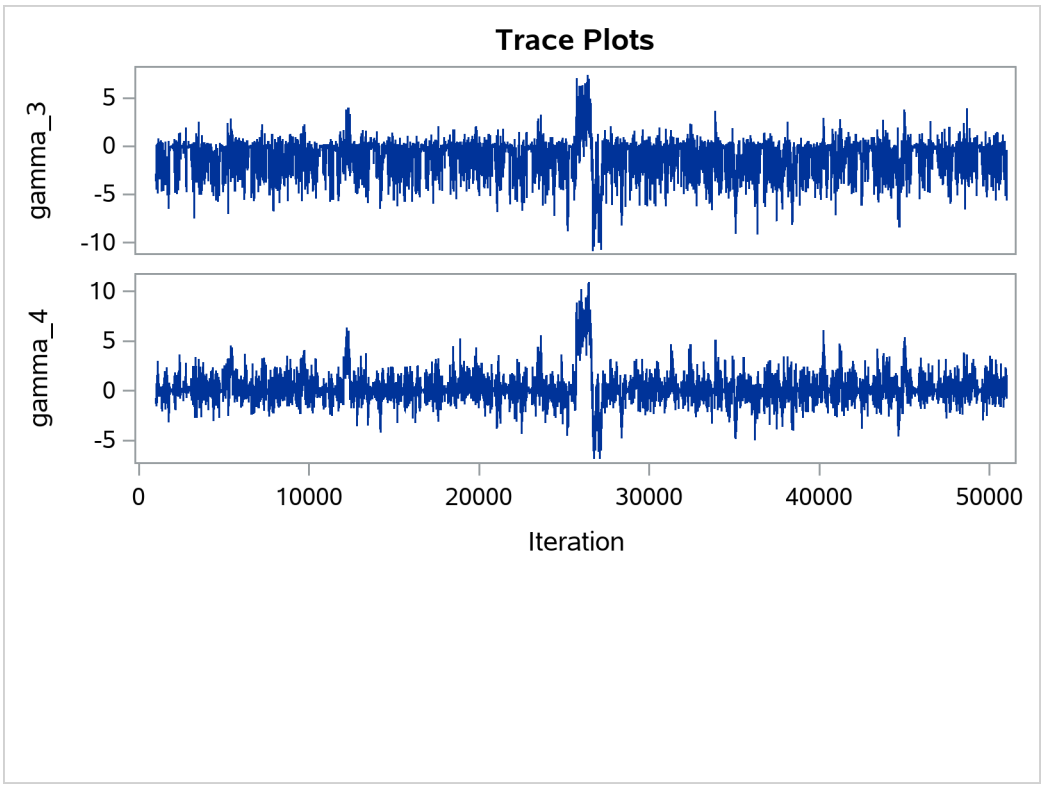


Figure 80.10 *continued*





From the summary statistics table, you see that the HPD interval for β_0 is positive, and the interval for β_1

is negative, respectively. This indicates the positive effect of the intercept and the negative effect of gf on predicting height in this model.

Syntax: MCMC Procedure

The following statements are available in the MCMC procedure. Items within < > are optional.

PROC MCMC < *options* > **;**
ARRAY *arrayname* **[** *dimensions* **]** < **$** > < *variables-and-constants* > **;**
BEGINCNST/ENDCNST **;**
BEGINNODATA/ENDNODATA **;**
BY *variables* **;**
CMPTMODEL *required-options conditionally-required-options* < *options* > **;**
MODEL *variable* ∼ *distribution* < *options* > **;**
PARMS *parameter* < **=** > *number* < / *options* > **;**
PREDDIST < '*label*' > **OUTPRED=***SAS-data-set* < *options* > **;**
PRIOR/HYPERPRIOR *parameter* ∼ *distribution* **;**
Programming statements **;**
RANDOM *random-effects-specification* < / *options* > **;**
UDS *subroutine-name* **(***subroutine-argument-list***) ;**

The PARMS statements declare parameters in the model and assign optional starting values for the Markov chain. The PRIOR/HYPERPRIOR statements specify the prior distributions of the parameters. The MODEL statements specify the log-likelihood functions for the response variables. These statements form the basis of most Bayesian models.

In addition, you can use the ARRAY statement to define constant or parameter arrays, the BEGINCNST/ENDCNST and BEGINNODATA/ENDNODATA statements to omit unnecessary evaluation and reduce simulation time, the PREDDIST statement to generate samples from the posterior predictive distribution, the program statements to specify more complicated models that you want to fit, the RANDOM statement to specify random effects and their prior distributions, and the UDS statement to define your own Gibbs samplers to sample parameters in the model.

The following sections provide a description of each of these statements.

PROC MCMC Statement

PROC MCMC *options* **;**

The PROC MCMC statement invokes the MCMC procedure. Table 80.1 summarizes the *options* available in the PROC MCMC statement.

Table 80.1 PROC MCMC Statement Options

Option	Description
Basic Options	
DATA=	Names the input data set
OUTPOST=	Names the output data set for posterior samples of parameters
Debugging Output	
LIST	Displays the model program and variables
LISTCODE	Displays the compiled model program
TRACE	Displays detailed model execution messages
Frequently Used MCMC Options	
ALG=	Specifies the default sampling algorithm
MAXTUNE=	Specifies the maximum number of tuning loops
MINTUNE=	Specifies the minimum number of tuning loops
NBI=	Specifies the number of burn-in iterations
NMC=	Specifies the number of MCMC iterations, excluding the burn-in iterations
NTHREADS=	Specifies the number of threads to use
NTU=	Specifies the number of tuning iterations
PROPCOV=	Controls options for constructing the initial proposal covariance matrix
SEED=	Specifies the random seed for simulation
THIN=	Specifies the thinning rate
Less Frequently Used MCMC Options	
ACCEPTTOL=	Specifies a tolerance for acceptance probabilities
BINARYJOINT	Jointly samples a block of binary parameters
DISCRETE=	Controls the sampling of discrete parameters
INIT=	Controls the generation of initial values
MCHISTORY=	Displays the Markov chain sampling history
MAXINDEXPRINT=	Specifies the maximum number of observation indices to print in models with missing data
MAXSUBVALUEPRINT=	Specifies the maximum number of subject values to print in the "Random Effects Parameters" table
REOBSINFO	Displays more detailed information about each random effect
SCALE=	Specifies the initial scale applied to the proposal distribution
TARGACCEPT=	Specifies the target acceptance rate for the random-walk sampler
TARGACCEPTI=	Specifies the target acceptance rate for the independence sampler
TUNEWT=	Specifies the weight used in covariance updating
Summary, Diagnostics, and Plotting Options	
AUTOCORLAG=	Specifies the number of autocorrelation lags used to compute effective sample sizes and Monte Carlo errors
DIAGNOSTICS=	Controls the convergence diagnostics
DIC	Computes the deviance information criterion (DIC)
MONITOR=	Outputs the analysis for a list of symbols of interest
PLOTS=	Controls plotting
STATISTICS=	Controls posterior statistics

Table 80.1 *continued*

Option	Description
Other Options	
INF=	Specifies the machine numerical limit for infinity
JOINTMODEL	Specifies the joint log-likelihood function
MISSING=	Indicates how missing values are handled
NOLOGDIST	Omits the calculation of the logarithm of the joint distribution of the parameters
SIMREPORT=	Controls the frequency of the report for the expected run time
SINGDEN=	Specifies the singularity tolerance

These *options* are described in alphabetical order.

ACCEPTTOL=*n*
specifies a tolerance for acceptance probabilities. By default, ACCEPTTOL=0.075.

ALG=*value*
PROPDIST=*value*
specifies the default sampling algorithm for continuous parameters when more optimal algorithms, such as conjugate samplers, are not available. For more information, see the sections "Hamiltonian Monte Carlo Sampler" on page 161 and "Metropolis and Metropolis-Hastings Algorithms" on page 156 in Chapter 8, "Introduction to Bayesian Analysis Procedures." By default, ALG=NORMAL, a normal kernel based random-walk Metropolis.

You can specify the following *values*:

HMC< (*hmc-options***) >**
specifies the Hamiltonian Monte Carlo algorithm with a fixed step size and predetermined number of steps. You can specify the following *hmc-options*:

NSTEPS=*value*
N=*value*
specifies the number of steps in the HMC algorithm. By default, N=15.

SAVEGRAD
saves the gradient calculation in the OUTPOST= data set.

STEPSIZE=*value*
specifies the step size in the HMC algorithm. By default, STEPSIZE=0.1.

NORMAL
N
specifies a normal distribution as the proposal distribution in the random-walk Metropolis algorithm. This is the default.

NUTS< (*nuts-options*) >
specifies the No-U-Turn Sampler of the Hamiltonian algorithm. You can specify the following *nuts-options*:

DELTA=*value*
specifies the target acceptance rate during the tuning process. By default, DELTA=0.6. Increasing the *value* can often improve mixing, but it can also significantly slow down the sampling.

FCALLS
outputs the number of function evaluations at each iteration.

MAXHEIGHT=*value*
specifies the maximum height of the NUTS tree. The taller the tree, the more gradient evaluations per iteration the procedure calculates. The number of evaluations is 2^{height}. By default, MAXHEIGHT=10. Usually, the height of a tree should be no more than 7 or 8 during the sampling stage, but it can go higher during the tuning stage. A larger number indicates that the algorithm is having difficulty converging. PROC MCMC stops when the height of a NUTS tree surpasses MAXHEIGHT by the number of times specified in the MAXTIME= option. You can increase the height of the tree and the MAXTIME value.

MAXTIME=*value*
specifies the maximum number of iterations that it takes the algorithm to surpass the MAXHEIGHT of the NUTS tree before the procedure stops. By default, MAXTIME=1.

NTU=*value*
specifies the number of tuning iterations used by NUTS. By default, NTU=1000.

SAVEGRAD
saves the gradient calculation in the OUTPOST= data set.

T< (*df*) >
specifies a *t* distribution with the degrees of freedom *df* in the random-walk Metropolis algorithm. By default, *df* = 3. If *df* > 100, the normal distribution is used, because the two distributions are almost identical.

AUTOCORLAG=*n*

ACLAG=*n*
specifies the maximum number of autocorrelation lags used in computing the effective sample size; see the section "Effective Sample Size" on page 175 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. The value is used in the calculation of the Monte Carlo standard error; see the section "Standard Error of the Mean Estimate" on page 176 in Chapter 8, "Introduction to Bayesian Analysis Procedures." By default, AUTOCORLAG=MIN(500, MCsample/4), where MCsample is the Markov chain sample size kept after thinning—that is, $\text{MCsample} = \left[\frac{\text{NMC}}{\text{NTHIN}}\right]$. If the value of the AUTOCORLAG= option is set too low, you might observe significant lags, and the effective sample size cannot be calculated accurately. A warning message appears, and you can increase either AUTOCORLAG= or NMC=, accordingly.

BINARYJOINT
jointly samples binary parameters in a block.

DISCRETE=*keyword*
specifies the proposal distribution used in sampling discrete parameters. By default, DISCRETE=BINNING.

You can specify the following *keywords*:

BINNING
uses continuous proposal distributions for all discrete parameter blocks. The proposed sample is then discretized (binned) before further calculations. This sampling method approximates the correlation structure among the discrete parameters in the block and could improve mixing in some cases.

GEO
uses independent symmetric geometric proposal distributions for all discrete parameter blocks. This proposal does not take parameter correlations into account. However, it can work better than the BINNING option in cases where the range of the parameters is relatively small and a normal approximation can perform poorly.

DIAGNOSTICS=NONE | (*keyword-list***)**
DIAG=NONE | (*keyword-list***)**
specifies options for MCMC convergence diagnostics. By default, PROC MCMC computes the Geweke test, sample autocorrelations, effective sample sizes, and Monte Carlo errors. The Raftery-Lewis and Heidelberger-Welch tests are also available. See the section "Assessing Markov Chain Convergence" on page 162 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details on convergence diagnostics. You can request all of the diagnostic tests by specifying DIAGNOSTICS=ALL. You can suppress all the tests by specifying DIAGNOSTICS=NONE.

You can use postprocessing autocall macros to calculate convergence diagnostics of the posterior samples after PROC MCMC has exited. See the section "Autocall Macros for Postprocessing" on page 6329.

The following *options* are available.

ALL
computes all diagnostic tests and statistics. You can combine the option ALL with any other specific tests to modify test options. For example DIAGNOSTICS=(ALL AUTOCORR(LAGS=(1 5 35))) computes all tests with default settings and autocorrelations at lags 1, 5, and 35.

AUTOCORR < (*autocorr-options***) >**
computes default autocorrelations at lags 1, 5, 10, and 50 for each variable. You can choose other lags by using the following *autocorr-options*:

LAGS | AC=*numeric-list*
specifies autocorrelation lags. The *numeric-list* must take positive integer values.

ESS

computes the effective sample sizes (Kass et al. (1998)) of the posterior samples of each parameter. It also computes the correlation time and the efficiency of the chain for each parameter. Small values of ESS might indicate a lack of convergence. See the section "Effective Sample Size" on page 175 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details.

GEWEKE < (*Geweke-options*) >

computes the Geweke spectral density diagnostics; this is a two-sample t-test between the first f_1 portion and the last f_2 portion of the chain. See the section "Geweke Diagnostics" on page 169 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. The default is FRAC1=0.1 and FRAC2=0.5, but you can choose other fractions by using the following *Geweke-options*:

FRAC1 | F1=*value*

specifies the beginning FRAC1 proportion of the Markov chain. By default, FRAC1=0.1.

FRAC2 | F2=*value*

specifies the end FRAC2 proportion of the Markov chain. By default, FRAC2=0.5.

HEIDELBERGER | HEIDEL < (*Heidel-options*) >

computes the Heidelberger and Welch diagnostic (which consists of a stationarity test and a halfwidth test) for each variable. The stationary diagnostic test tests the null hypothesis that the posterior samples are generated from a stationary process. If the stationarity test is passed, a halfwidth test is then carried out. See the section "Heidelberger and Welch Diagnostics" on page 170 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details.

These diagnostics are not performed by default. You can specify the DIAGNOSTICS=HEIDELBERGER option to request these diagnostics, and you can also specify suboptions, such as DIAGNOSTICS=HEIDELBERGER(EPS=0.05), as follows:

SALPHA=*value*

specifies the α level ($0 < \alpha < 1$) for the stationarity test. By default, SALPHA=0.05.

HALPHA=*value*

specifies the α level ($0 < \alpha < 1$) for the halfwidth test. By default, HALPHA=0.05.

EPS=*value*

specifies a small positive number ϵ such that if the halfwidth is less than ϵ times the sample mean of the retaining iterates, the halfwidth test is passed. By default, EPS=0.1.

MCSE

MCERROR

computes the Monte Carlo standard error for the posterior samples of each parameter.

NONE

suppresses all of the diagnostic tests and statistics. This is not recommended.

RAFTERY | RL <(*Raftery-options*)>
computes the Raftery and Lewis diagnostics, which evaluate the accuracy of the estimated quantile ($\hat{\theta}_Q$ for a given Q $\in (0,1)$) of a chain. $\hat{\theta}_Q$ can achieve any degree of accuracy when the chain is allowed to run for a long time. The algorithm stops when the estimated probability $\hat{P}_Q = \Pr(\theta \leq \hat{\theta}_Q)$ reaches within $\pm$R of the value Q with probability S; that is, $\Pr(\text{Q} - \text{R} \leq \hat{\text{P}}_Q \leq \text{Q} + \text{R}) = \text{S}$. See the section "Raftery and Lewis Diagnostics" on page 172 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. The *Raftery-options* enable you to specify Q, R, S, and a precision level ϵ for a stationary test.

These diagnostics are not performed by default. You can specify the DIAGNOSTICS=RAFERTY option to request these diagnostics, and you can also specify suboptions, such as DIAGNOSTICS=RAFERTY(QUANTILE=0.05), as follows:

QUANTILE | Q=*value*
specifies the order (a value between 0 and 1) of the quantile of interest. By default, QUANTILE=0.025.

ACCURACY | R=*value*
specifies a small positive number as the margin of error for measuring the accuracy of estimation of the quantile. By default, ACCURACY=0.005.

PROB | S=*value*
specifies the probability of attaining the accuracy of the estimation of the quantile. By default, PROB=0.95.

EPS=*value*
specifies the tolerance level (a small positive number) for the stationary test. By default, EPS=0.001.

DIC
computes the Deviance Information Criterion (DIC). DIC is calculated using the posterior mean estimates of the parameters. See the section "Deviance Information Criterion (DIC)" on page 177 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details.

DATA=*SAS-data-set*
specifies the input data set. Observations in this data set are used to compute the log-likelihood function that you specify with PROC MCMC statements.

INF=*value*
specifies the numerical definition of infinity in PROC MCMC. The default is INF=1E15. For example, PROC MCMC considers 1E16 to be outside of the support of the normal distribution and assigns a missing value to the log density evaluation. You can select a larger value with the INF= option. The minimum value allowed is 1E10.

INIT=(*keyword-list***)**
specifies options for generating the initial values for the parameters. These options apply only to prior distributions that are recognized by PROC MCMC. See the section "Standard Distributions" on page 6275 for a list of these distributions. If either of the functions GENERAL or DGENERAL is used, you must supply explicit initial values for the parameters. By default, INIT=MODE. The following *keywords* are used:

MODE
uses the mode of the prior density as the initial value of the parameter, if you did not provide one. If the mode does not exist or if it is on the boundary of the support of the density, the mean value is used. If the mean is outside of the support or on the boundary, which can happen if the prior distribution is truncated, a random number drawn from the prior is used as the initial value.

PINIT
tabulates parameter values after the tuning phase. This option also tabulates the tuned proposal parameters used by the Metropolis algorithm. These proposal parameters include covariance matrices for continuous parameters and probability vectors for discrete parameters for each block. By default, PROC MCMC does not display the initial values or the tuned proposal parameters after the tuning phase.

RANDOM
generates a random number from the prior density and uses it as the initial value of the parameter, if you did not provide one.

REINIT
resets the parameters, after the tuning phase, with the initial values that you provided explicitly or that were assigned by PROC MCMC. By default, PROC MCMC does not reset the parameters because the tuning phase usually moves the Markov chains to a more favorable place in the posterior distribution.

LIST
displays the model program and variable lists. The LIST option is a debugging feature and is not normally needed.

LISTCODE
displays the compiled program code. The LISTCODE option is a debugging feature and is not normally needed.

JOINTMODEL

JOINTLLIKE
specifies how the likelihood function is calculated. By default, PROC MCMC assumes that the observations in the data set are independent so that the joint log-likelihood function is the sum of the individual log-likelihood functions for the observations, where the individual log-likelihood function is specified in the MODEL statement. When your data are not independent, you can specify the JOINTMODEL option to modify the way that PROC MCMC computes the joint log-likelihood function. In this situation, PROC MCMC no longer steps through the input data set to sum the individual log likelihood.

To use this option correctly, you need to do the following two things:

- create ARRAY symbols to store all data set variables that are used in the program. This can be accomplished with the BEGINCNST and ENDCNST statements.
- program the joint log-likelihood function by using these ARRAY symbols only. The MODEL statement specifies the joint log-likelihood function for the entire data set. Typically, you use the function GENERAL in the MODEL statement.

See the sections "BEGINCNST/ENDCNST Statement" on page 6234 and "Modeling Joint Likelihood" on page 6304 for details.

MAXTUNE=*n*

specifies an upper limit for the number of proposal tuning loops. By default, MAXTUNE=24. See the section "Covariance Tuning" on page 6270 for more details.

MAXINDEXPRINT=*number* | **ALL**

MAXIPRINT=*number* | **ALL**

specifies the maximum number of observation indices to print in the ODS tables "Missing Response Information" table and "Missing Covariates Information" table. This option applies only to programs that model missing data. The default value is 20. MAXINDEXPRINT=ALL prints all observation indices for every missing variable that is modeled in PROC MCMC.

MAXSUBVALUEPRINT=*number* | **ALL**

MAXSVPRINT=*number* | **ALL**

specifies the maximum number of subject values to display in the "Subject Values" column of the ODS table "Random Effects Parameters." This option applies only to programs that have RANDOM statements. The default value is 20. MAXSUBVALUEPRINT=ALL prints all subject values for every random effect in the program.

MCHISTORY=*keyword*

MCHIST=*keyword*

controls the display of the Markov chain sampling history.

BRIEF

produces a summary output for the tuning, burn-in, and sampling history tables. The tables show the following when applicable:

- "RWM Scale" shows the scale, or the range of the scales, used in each random walk Metropolis block that is normal or is based on a *t* distribution.
- "Probability" shows the proposal probability parameter, or the range of the parameters, used in each random walk Metropolis block that is based on a geometric distribution.
- "RWM Acceptance Rate" shows the acceptance rate, or the range of the acceptance rates, for each random walk Metropolis block.
- "IM Acceptance Rate" shows the acceptance rate, or the range of the acceptance rates, for each independent Metropolis block.

DETAILED

produces detailed output of the tuning, burn-in, and sampling history tables, including scale values, acceptance probabilities, blocking information, and so on. Use this option with caution, especially in random-effects models that have a large number of random-effects groups. This option can produce copious output.

NONE

produces none of the tuning history, burn-in history, and sampling history tables.

The default is MCHISTORY=NONE.

MINTUNE=*n*

specifies a lower limit for the number of proposal tuning loops. By default, MINTUNE=2. See the section "Covariance Tuning" on page 6270 for more details.

MISSING=*keyword*

MISS=*keyword*

specifies how to handle missing values. For more information, see the section "Handling of Missing Data" on page 6338. By default, MISSING=CCMODELY. PROC MCMC models missing response variables and discard observations with missing covariates.

AC | ALLCASE

enables you to directly model the missing values in an all-cases analysis and does not attempt to model the missing response values. You can use any techniques that you want to deal with the missing values (for example, you can use fully Bayesian or multiple imputation).

ACMODELY

models the variable in missing responses without discarding observations that have missing covariates. This option is useful in modeling censored data.

CC | COMPLETECASE

assumes a complete case analysis, so all observations that have missing variable values are discarded prior to the simulation.

CCMODELY

models the missing response variables and discards observations that have missing covariates.

By default, MISSING=CCMODELY.

MONITOR= (*symbol-list***)**

outputs analysis for selected symbols of interest in the program. The symbols can be any of the following: model parameters (symbols in the PARMS statement), secondary parameters (assigned using the operator "="), the log of the posterior density (LOGPOST), the log of the prior density (LOGPRIOR), the log of the hyperprior density (LOGHYPER) if the HYPER statement is used, or the log of the likelihood function (LOGLIKE). You can use the keyword _PARMS_ as a shorthand for all of the model parameters. PROC MCMC performs only posterior analyses (such as plotting, diagnostics, and summaries) on the symbols selected with the MONITOR= option. You can also choose to monitor an entire array by specifying the name of the array. By default MONITOR=_PARMS_.

Posterior samples of any secondary parameters listed in the MONITOR= option are saved in the OUTPOST= data set. Posterior samples of model parameters are always saved to the OUTPOST= data set, regardless of whether they appear in the MONITOR= option.

NBI=*n*

specifies the number of burn-in iterations to perform before beginning to save parameter estimate chains. By default, NBI=1000. See the section "Burn-In, Thinning, and Markov Chain Samples" on page 162 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details.

NMC=*n*

specifies the number of iterations in the main simulation loop. This is the MCMC sample size if THIN=1. By default, NMC=1000.

NOLOGDIST

omits the calculation of the logarithm of the joint distribution of the model parameters at each iteration. The option applies only if all parameters in the model are updated directly from their target distribution, either from the full conditional posterior via conjugacy or from the marginal distribution. Such algorithms do not require the calculation of the joint posterior distribution; hence PROC MCMC runs faster by avoiding these unnecessary calculations. As a result, the OUTPOST= data set does not contain the LOGPRIOR, LOGLIKE, and LOGPOST variables.

NTHREADS=*n*

specifies the number of threads for simulation. PROC MCMC performs two types of threading. In sampling model parameters, PROC MCMC allocates data to different threads and calculates the objective function by accumulating values from each thread; in sampling of random-effects parameters and missing data variables, each thread generates a subset of these parameters simultaneously at each iteration. Most sampling algorithms are threaded. NTHREADS=–1 sets the number of available threads to the number of hyperthreaded cores available on the system. By default, NTHREADS=1.

NTU=*n*

specifies the number of iterations to use in each proposal tuning phase. By default, NTU=500.

OUTPOST=*SAS-data-set*

specifies an output data set that contains the posterior samples of all model parameters, the iteration numbers (variable name ITERATION), the log of the posterior density (LOGPOST), the log of the prior density (LOGPRIOR), the log of the hyperprior density (LOGHYPER), if the HYPER statement is used, and the log likelihood (LOGLIKE). Any secondary parameters (assigned using the operator "=") listed in the MONITOR= option are saved to this data set. By default, no OUTPOST= data set is created.

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

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

controls the display of diagnostic plots. Three types of plots can be requested: trace plots, autocorrelation function plots, and kernel density plots. By default, the plots are displayed in panels unless the global plot option UNPACK is specified. Also when more than one type of plot is specified, the plots are grouped by parameter unless the global plot option GROUPBY=TYPE is specified. When you specify only one plot request, you can omit the parentheses around the plot-request, as shown in the following example:

```
plots=none
plots(unpack)=trace
plots=(trace density)
```

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

```
ods graphics on;
proc mcmc data=exi seed=7 outpost=p1 plots=all;
   parm mu;
```

```
   prior mu ~ normal(0, sd=10);
   model y ~ normal(mu, sd=1);
run;
ods graphics off;
```

For more information about enabling and disabling ODS Graphics, see the section "Enabling and Disabling ODS Graphics" on page 687 in Chapter 24, "Statistical Graphics Using ODS."

If ODS Graphics is enabled but you do not specify the PLOTS= option, then PROC MCMC produces, for each parameter, a panel that contains the trace plot, the autocorrelation function plot, and the density plot. This is equivalent to specifying PLOTS=(TRACE AUTOCORR DENSITY).

The *global-plot-options* include the following:

FRINGE
adds a fringe plot to the horizontal axis of the density plot.

GROUPBY|GROUP=PARAMETER | TYPE
specifies how the plots are grouped when there is more than one type of plot. GROUPBY=PARAMETER is the default. The choices are as follows:

TYPE
specifies that the plots are grouped by type.

PARAMETER
specifies that the plots are grouped by parameter.

LAGS=*n*
specifies the number of autocorrelation lags used in plotting the ACF graph. By default, LAGS=50.

SMOOTH
smooths the trace plot with a fitted penalized B-spline curve (Eilers and Marx 1996).

UNPACKPANEL

UNPACK
specifies that all paneled plots are to be unpacked, so that each plot in a panel is displayed separately.

The *plot-requests* are as follows:

ALL
requests all types of plots. PLOTS=ALL is equivalent to specifying PLOTS=(TRACE AUTOCORR DENSITY).

AUTOCORR | ACF
displays the autocorrelation function plots for the parameters.

DENSITY | D | KERNEL | K
displays the kernel density plots for the parameters.

NONE
suppresses the display of all plots.

TRACE | T
displays the trace plots for the parameters.

Consider a model with four parameters, X1–X4. Displays for various specifications are depicted as follows.

- PLOTS=(TRACE AUTOCORR) displays the trace and autocorrelation plots for each parameter side by side with two parameters per panel:

Display 1	Trace(X1)	Autocorr(X1)
	Trace(X2)	Autocorr(X2)
Display 2	Trace(X3)	Autocorr(X3)
	Trace(X4)	Autocorr(X4)

- PLOTS(GROUPBY=TYPE)=(TRACE AUTOCORR) displays all the paneled trace plots, followed by panels of autocorrelation plots:

Display 1	Trace(X1)	
	Trace(X2)	
Display 2	Trace(X3)	
	Trace(X4)	
Display 3	Autocorr(X1)	Autocorr(X2)
	Autocorr(X3)	Autocorr(X4)

- PLOTS(UNPACK)=(TRACE AUTOCORR) displays a separate trace plot and a separate correlation plot, parameter by parameter:

Display 1	Trace(X1)
Display 2	Autocorr(X1)
Display 3	Trace(X2)
Display 4	Autocorr(X2)
Display 5	Trace(X3)
Display 6	Autocorr(X3)
Display 7	Trace(X4)
Display 8	Autocorr(X4)

- PLOTS(UNPACK GROUPBY=TYPE)=(TRACE AUTOCORR) displays all the separate trace plots followed by the separate autocorrelation plots:

Display 1	Trace(X1)
Display 2	Trace(X2)
Display 3	Trace(X3)
Display 4	Trace(X4)
Display 5	Autocorr(X1)
Display 6	Autocorr(X2)
Display 7	Autocorr(X3)
Display 8	Autocorr(X4)

PROPCOV=*value*

specifies the method used in constructing the initial covariance matrix for the Metropolis-Hastings algorithm. The QUANEW and NMSIMP methods find numerically approximated covariance matrices at the optimum of the posterior density function with respect to all continuous parameters. The optimization does not apply to discrete parameters. The tuning phase starts at the optimized values; in some problems, this can greatly increase convergence performance. If the approximated covariance matrix is not positive definite, then an identity matrix is used instead. Valid values are as follows:

IND

uses the identity covariance matrix. This is the default. See the section "Tuning the Proposal Distribution" on page 6269.

CONGRA< (*optimize-options***) >**

performs a conjugate-gradient optimization.

DBLDOG< (*optimize-options***) >**

performs a double-dogleg optimization.

QUANEW< (*optimize-options***) >**

performs a quasi-Newton optimization.

NMSIMP | SIMPLEX< (*optimize-options***) >**

performs a Nelder-Mead simplex optimization.

The *optimize-options* are as follows:

ITPRINT

prints optimization iteration steps and results.

REOBSINFO <(*display-options*)>

displays the ODS table "Random Effect Observation Information." The table lists the name of each random effect, the unique values in the corresponding subject variable, the number of observations in each subject, and the observation indices for each subject value.

To understand how this option works, consider the following statements:

```
data input;
   array names{*} $ n1-n10 ("John" "Mary" "Chris" "Rob" "Greg"
                    "Jen" "Henry" "Alice" "James" "Toby");
   call streaminit(17);
   do i = 1 to 20;
      j = ceil(rand("uniform") * 10 );
      index = names[j];
      output;
   end;
   drop n: j;
run;

proc print data=input;
run;
```

The input data set (Figure 80.11) contains the index variable, which indicates subjects in a hypothetical random-effects model.

Figure 80.11 Subject Variable in an Input Data Set

Obs	i	index
1	1	Mary
2	2	James
3	3	Mary
4	4	Greg
5	5	Chris
6	6	James
7	7	James
8	8	Chris
9	9	James
10	10	James
11	11	Chris
12	12	Rob
13	13	Rob
14	14	Greg
15	15	Greg
16	16	Alice
17	17	Jen
18	18	Alice
19	19	John
20	20	Chris

The following statements illustrate the use of the REOBSINFO option:

```
ods select reobsinfo;
proc mcmc data=input reobsinfo stats=none diag=none;
   random u ~ normal(0, sd=1) subject=index;
   model general(0);
run;
```

Figure 80.12 displays the "Random Effect Observation Information" table. The table contains the name of the random-effect parameter (u), the values of the subject variable index, the total number of observations, and the row index of these observations in each of the subject values.

Figure 80.12 Random Effect Observation Information

The MCMC Procedure

Random Effect Observation Information

Parameter	Subject Values	Number of Observations in Subject	Observation Indices
u	Mary	2	1 3
	James	5	2 6 7 9 10
	Greg	3	4 14 15
	Chris	4	5 8 11 20
	Rob	2	12 13
	Alice	2	16 18
	Jen	1	17
	John	1	19

The *display-options* are as follows:

MAXVALUEPRINT=*number* | **ALL**

MAXVPRINT=*number* | **ALL**

prints the number of subject values for each random effect (that is, the number of rows that are displayed in the "Random Effect Observation Information" table for each random effect). The default value is 20. MAXVALUEPRINT=ALL displays all subject values.

MAXOBSPRINT=*number* | **ALL**

MAXOPRINT=*number* | **ALL**

prints the number of observation indices for each subject value of every random effect (that is, the maximum number of indices that are displayed in the "Observation Indices" column in the "Random Effect Observation Information" table). The default value is 20. MAXOBSPRINT=ALL displays indices for every subject value.

SCALE=*value*

controls the initial multiplicative scale to the covariance matrix of the proposal distribution. By default, SCALE=2.38. See the section "Scale Tuning" on page 6269 for more details.

SEED=*n*

specifies the random number seed. By default, SEED=0, and PROC MCMC gets a random number seed from the clock.

SIMREPORT=*n*

controls the number of times that PROC MCMC reports the expected run time of the simulation. This can be useful for monitoring the progress of CPU-intensive programs. For example, with SIMREPORT=2, PROC MCMC reports the simulation progress twice. By default, SIMREPORT=0, and there is no reporting. The expected run times are displayed in the log file.

SINGDEN=*value*

defines the singularity criterion in PROC MCMC. By default, SINGDEN=1E-11. The *value* indicates the exclusion of an endpoint in an interval. The mathematical notation "(0" is equivalent to "[*value*" in PROC MCMC—that is, $x < 0$ is treated as $x \leq$ *value* in PROC MCMC. The maximum SINGDEN allowed is 1E-6.

STATISTICS< (*global-options***) > = NONE | ALL |***stats-request***
STATS< (*global-options***) > = NONE | ALL |***stats-request*

specifies options for posterior statistics. By default, PROC MCMC computes the posterior mean, standard deviation, quantiles, and two 95% credible intervals: equal-tail and highest posterior density (HPD). Other available statistics include the posterior correlation and covariance. See the section "Summary Statistics" on page 175 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. You can request all of the posterior statistics by specifying STATS=ALL. You can suppress all the calculations by specifying STATS=NONE.

You can use postprocessing autocall macros to calculate posterior summary statistics of the posterior samples after PROC MCMC has exited. See the section "Autocall Macros for Postprocessing" on page 6329.

You can specify the following *global-options* to display interval and percentile estimates:

ALPHA=*numeric-list*

specifies the α level for the equal-tail and HPD intervals. The value α must be between 0 and 0.5. By default, ALPHA=0.05.

PERCENTAGE | PERCENT=*numeric-list*

calculates the posterior percentages. The *numeric-list* contains values between 0 and 100. By default, PERCENTAGE=(25 50 75).

You can specify the following *stats-requests*:

ALL

computes all posterior statistics. You can combine the option ALL with any other options. For example STATS(ALPHA=(0.02 0.05 0.1))=ALL computes all statistics with the default settings and intervals at α levels of 0.02, 0.05, and 0.1.

BRIEF

computes the posterior means, standard deviations, and the $100(1-\alpha)\%$ equal-tail intervals for each variable.

CORR

computes the posterior correlation matrix.

COV

computes the posterior covariance matrix.

INTERVAL
INT

computes the $100(1-\alpha)\%$ equal-tail and HPD credible intervals for each variable. For more information, see the sections "Equal-Tail Credible Interval" on page 177 in Chapter 8, "Introduction to Bayesian Analysis Procedures," and "Highest Posterior Density (HPD) Interval" on page 177 in Chapter 8, "Introduction to Bayesian Analysis Procedures." By default, $\alpha = 0.05$, but you can use the ALPHA= *global-option* to request other intervals of any probabilities.

NONE
suppresses all of the statistics.

SUMMARY

SUM
computes the posterior means, standard deviations, and percentile points for each variable. By default, the 25th, 50th, and 75th percentile points are produced, but you can use the global PERCENT= option to request specific percentile points.

TARGACCEPT=*value*
specifies the target acceptance rate for the random walk Metropolis algorithm. For more information, see the section "Metropolis and Metropolis-Hastings Algorithms" on page 156 in Chapter 8, "Introduction to Bayesian Analysis Procedures." The numeric *value* must be between 0.01 and 0.99. By default, TARGACCEPT=0.45 for models that have one parameter; TARGACCEPT=0.35 for models that have two, three, or four parameters; and TARGACCEPT=0.234 for models that have more than four parameters (Roberts, Gelman, and Gilks 1997; Roberts and Rosenthal 2001).

TARGACCEPTI=*value*
specifies the target acceptance rate for the independence sampler algorithm. The independence sampler is used for blocks of binary parameters. For more information, see the section "Independence Sampler" on page 160 in Chapter 8, "Introduction to Bayesian Analysis Procedures." The numeric *value* must be between 0 and 1. By default, TARGACCEPTI=0.6.

THIN=*n*

NTHIN=*n*
controls the thinning rate of the simulation. PROC MCMC keeps every *n*th simulation sample and discards the rest. All the posterior statistics and diagnostics are calculated using the thinned samples. By default, THIN=1. For more information, see the section "Burn-In, Thinning, and Markov Chain Samples" on page 162 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

TRACE
displays the result of each operation in each statement in the model program as it is executed. This debugging option is very rarely needed, and it produces voluminous output. If you use this option, also specify small numbers in the NMC=, NBI=, MAXTUNE=, and NTU= options.

TUNEWT=*value*
specifies the multiplicative weight used in updating the covariance matrix of the proposal distribution. The numeric *value* must be between 0 and 1. By default, TUNEWT=0.75. For more information, see the section "Covariance Tuning" on page 6270.

ARRAY Statement

ARRAY *arrayname* **[** *dimensions* **]** < **$** > < *variables-and-constants* > **;**

The ARRAY statement associates a name (of no more than eight characters) with a list of variables and constants. The ARRAY statement is similar to, but not the same as, the ARRAY statement in the DATA step, and it is the same as the ARRAY statements in the NLIN, NLP, NLMIXED, and MODEL procedures. The array name is used with subscripts in the program to refer to the array elements, as illustrated in the following statements:

```
array r[8] r1-r8;

do i = 1 to 8;
   r[i] = 0;
end;
```

The ARRAY statement does not support all the features of the ARRAY statement in the DATA step. Implicit indexing of variables 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.

Both variables and constants can be array elements. Constant array elements cannot have values assigned to them while variables can. Both the dimension specification and the list of elements are optional, but at least one must be specified. When the list of elements is not specified or fewer elements than the size of the array are listed, array variables are created by appending element numbers to the array name to complete the element list. You can index array elements by enclosing a subscript in braces ({ }) or brackets ([ ]), but not in parentheses (( )). The parentheses are reserved for function calls only.

For example, the following statement names an array day:

```
array day[365];
```

By default, the variables names are day1 to day365. However, since **day** is a SAS function, any subscript that uses parentheses gives you the wrong results. The expression **day(4)** returns the value 5 and does not reference the array element day4.

BEGINCNST/ENDCNST Statement

BEGINCNST ;

ENDCNST ;

The BEGINCNST and ENDCNST statements define a block within which PROC MCMC processes the programming statements only during the setup stage of the simulation. You can use the BEGINCNST and ENDCNST statements to define constants or import data set variables into arrays. Storing data in arrays enables you to work with data that are not identically distributed (see the section "Modeling Joint Likelihood" on page 6304) or to implement your own Markov chain sampler (see the section "UDS Statement" on page 6262). You can also use the BEGINCNST and ENDCNST statements to assign initial values to the parameters (see the section "Assignments of Parameters" on page 6274).

Assign Constants

Whenever you have programming statements that calculate constants that do not need to be evaluated multiple times throughout the simulation, you should put them within the BEGINCNST and ENDCNST statements. Using these statements can reduce redundant processing. For example, you can assign a constant to a symbol or fill in an array with numbers:

```
array cnst[17];
begincnst;
   offset = 17;
   do i = 1 to 17;
      cnst[i] = i * i;
   end;
endcnst;
```

During the setup process, PROC MCMC evaluates the programming statements within the BEGINCNST/ENDCNST once for each observation in the data set and ignores the statements in the rest of the simulation.

READ_ARRAY Function

Sometimes you might need to store variables, either from the current input data set or from a different data set, in arrays and use these arrays to specify your model. The READ_ARRAY function is convenient for that purpose.

The following two forms of the READ_ARRAY function are available:

rc = READ_ARRAY (*data-set***,** *array***) ;**

rc = READ_ARRAY (*data-set***,** *array* **<,** *"col-name1"***> <,** *"col-name2"***> <,** *. . .* **>) ;**

where

- *rc* returns 0 if the function is able to successfully read the data set.
- *data-set* specifies the name of the data set from which the array data is read. The value specified for *data-set* must be a character literal or a variable that contains the member name (libname.memname) of the data set to be read from.
- *array* specifies the PROC MCMC array variable into which the data is read. The value specified for *array* must be a local temporary array variable because the function might need to grow or shrink its size to accommodate the size of the data set.
- *col-name* specifies optional names for the specific columns of the data set that are read. If specified, *col-name* must be a literal string enclosed in quotation marks. In addition, *col-name* cannot be a PROC MCMC variable. If column names are not specified, PROC MCMC reads all of the columns in the data set.

When SAS translates between an array and a data set, the array is indexed as [row,column].

The READ_ARRAY function attempts to dynamically resize the array to match the dimensions of the input data set. Therefore, the array must be dynamic; that is, the array must be declared with the /NOSYMBOLS option.

For examples that use the READ_ARRAY function, see "Modeling Joint Likelihood" on page 6304, "Example 80.14: Time Independent Cox Model" on page 6437, and "Example 80.19: Implement a New Sampling Algorithm" on page 6464.

BEGINNODATA/ENDNODATA Statements

BEGINNODATA ;

ENDNODATA ;

BEGINPRIOR ;

ENDPRIOR ;

The BEGINNODATA and ENDNODATA statements define a block within which PROC MCMC processes the programming statements without stepping through the entire data set. The programming statements are executed only twice: at the first and the last observation of the data set. The BEGINNODATA and ENDNODATA statements are best used to reduce unnecessary observation-level computations. Any computations that are identical to every observation, such as transformation of parameters, should be enclosed in these statements.

At the first observation, PROC MCMC executes all programming statements, including those that are enclosed by these two statements. This enables a quick update of all the symbols enclosed by the BEGINNODATA and ENDNODATA statements. The goal is to ensure that subsequent statements (for example, the MODEL statement) use symbol values that have been calculated correctly. At the last observation, PROC MCMC executes the enclosed programming statements again and adds the log of the prior density to the log of the posterior density.

The BEGINPRIOR and ENDPRIOR statements are aliases for the BEGINNODATA and ENDNODATA statements, respectively. You can enclose PRIOR statements in the BEGINNODATA and ENDNODATA statements.

BY Statement

BY *variables* **;**

You can specify a BY statement in PROC MCMC to obtain separate analyses of observations in groups that are defined by the BY variables. When a BY statement appears, the procedure expects the input data set to be sorted in order of the BY variables. If you specify more than one BY statement, only the last one specified is used.

If your input data set is not sorted in ascending order, use one of the following alternatives:

- Sort the data by using the SORT procedure with a similar BY statement.
- Specify the NOTSORTED or DESCENDING option in the BY statement in the MCMC procedure. The NOTSORTED option does not mean that the data are unsorted but rather that the data are arranged in groups (according to values of the BY variables) and that these groups are not necessarily in alphabetical or increasing numeric order.

- Create an index on the BY variables by using the DATASETS procedure (in Base SAS software).

For more information about BY-group processing, see the "Grouping Data" section of *SAS Programmers Guide: Essentials*. For more information about the DATASETS procedure, see the discussion in the *Base SAS Procedures Guide*.

CMPTMODEL Statement

CMPTMODEL *required-options conditionally-required-options* < *options* > **;**

The CMPTMODEL statement computes predicted concentrations from a specified one-, two-, or three-compartment model. The CMPTMODEL statement includes three types of options:

- The following *required-options* are required: PCONC=, TIME=, NCOMPS=, ADMTYPE=, and PARMTYPE=.
- The following *conditionally-required-options* are conditionally required depending on the specification of the *required-options*: CL*n*=, VOL*n*=, K12=, K21=, K13=, K31=, KA=, K*n*0=, RATE=, and DURN=.
- The following *options* are optional: DOSE*n*=, SCALE*n*=, PCONC0=, PCONC2=, and PCONC3=.

You must specify the following *required-options*:

ADMTYPE=IVB | INF | ORAL
specifies the administration type. You can specify the following values:

IVB specifies the bolus type of administration.

INF specifies the infusion type of administration.

ORAL specifies the oral type of administration.

NCOMPS=1 | 2 | 3
specifies the number of compartments in a model.

PARMTYPE=1 | 2

PTYPE=1 | 2
specifies the parameterization type. You can specify the following values:

1 parameterizes the compartment model in terms of elimination and transfer rate constants.

2 parameterizes the compartment model in terms of clearance and volume constants.

PCONC=*variable*

PCONC1=*variable*
specifies the outcome variable that is the predicted concentration in the first or central compartment at each time point.

TIME=*variable*
specifies the time value at which the predicted concentrations are computed, where *variable* can be a data set variable or a defined variable.

You can combine the NCOMPS=, ADMTYPE=, and PARMTYPE= options to fit a compartment model (one-, two-, or three- compartments) with different administration methods (bolus, infusion, or oral) using different parameterizations. The PCONC= variable is the outcome that is the evaluated predicted concentration at the TIME= value. You can use this variable in model specification in your program.

You might need to specify one or more of the following *conditionally-required-options* based on what you specify in the *required-options*:

CL*n***=***variable*
specifies the clearance value for the *n*th compartment, where *n* takes the values of 1, 2, or 3. (CL is an alias for CL1.) This option is valid only in models that require clearance (PARMTYPE=2). The *variable* can be a data set variable or a defined variable.

- You must specify a CL1= option when you specify a one-compartment model (NCOMPS=1)
- You must specify CL1= and CL2= options when you specify a two-compartment model (NCOMPS=2)
- You must specify CL1=, CL2=, and CL3= options when you specify a three-compartment model (NCOMPS=3)

DURN=*variable*
specifies the duration of the infusion. This option is valid only in infusion models (ADMTYPE=INF). The *variable* can be a data set variable or a defined variable. In an infusion model, you must specify either this option or the RATE= option.

KA=*variable*
specifies the absorption rate constant. This option is valid only for the oral type of compartment models (ADMTYPE=ORAL).

K12=*variable*
specifies the transfer rate constant from the first (central) compartment to the second compartment. This option is valid only in models that require transfer rate (PARMTYPE=1). The *variable* can be a data set variable or a defined variable. You must specify this option in a two- or three-compartment model.

K21=*variable*
specifies the transfer rate constant from the second compartment to the first (central) compartment. This option is valid only in models that require transfer rate (PARMTYPE=1). The *variable* can be a data set variable or a defined variable. You must specify this option in a two- or three- compartment model.

K13=*variable*
specifies the transfer rate constant from the first (central) compartment to the third compartment. This option is valid only in models that require transfer rate (PARMTYPE=1). The *variable* can be a data set variable or a defined variable. You must specify this option in a three-compartment model.

K31=*variable*
specifies the transfer rate constant from the third compartment to the first (central) compartment. This option is valid only in models that require transfer rate (PARMTYPE=1). The *variable* can be a data set variable or a defined variable. You must specify this option in a three-compartment model.

K*n***0=***variable*
specifies the elimination rate constant for the *n*th compartment, where *n* takes the values of 1, 2, or 3. (Ke is an alias for K10.) This option is valid only in models that require elimination rate (PARMTYPE=1). The *variable* can be a data set variable or a defined variable.

- You must specify a K10= option when you specify a one-compartment model (NCOMPS=1)
- You must specify a K10= option and you can optionally specify a K20= option when you specify a two-compartment model (NCOMPS=2)
- You must specify a K10= option and you can optionally specify K20= and K30= options when you specify a three-compartment model (NCOMPS=3)

RATE=*variable*
specifies the rate of the infusion. This option is valid only in infusion models (ADMTYPE=INF). The *variable* can be a data set variable or a defined variable. In an infusion model, you must specify either this option or the DURN= option.

VOL*n***=***variable*
specifies the apparent volume of distribution of a drug in the *n*th compartment, where *n* takes the values of 1, 2, or 3. (VOL is an alias for VOL1.) This option is valid only in models that require volume (PARMTYPE=2). The *variable* can be a data set variable or a defined variable.

- You must specify a VOL1= option when you specify a one-compartment model (NCOMPS=1)
- You must specify VOL1= and VOL2= options when you specify a two-compartment model (NCOMPS=2)
- You must specify VOL1=, VOL2=, and VOL3= options when you specify a three-compartment model (NCOMPS=3)

You can also specify the following optional *options*:

DOSE*n***=***variable*
specifies the amount of the dose for the *n*th compartment, where *n* takes the values of 0, 1, 2, or 3. (DOSE is an alias for DOSE1.) The DOSE0= option specifies the amount of dose in the zero (depot) compartment, and is valid only in oral models (ADMTYPE=ORAL). The *variable* can be a data set variable or a defined variable.

- DOSE1= is optional and valid for one-, two-, and three-compartment models
- DOSE2= is optional and valid only for two-, and three-compartment models
- DOSE3= is optional and valid only for a three-compartment model

When ADMTYPE=ORAL is specified, DOSE0=1, DOSE1=0, DOSE2=0, and DOSE3=0 by default. Otherwise, DOSE1=1, DOSE2=0, and DOSE3=0 by default.

PCONC0=*variable*
specifies the outcome *variable* that is the predicted concentration of the zero (depot) compartment. This option is valid only when the ADMTYPE= ORAL option is specified.

PCONC2=*variable*
specifies the outcome *variable* that is the predicted concentration of the second compartment. This option is valid only in two- and three-compartment models.

PCONC3=*variable*
specifies the outcome *variable* that is the predicted concentration of the second compartment. This option is valid only in three-compartment models.

SCALE*n***=***variable*
specifies the scale value for the *n*th compartment. The *variable* can be a data set variable or a defined variable. SCALE0= option is valid only in oral models (ADMTYPE=ORAL).

- SCALE1= is optional and valid for one-, two-, and three-compartment models
- SCALE2= is optional and valid only for two-, and three-compartment models
- SCALE3= is optional and valid only for a three-compartment model

By default, all optional SCALE*n* values are set to 1.

For more information, see the section "Compartment Models" on page 6309.

MODEL Statement

MODEL *dependent-variable-list* ~ *distribution* < *options* > **;**

The MODEL statement specifies the conditional distribution of the data given the parameters (the likelihood function). You specify a single dependent variable or a list of dependent variables, a tilde ~, and then a distribution with its arguments. The dependent variables can be variables from the input data set or functions of the symbols in the program. You must specify the dependent variables unless you use the GENERAL function or the DGENERAL function (see the section "Specifying a New Distribution" on page 6290 for more details).

The MODEL statement assumes that the observations are independent of each other, conditional on the model parameters. If you want to model dependent data—that is, $f(y_i|\theta, y_j)$ for $j \neq i$—you can use the JOINTMODEL option in the PROC MCMC statement. See the section "Modeling Joint Likelihood" on page 6304 for more details. By default, the log-likelihood value is the sum of the individual log-likelihood value for each observation.

You can specify multiple MODEL statements. You can define likelihood functions that are independent of each other. For example, in the following statements, the dependent variables y1 and y2 are independent of each other:

```
model y1 ~ normal(alpha, var=s21);
model y2 ~ normal(beta,  var=s22);
```

Alternatively, you can use marginal and conditional distributions to define a joint log-likelihood function for multiple dependent variables. For example, the following statements jointly define a distribution over (y1, y2). They specify a marginal distribution for the dependent variable y1 and a conditional distribution for the dependent variable y2:

```
model y1 ~ normal(alpha, var=s21);
model y2 ~ normal(beta * y1,  var=s22);
```

Every program must have at least one MODEL statement. If you want to run a Monte Carlo simulation that does not require a response variable, use the GENERAL function in the MODEL statement:

```
model general(0);
```

PROC MCMC interprets the statement as a flat likelihood function with a constant log-likelihood value of 0.

PROC MCMC is a programming language that is similar to the DATA step, and the order of statement evaluation is important. For example, the MODEL statement must come after any SAS programming statements that define or modify arguments used in the construction of the log likelihood. In PROC MCMC, a symbol can be defined multiple times and used at different places. Using an expression out of order produces erroneous results that can also be hard to detect.

Do not embed the MODEL statement within programming statements. For example, suppose you have three response variables, y1, y2, and y3, and want to model each with a normal distribution. The following statements lead to erroneous output:

```
array Y[3] y1 y2 y3;
do i = 1 to 3;
   model y[i] ~ normal(mu, sd=s);
end;
```

Instead, you should do one of the following.

- Use separate MODEL statements:

  ```
  model y1 ~ normal(mu, sd=s);
  model y2 ~ normal(mu, sd=s);
  model y3 ~ normal(mu, sd=s);
  ```

- Use the GENERAL function to construct a joint distribution of the three dependent variables and use a single MODEL statement to specify the log-likelihood function:

  ```
  llike = logpdf("normal", y1, mu, s) +
          logpdf("normal", y2, mu, s) +
          logpdf("normal", y3, mu, s);
  model y1 y2 y3 ~ general(llike);
  ```

  See the section "Specifying a New Distribution" on page 6290 for more information about how to use the GENERAL function to specify an arbitrary distribution.

Missing data are allowed in the response variables; the MODEL statement augments missing data automatically. In each iteration, PROC MCMC samples missing values from their posterior distributions and

incorporates them as part of the simulation. PROC MCMC creates one variable for each missing response value. There are two ways to create the missing value variable names; see the NAMESUFFIX= option for the naming convention of the variables.

Distributions in MODEL Statement

Standard distributions that the MODEL statement supports are listed in the Table 80.2 (univariate) and Table 80.3 (multivariate). See the section "Standard Distributions" on page 6275 for density specifications. You can also specify all distributions except the multinomial distribution in the PRIOR and HYPERPRIOR statements. The RANDOM statement supports only a subset of the distributions (see Table 80.4).

PROC MCMC allows some distributions to be parameterized in multiple ways. For example, you can specify a normal distribution with a variance, standard deviation, or precision parameter. For distributions that have different parameterizations, you must specify an option to clearly name the ambiguous parameter. For example, in the normal distribution, you must indicate whether the second argument represents variance, standard deviation, or precision.

All univariate distributions, except for binary and uniform distributions, can have the optional LOWER= and UPPER= arguments (which specify a truncated density) and CLOWER= and CUPPER= arguments (which specify censoring). For more information, see the section "Truncation and Censoring" on page 6294. Truncation and censoring are not supported for multivariate distributions.

Table 80.2 Univariate Distributions

Distribution Name	Definition
beta(<**a=**>α, <**b=**>β)	Beta distribution with shape parameters α and β
binary(<**prob**|**p=**>p)	Binary (Bernoulli) distribution with probability of success p. You can use the alias **bern** for this distribution.
binomial (<**n=**>n, <**prob**|**p=**>p)	Binomial distribution with count n and probability of success p
cauchy (<**location**|**loc**|**l=**>θ, <**scale**|**s=**>λ)	Cauchy distribution with location θ and scale λ
chisq(<**df=**>ν)	χ^2 distribution with ν degrees of freedom
dgeneral(*ll*)	General log-likelihood function that you construct using SAS programming statements for single or multiple discrete parameters. Also see the function **general**. The name **dlogden** is an alias for this function.
expchisq(<**df=**>ν)	Log transformation of a χ^2 distribution with ν degrees of freedom: $\theta \sim$ **chisq**$(\nu) \Leftrightarrow \log(\theta) \sim$ **expchisq**(ν). You can use the alias **echisq** for this distribution.

Table 80.2 *continued*

Distribution Name	Definition
expexpon(**scale**|**s=** λ) **expexpon**(**iscale**|**is=** λ)	Log transformation of an exponential distribution with scale or inverse-scale parameter λ: $\theta \sim$ **expon**$(\lambda) \Leftrightarrow \log(\theta) \sim$ **expexpon**(λ). You can use the alias **eexpon** for this distribution.
expGamma(<**shape**|**sp=**> a, **scale**|**s=** λ) **expGamma**(<**shape**|**sp=**> a, **iscale**|**is=** λ)	Log transformation of a gamma distribution with shape a and scale or inverse-scale λ: $\theta \sim$ **gamma**$(a, \lambda) \Leftrightarrow \log(\theta) \sim$ **expgamma**(a, λ). You can use the alias **egamma** for this distribution.
expichisq(<**df=**> ν)	Log transformation of an inverse χ^2 distribution with ν degrees of freedom: $\theta \sim$ **ichisq**$(\nu) \Leftrightarrow \log(\theta) \sim$ **expichisq**(ν). You can use the alias **eichisq** for this distribution.
expiGamma(<**shape**|**sp=**> a, **scale**|**s=** λ) **expiGamma**(<**shape**|**sp=**> a, **iscale**|**is=** λ)	Log transformation of an inverse gamma distribution with shape a and scale or inverse-scale λ: $\theta \sim$ **igamma**$(a, \lambda) \Leftrightarrow \log(\theta) \sim$ **expigamma**(a, λ). You can use the alias **eigamma** for this distribution.
expsichisq(<**df=**> ν, <**scale**|**s=**> s)	Log transformation of a scaled inverse χ^2 distribution with ν degrees of freedom and scale parameter s: $\theta \sim$ **sichisq**$(\nu) \Leftrightarrow \log(\theta) \sim$ **expsichisq**(ν). You can use the alias **esichisq** for this distribution.
expon(**scale**|**s=** λ) **expon**(**iscale**|**is=** λ)	Exponential distribution with scale or inverse-scale parameter λ
gamma(<**shape**|**sp=**> a, **scale**|**s=** λ) **gamma**(<**shape**|**sp=**> a, **iscale**|**is=** λ)	Gamma distribution with shape a and scale or inverse-scale λ
geo(<**prob**|**p=**> p)	Geometric distribution with probability p

Table 80.2 *continued*

Distribution Name	Definition
general(*ll*)	General log-likelihood function that you construct using SAS programming statements for a single or multiple continuous parameters. The argument *ll* is an expression for the log of the distribution. If there are multiple variables specified before the tilde in a MODEL, PRIOR, or HYPERPRIOR statement, *ll* is interpreted as the log of the joint distribution for these variables. Note that in the MODEL statement, the response variable specified before the tilde is just a place holder and is of no consequence; the variable must have appeared in the construction of *ll* in the programming statements. **general**(*constant*) is equivalent to a uniform distribution on the real line. You can use the alias **logden** for this distribution.
ichisq(<**df=**>ν)	Inverse χ^2 distribution with ν degrees of freedom
igamma(<**shape**|**sp=**> a, **scale**|**s=** λ) **igamma**(<**shape**|**sp=**> a, **iscale**|**is=** λ)	Inverse gamma distribution with shape a and scale or inverse-scale λ
laplace(<**location**|**loc**|**l=**> θ, **scale**|**s=** λ) **laplace**(<**location**|**loc**|**l=**> θ, **iscale**|**is=** λ)	Laplace distribution with location θ and scale or inverse-scale λ. This is also known as the *double exponential* distribution. You can use the alias **dexpon** for this distribution.
logistic(<**location**|**loc**|**l=**> a, <**scale**|**s=**> b)	Logistic distribution with location a and scale b
lognormal(<**mean**|**m=**> μ, **sd=** λ) **lognormal**(<**mean**|**m=**> μ, **var**|**v=** λ) **lognormal**(<**mean**|**m=**> μ, **prec=** λ)	Log-normal distribution with mean μ and a value of λ for the standard deviation, variance, or precision. You can use the aliases **lognormal** or **lnorm** for this distribution.
negbin(<**n=**> n, <**prob**|**p=**> p)	Negative binomial distribution with count n and probability of success p. You can use the alias **nb** for this distribution.
normal(<**mean**|**m=**> μ, **sd=** λ) **normal**(<**mean**|**m=**> μ, **var**|**v=** λ) **normal**(<**mean**|**m=**> μ, **prec=** λ)	Normal (Gaussian) distribution with mean μ and a value of λ for the standard deviation, variance, or precision. You can use the aliases **gaussian**, **norm**, or **n** for this distribution.
pareto(<**shape**|**sp=**> a, <**scale**|**s=**> b)	Pareto distribution with shape a and scale b
poisson(<**mean**|**m=**> λ)	Poisson distribution with mean λ

Table 80.2 *continued*

Distribution Name	Definition
sichisq(<**df=**> ν, <**scale**|**s=**> s)	Scaled inverse χ^2 distribution with ν degrees of freedom and scale parameter s
t(<**mean**|**m=**> μ, **sd=** λ, <**df=**> ν) **t**(<**mean**|**m=**> μ, **var**|**v=** λ, <**df=**> ν) **t**(<**mean**|**m=**> μ, **prec=** λ, <**df=**> ν)	T distribution with mean μ, standard deviation or variance or precision λ, and ν degrees of freedom
table(<**p=**> p)	Table (categorical) distribution with probability vector p. You can also use the alias **cat** for this distribution.
uniform(<**left**|**l=**> a, <**right**|**r=**> b)	Uniform distribution with range a and b. You can use the alias **unif** for this distribution.
wald(<**mean**|**m=**> μ, <**iscale**|**is=**> λ)	Wald distribution with mean parameter μ and inverse scale parameter λ. This is also known as the *Inverse Gaussian* distribution. You can use the alias **igaussian** for this distribution.
weibull(μ, c, σ)	Weibull distribution with location (threshold) parameter μ, shape parameter c, and scale parameter σ

Table 80.3 Multivariate Distributions

Distribution Name	Definition
dirichlet(<**alpha=**>α)	Dirichlet distribution with parameter vector α, where α must be a one-dimensional array of length greater than 1
iwish(<**df=**>ν, <**scale=**>S)	Inverse Wishart distribution with ν degrees of freedom and symmetric positive definite scale array S
multinom(<**p=**>p)	Multinomial distribution with probability vector p
mvn(<**mu=**>μ, <**cov=**>Σ)	Multivariate normal distribution with mean vector μ and covariance matrix Σ
MVNAR(<**mu=**>μ, **sd=** λ, <**rho=**>ρ) **MVNAR**(<**mu=**>μ, **var=** λ, <**rho=**>ρ) **MVNAR**(<**mu=**>μ, **prec=** λ, <**rho=**>ρ)	Multivariate normal distribution with mean vector μ and a covariance matrix Σ. The covariance matrix Σ is a multiple of the scale and a matrix with a first-order autoregressive structure. When RHO=0, this distribution becomes a multivariate normal distribution with shared variance.

Options for the MODEL Statement

The *options* in the MODEL statement apply when there are missing values in the response variable, or in the case of the ICOND= option, when there are lag or lead variables for the response variable. You can specify the following *options*.

ICOND=*variable-list* | *numeric-list*

specifies the initial conditions (or initial states) of the lag or lead variables for the response variable when the observation indices are out of the range. (For more information about rules of constructing lag and lead variables in PROC MCMC, see the section "Access Lag and Lead Variables" on page 6306.) For example, you can use the ICOND= option to specify the lag 1 value of the response for the first observation. This option works similarly to the ICOND= option in the RANDOM statement, except that the index is done according to observations, not a subject variable. The initial conditions can be model parameters, functions of model parameters, or constants. By default, *numeric-list* is set to 0.

The ICOND= option in a MODEL statement sets the initial conditions for all lag or lead variables (for the associated response variable) that appear in the program, not just those that appear in the MODEL statement. Suppose you have a maximum L number of lag variables and a maximum M number of lead variables of the response y in the program, and there are n observations. The program has the following variables that need to be resolved during the simulation:

$$\mathsf{y}_{L+1}, \ldots, \mathsf{y}_0, \mathsf{y}_1, \ldots, \mathsf{y}_n, \mathsf{y}_{n+1}, \ldots, \mathsf{y}_{n+M}$$

Of these variables, n are observations of y from the input data set and the remaining $L+M$ are initial conditions that are specified in the ICOND= option. In essence, the ICOND= numeric list stretches the input data set by filling in the first L and last M values. As PROC MCMC steps through the input data set, it resolves the current, lagged, and lead variables according to this stretched vector of observations.

The *variable-list* (or the *number-list*) should be of length $L+M$, which can be greater than the number of lag or lead response variables that appear in a program. Here is an example.

Suppose you want to fit an autoregressive model of order 2. And instead of two lagged values, the model requires only the second lag,

$$Y_i = A + \phi \cdot Y_{i-2} + \epsilon_i$$

where the noise is assumed to be normal. To specify this autoregressive model, you would use the statements

```
mu = A + phi * y.l2;
model y ~ normal(mu, var=s2) icond=(-2 -1);
```

where the Y_{i-2}, or the lag-2 of Y, variable is constructed by concatenating the variable name, the letter L (for "lag"), and a lag number.

This model requires two initial conditions for the lag-2 variable of Y, at the first and second observations. Therefore, the ICOND= option expects a numeric list of two values. In this example, at the first observation, the variable y.l2 is given a value of –2; at the second observation, y.l2 is given a value of –1. If you provide a partial list that contains less than the expected number of conditions, PROC MCMC fills the remaining list with the value of 0.

INITIAL=*SAS-data-set* | *constant* | *numeric-list*

specifies the initial values of the missing values. By default, PROC MCMC uses a sample average of the nonmissing values of a response variable as the starting values for all missing values in the simulation for that variable. You can use the INITIAL= option to start the Markov chain at a different place.

If you use a *SAS-data-set* to store initial values, the data set must consist of variable names that agree with the missing variable names that are used by PROC MCMC. The easiest way to find the names of the internally created variables is to run a default analysis with a very small number of simulations and check the variable names in the OUTPOST= data set. You can provide a subset of the initial values in the *SAS-data-set*, and PROC MCMC uses a default mechanism to fill in the rest of the missing initial values.

For example, the following statement creates a data set with initial values for the first three missing values of a response variable:

```
data RandomInit;
   input y_1 y_2 y_3;
   datalines;
2.3 3 -3
;
```

The following MODEL statement uses the values in the RandomInit data set as the initial values of the corresponding missing values in the model:

```
model y ~ normal(0,var=s2u) init=randominit;
```

Specifying a *constant* assigns that constant as the initial value to all missing values in that response variable. For example, the following statement assigns the value 5 to be used as an initial value for all missing y_i in the model:

```
model y ~ normal(0,var=s2u) init=5;
```

If you have a multidimensional response variable, you can provide a list of numbers that have the same length as the dimension of your response array. Each number is then given to all corresponding missing variables in order. For example, the following statement assigns the value 2 to be used as an initial value for all missing $w1_i$ and the value 3 to be used for all missing $w2_i$ in the model:

```
array w[2] w1 w2;
model w ~ mvn(mu, cov) init=(2 3);
```

MONITOR= (*symbol-list* | *number-list* | **RANDOM(***number***))**

outputs analysis for selected missing data variables. You can choose to monitor the missing values by listing the response variable names, the missing data variable names, or indices, or you can have them randomly selected by PROC MCMC.

For example, suppose that the data set contains 10 observations and the response variable y has missing values in observations 2, 3, 7, 9, and 10. To monitor all missing data variables (five in total), you specify the response variable name in the MONITOR= option:

```
model y ~ normal(0,var=s2u) monitor=(y);
```

Suppose you want to monitor the missing data variables that correspond to the missing values in observations 2, 3, and 10. You have two options: provide either a list of variable names or a list of indices.

The following statement selects monitored variables by their variable names:

```
model y ~ normal(0,var=s2u) monitor=(y_2 y_3 y_10);
```

The variable names must match the internally created variable names for each missing value. See NAMESUFFIX= option for the naming convention of the variables. By default, the names are created by concatenating the response variable with the observation index; hence you use the name_obs format to construct the names. The numbers 2, 3, and 10 are the corresponding observation indices to the missing values in the input data set.

The following statement selects monitored variables by indices:

```
model y ~ normal(0,var=s2u) monitor=(1 2 5);
```

The indices are not a list of the observation numbers, but rather the order by which the missing values appear in the data set: PROC MCMC reports back the first, the second, and the fifth missing value variables that it creates. The actual variable names that appear in the output are still y_2, y_3, and y_10, honoring the control of the NAMESUFFIX= option.

Lastly, PROC MCMC can randomly choose a subset of the variables to monitor. The following statement randomly selects 3 variables to monitor:

```
model y ~ normal(0,var=s2u) monitor=(random(3));
```

The list of the random indices is controlled by the SEED= option in the PROC MCMC statement. Therefore, the selected variables will be the same when the SEED= option is the same.

NAMESUFFIX=OBSERVATION | POSITION | ORDER

specifies how the names of the missing data variables are created. By default, the names are created by concatenating the response variable symbol, an underscore ("_"), and the observation number of the missing value.

NAMESUFFIX=OBSERVATION constructs the parameter names by appending the observation number to the response variable symbol. This is the default. NAMESUFFIX=POSITION or NAMESUFFIX=ORDER construct the parameter names by appending the numbers 1, 2, 3, and so on, where the number indicates the order in which the missing values appear in the data set.

For example, suppose you have a response variable y with 10 observations in total, of which five are missing (observations 2, 3, 7, 9, and 10). By default, PROC MCMC creates five variable names y_2, y_3, y_7, y_9, and y_10. Using NAMESUFFIX=POSITION changes the names to y_1, y_2, y_3, y_4, and y_5.

NOOUTPOST

suppresses the output of the posterior samples of missing data variables to the posterior output data set (which is specified in the OUTPOST= option in the PROC MCMC statement). In models with a large number of missing values (for example, tens of thousands), PROC MCMC can run faster if it does not save the posterior samples.

When you specify both the NOOUTPOST option and the MONITOR= option, PROC MCMC outputs the list of variables that are monitored.

The maximum number of variables that can be saved to an OUTPOST= data set is 32,767. If the total number of parameters in your model, including the number of missing data variables, exceeds the limit, the NOOUTPOST option is evoked automatically and PROC MCMC does not save the missing value draws to the posterior output data set. You can use the MONITOR= option to select a subset of the parameters to store in the OUTPOST= data set.

PARMS Statement

PARMS *name* |(*name-list*)< **=** > < **{** > *number* | *number-list* < **}** >
< *name* |(*name-list*)< **=** > < **{** > *number* | *number-list* < **}** > . . . >
< / *options* > **;**

The PARMS statement lists the names of the parameters in the model and specifies optional initial values for these parameters. These parameters are referred to as the *model parameters*. You can specify multiple PARMS statements. Each PARMS statement defines a block of parameters, and the blocked Metropolis algorithm updates the parameters in each block simultaneously. See the section "Blocking of Parameters" on page 6266 for more details. PROC MCMC generates missing initial values from the prior distributions whenever needed, as long as they are the standard distributions and not the GENERAL or DGENERAL function.

If your model contains a multidimensional parameter (for example, a parameter with a multivariate normal prior distribution), you must declare the parameter as an array (using the ARRAY statement). You can use braces { } after the parameter name in the PARM statement to assign initial values. For example:

```
array mu[3];
parms mu {1 2 3};
```

You cannot use the ARRAY statement to assign initial values. If you use the ARRAY statement to store values in array elements, the declared array becomes a constant array and cannot be used as parameters in the PARMS statement. For example, the following statement assigns three numbers to mu:

```
array mu[3] (1 2 3);
```

The array mu can no longer be a model parameter.

Every parameter in the PARMS statement must have a corresponding prior distribution in the PRIOR statement. The program exits if this one-to-one requirement is not satisfied.

You can specify the following *options* to control different samplers explicitly for that block of parameters.

NORMAL | N
uses the normal proposal distribution in the random walk Metropolis. This is the default.

T <(*df*)>
uses the t distribution with *df* degrees of freedom as an alternative proposal distribution. A t distribution with a small number of degrees of freedom has thicker tails and can sometimes improve the mixing of the Markov chain. When $df > 100$, the normal distribution is used instead.

SLICE
applies the slice sampler to each parameter in the PARMS statement individually. See the section"Slice Sampler" on page 159 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for details. PROC MCMC does not implement a multidimensional version of the slice sampler. Because the slice sampler usually requires multiple evaluations of the objective function (the posterior distribution) in each iteration, the associated computational cost could be potentially high with this sampling algorithm.

UDS
implements a user-defined sampler for any of the parameters in the block. See the section "UDS Statement" on page 6262 for details and "Example 80.19: Implement a New Sampling Algorithm" on page 6464 for a realistic example. When you specify the UDS option, PROC MCMC hands off the sampling of these parameters to you at each iteration and relies on your sampler to return a random draw from the conditional posterior distribution. This option is useful if you have a model-specific sampler that you want to implement or a new algorithm that can improve the convergence and mixing of the Markov chain. This functionality is for advanced users, and you should proceed with caution.

PREDDIST Statement

PREDDIST <'*label*'> **OUTPRED=***SAS-data-set* <**NSIM=***n*> <**COVARIATES=***SAS-data-set*>
<**STATISTICS=***options*> **;**

The PREDDIST statement creates a new SAS data set that contains random samples from the posterior predictive distribution of the response variable. The posterior predictive distribution is the distribution of unobserved observations (prediction) conditional on the observed data. Let $\mathbf{y}$ be the observed data, $\mathbf{X}$ be the covariates, θ be the parameter, and $\mathbf{y}_{\text{pred}}$ be the unobserved data. The posterior predictive distribution is defined to be the following:

$$\begin{aligned} p(\mathbf{y}_{\text{pred}}|\mathbf{y},\mathbf{X}) &= \int p(\mathbf{y}_{\text{pred}},\theta|\mathbf{y},\mathbf{X})d\theta \\ &= \int p(\mathbf{y}_{\text{pred}}|\theta,\mathbf{y},\mathbf{X})p(\theta|\mathbf{y},\mathbf{X})d\theta \end{aligned}$$

Given the assumption that the observed and unobserved data are conditional independent given θ, the posterior predictive distribution can be further simplified as the following:

$$p(\mathbf{y}_{\text{pred}}|\mathbf{y},\mathbf{X}) = \int p(\mathbf{y}_{\text{pred}}|\theta)p(\theta|\mathbf{y},\mathbf{X})d\theta$$

The posterior predictive distribution is an integral of the likelihood function $p(\mathbf{y}_{\text{pred}}|\theta)$ with respect to the posterior distribution $p(\theta|\mathbf{y})$. The PREDDIST statement generates samples from a posterior predictive distribution based on draws from the posterior distribution of θ.

The PREDDIST statement works only on response variables that have standard distributions, and it does not support either the GENERAL or DGENERAL functions. Multiple PREDDIST statements can be specified, and an optional label (specified as a quoted string) helps identify the output.

The following list explains specifications in the PREDDIST statement:

COVARIATES=*SAS-data-set*
: names the SAS data set that contains the sets of explanatory variable values for which the predictions are established. This data set must contain data with the same variable names as are used in the likelihood function. If you omit the COVARIATES= option, the DATA= data set specified in the PROC MCMC statement is used instead.

NSIM=*n*
: specifies the number of simulated predicted values. By default, NSIM= uses the NMC= option value specified in the PROC MCMC statement.

OUTPRED=*SAS-data-set*
: creates an output data set to contain the samples from the posterior predictive distribution. The output variable names are listed as resp_1–resp_*m*, where resp is the name of the response variable and *m* is the number of observations in the COVARIATES= data set in the PREDDIST statement. If the COVARIATES= data set is not specified, *m* is the number of observations in the DATA= data set specified in the PROC statement.

SAVEPARM
: outputs to the OUTPRED= data set sampled parameter values that are used in each predictive draw.

STATISTICS< **(***global-options***)** **>** **=** **NONE** | **ALL** |*stats-request*

STATS< **(***global-options***)** **>** **=** **NONE** | **ALL** |*stats-request*
: specifies options for calculating posterior statistics. This option works identically to the STATISTICS= option in the PROC statement. By default, this option takes the specification of the STATISTICS= option in the PROC MCMC statement.

For an example that uses the PREDDIST statement, see the section "Posterior Predictive Distribution" on page 6334.

PRIOR/HYPERPRIOR Statement

PRIOR *parameter-list* ∼ *distribution* **;**

HYPERPRIOR *parameter-list* ∼ *distribution* **;**

HYPER *parameter-list* ∼ *distribution* **;**

The PRIOR statement specifies the prior distribution of the model parameters. You must specify a single parameter or a list of parameters, a tilde ∼, and then a distribution with its parameters.

You can specify multiple PRIOR statements to define models with multiple prior components. Your model can have as many hierarchical levels as you want. But in many cases, such as random-effects models, it is better to use the RANDOM statements to build up the model hierarchy. The log of the prior is the sum of the log prior values from each of the PRIOR statements. Similar to the MODEL statement, you can use the PRIOR statement to specify marginal or conditional prior distributions. See the section "MODEL

Statement" on page 6240 for the names of the standard distributions and the section "Standard Distributions" on page 6275 for density specification.

The PRIOR statements are processed twice at every Markov chain simulation—that is, twice per pass through the data set. The statements are called at the first and the last observation of the data set, just as the BEGINNODATA and ENDNODATA statements are processed. If you run a Monte Carlo simulation that is data-independent, you can specify the NOLOGDIST option in the PROC MCMC statement to omit the calculation of the prior distribution. Omitting this calculation enables PROC MCMC to run faster.

The HYPERPRIOR statement is treated internally the same as the PRIOR statement. It provides a notational convenience in case you want to fit a multilevel hierarchical model. It specifies the hyperprior distribution of the prior distribution parameters. The log of the hyperprior is the sum of the log hyperprior values from each of the HYPERPRIOR statements.

Parameters in the PRIOR statements can appear as hyperparameters in the RANDOM statement. The reverse is not allowed: random-effects parameters cannot be hyperparameters in a PRIOR statement.

You can have a program that contains a RANDOM statement but no PRIOR statements. (In SAS 9.3 and earlier, each program had to contain a PRIOR statement.) A program that contains a RANDOM statement but no PRIOR statements could be a random-effects model with no fixed-effects parameters or hyperparameters to the random effects. A MODEL statement is still required in every program.

Programming Statements

This section lists the programming statements available in PROC MCMC to compute the priors and log-likelihood functions. This section also documents the differences between programming statements in PROC MCMC and programming statements in the DATA step. The syntax of programming statements used in PROC MCMC is identical to that used in the NLMIXED procedure (see Chapter 89, "The NLMIXED Procedure") and the MODEL procedure (see Chapter 24, "The MODEL Procedure" (*SAS/ETS User's Guide*)). Most of the programming statements that can be used in the DATA step can also be used in PROC MCMC. See *SAS DATA Step Statements: Reference* for a description of SAS programming statements.

There are also a number of unique functions in PROC MCMC that calculate the log density of various distributions in the procedure. You can find them at the section "Using Density Functions in the Programming Statements" on page 6291.

For the list of matrix-based functions that is supported in PROC MCMC, see the section "Matrix Functions in PROC MCMC" on page 6298.

The following are valid statements:

ABORT;
ARRAY *arrayname* <**[** *dimensions* **]**> <**$**> < *variables-and-constants*>**;**
CALL *name* <**(***expression* <**,** *expression . . .* >**)**>**;**
DELETE;
DO < *variable* **=** *expression* <**TO** *expression*> <**BY** *expression*>>
    <**,** *expression* <**TO** *expression*> <**BY** *expression*>> . . .
    <**WHILE** *expression*> <**UNTIL** *expression*>**;**
END;
GOTO *statement-label***;**
IF *expression***;**
IF *expression* **THEN** *program-statement***;**
            ELSE *program-statement***;**
variable **=** *expression***;**
variable **+** *expression***;**
LINK *statement-label***;**
PUT < *variable*> <**=**> . . . **;**
RETURN;
SELECT <**(***expression***)**>**;**
STOP;
SUBSTR(*variable***,** *index***,** *length***)=** *expression***;**
WHEN (*expression***)***program-statement***;**
    OTHERWISE *program-statement***;**

For the most part, the SAS programming statements work the same as they do in the DATA step, as documented in *SAS Programmers Guide: Essentials*. However, there are several differences:

- The ABORT statement does not allow any arguments.
- The DO statement does not allow a character index variable. Thus

  ```
  do i = 1,2,3;
  ```

  is supported; however, the following statement is not supported:

  ```
  do i = 'A','B','C';
  ```

- The PUT statement, used mostly for program debugging in PROC MCMC (see the section "Handling Error Messages" on page 6352), supports only some of the features of the DATA step PUT statement, and it has some features that are not available with the DATA step PUT statement:
  - The PROC MCMC PUT statement does not support line pointers, factored lists, iteration factors, overprinting, _INFILE_, _OBS_, the colon (:) format modifier, or "$".
  - The PROC MCMC PUT statement does support expressions, but the expression must be enclosed in parentheses. For example, the following statement displays the square root of x:

    ```
    put (sqrt(x));
    ```

- The WHEN and OTHERWISE statements enable you to specify more than one target statement. That is, DO/END groups are not necessary for multiple statement WHENs. For example, the following syntax is valid:

```
select;
   when (exp1) stmt1;
               stmt2;
   when (exp2) stmt3;
               stmt4;
end;
```

You should avoid defining variables that begin with an underscore (_). They might conflict with internal variables created by PROC MCMC. The MODEL statement must come after any SAS programming statements that define or modify terms used in the construction of the log likelihood.

RANDOM Statement

RANDOM *random-effect* ∼ *distribution* **SUBJECT=***variable* < *options* > **;**

The RANDOM statement defines a single random effect and its prior distribution or an array of random effects and their prior distribution. The *random-effect* must be represented by either a symbol or an array. The RANDOM statement must consist of the *random-effect*, a tilde (∼), the distribution for the random effect, and then a SUBJECT= variable.

SUBJECT=*variable* | **_OBS_**
: identifies the subjects in the random-effects model. The *variable* must be part of the input data set, and it can be either a numeric variable or character literal. The *variable* does not need to be sorted, and the input data set does not need to be clustered according to it. SUBJECT=_OBS_ enables you fit an observation-level random-effects model (each observation has its own random effect) without specifying a subject variable in the input data set.

The random-effects parameters associated with each subject in the same RANDOM statement are assumed to be conditionally independent of each other, given other parameters and data set variables in the model. The other parameters include model parameters (declared in the PARMS statements), random-effects parameters (from other RANDOM statements), and missing data variables.

Table 80.4 shows the *distributions* that you can specify in the RANDOM statement.

Table 80.4 Valid Distributions in the RANDOM Statement

Distribution Name	Definition
beta(< **a=** >α, < **b=** >β)	Beta distribution with shape parameters α and β
binary(< **prob**|**p=** > p)	Binary (Bernoulli) distribution with probability of success p. You can use the alias **bern** for this distribution.
gamma(< **shape**|**sp=** > a, **scale**|**s=** λ) **gamma**(< **shape**|**sp=** > a, **iscale**|**is=** λ)	Gamma distribution with shape a and scale or inverse-scale λ

Table 80.4 *continued*

Distribution Name	Definition
dgeneral(*ll*)	General log-prior function that you construct using SAS programming statements for univariate or multivariate discrete random effects. See the section "Specifying a New Distribution" on page 6290 for more details.
general(*ll*)	General log-prior function that you construct using SAS programming statements for univariate or multivariate continuous random effects. See the section "Specifying a New Distribution" on page 6290 for more details.
igamma(< **shape**|**sp=** > *a*, **scale**|**s=** λ) **igamma**(< **shape**|**sp=** > *a*, **iscale**|**is=** λ)	Inverse gamma distribution with shape a and scale or inverse-scale λ
laplace(< **location**|**loc**|**l=** > θ, **scale**|**s=** λ) **laplace**(< **location**|**loc**|**l=** > θ, **iscale**|**is=** λ)	Laplace distribution with location θ and scale or inverse-scale λ. This is also known as the *double exponential* distribution. You can use the alias **dexpon** for this distribution.
normal(< **mean**|**m=** > μ, **sd=** λ) **normal**(< **mean**|**m=** > μ, **var**|**v=** λ) **normal**(< **mean**|**m=** > μ, **prec=** λ)	Normal (Gaussian) distribution with mean μ and a value of λ for the standard deviation, variance, or precision. You can use the aliases **gaussian**, **norm**, or **n** for this distribution.
poisson(< **mean**|**m=** > λ)	Poisson distribution with mean λ
table(< **p=** > p)	Table (categorical) distribution with probability vector p. You can also use the alias **cat** for this distribution
uniform(< **left**|**l=** > a, < **right**|**r=** > b)	Uniform distribution with range a and b. You can use the alias **unif** for this distribution.
MVN(< **mu=** >$\boldsymbol{\mu}$, < **cov=** >$\boldsymbol{\Sigma}$)	Multivariate normal distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
MVNAR(< **mu=** >μ, **sd=** λ, < **rho=** >ρ) **MVNAR**(< **mu=** >μ, **var=** λ, < **rho=** >ρ) **MVNAR**(< **mu=** >μ, **prec=** λ, < **rho=** >ρ)	Multivariate normal distribution with mean vector μ and a covariance matrix Σ. The covariance matrix Σ is a multiple of the scale and a matrix with a first-order autoregressive structure

Table 80.4 *continued*

Distribution Name	Definition
normalcar(**neighbors=**, **num=**, <**sd=**>λ) **normalcar**(**neighbors=**, **num=**, <**var=**>λ) **normalcar**(**neighbors=**, **num=**, <**prec=**>λ)	Intrinsic Gaussian conditional autoregressive (CAR) distribution. The NUM= option specifies the name of the data set variable that contains the number of neighbors for each subject; the NEIGHBORS= option specifies the prefix of data set variables that contain the neighboring indices of each subject; λ is the standard deviation, variance, or precision of the distribution.

The following RANDOM statement specifies a scale effect, where s2u can be a constant or a model parameter and index is a data set variable that indicates group membership of the random effect u:

```
random u ~ normal(0,var=s2u) subject=index;
```

The following statements specify multidimensional effects, where mu and cov can be either parameters in the model or constant arrays:

```
array w[2];
array mu[2];
array cov[2,2];
random w ~ mvn(mu, cov) subject=index;
```

You can specify multiple RANDOM statements. Hyperparameters in the prior distribution of a random effect can be other random effects in the model. For example, the following statements are allowed because the random effect g appears in the distribution for the random effect u:

```
random g ~ normal(0,var=s2g) subject=month;
random u ~ normal(g,var=s2u) subject=day;
```

These two RANDOM statements specify a nested hierarchical model in which the random effect g is the hyperparameter of the random effect u. You can build the hierarchical structure as deep as you want. You can also use multiple RANDOM statements to build non-nested random-effects models.

The number of random-effects parameters in each RANDOM statement is determined by the number of unique values in the SUBJECT= variable, which can be either unsorted numeric or unsorted character literal. Unlike the model parameters that are explicitly declared in the PARMS statement (with therefore a fixed total number), the number of random-effects parameters in a program depends on the values of the SUBJECT= data set variable. That number can change from one BY group to another.

The order of the RANDOM statements, or their relative placement with respect to other statements in the program (such as the PRIOR statement or the MODEL statement), is not important. The programming order becomes relevant if any hyperparameters are defined variables in the program. For example, in the following statements, the hyperparameter s is defined as a function of some variable or parameter in the model:

```
s = sqrt(s2g);
random g ~ normal(0,sd=s) subject=month;
```

That definition of s must appear before the RANDOM statement that requires it. If you switched the order of the statements as follows, PROC MCMC would not be able to calculate the prior density for some subjects correctly and would produce erroneous results.

```
random g ~ normal(0,sd=s) subject=month;
s = sqrt(s2g);
```

The names of the random-effects parameters are created internally. See the NAMESUFFIX= option for the naming convention of the random-effects parameters. The random-effects parameters are updated conditionally in the simulation. All posterior draws are saved to the OUTPOST= output data set by default, and you can use the MONITOR= option to monitor any of the parameters. For more information about available sampling algorithms, see the ALGORITHM= option. For more information about how to set a random-effects parameter to a constant (also known as corner-point constraint), see the CONSTRAINT option.

You can specify the following *options* in the RANDOM statement:

ALGORITHM=*option*

ALG=*option*

specifies the algorithm to use to sample the posterior distribution. The following *options* are available:

RWM

uses the random-walk Metropolis algorithm with normal proposal.

SLICE

uses the slice sampling algorithm.

GEO

uses the discrete random-walk Metropolis with symmetric geometric proposal.

When possible, PROC MCMC samples directly from the full conditional distribution. Otherwise, the default sampling algorithm is the RWM.

CENTER | NOCENTER

specifies whether to re-center the random-effects parameters after each draw. This option applies only when you use the NORMALCAR prior. The default is CENTER.

CONSTRAINT(VALUE=*value***) = FIRST | LAST | NONE | '***formatted-value***'**

ZERO=FIRST | LAST | NONE | '*formatted-value***'**

sets one of the random-effects parameters to a fixed value. The default is ZERO=NONE, which does not fix any of the parameters to be a constant. This option enables you to eliminate one of the parameters.

For example, this option could be useful if you want to fit a regression model with categorical covariates and, instead of creating a design matrix, you treat the parameters as "random effects" and fit an equivalent random-effects model.

Suppose you have a regression that includes a categorical variable X with J levels. You can construct a full-rank design matrix with J–1 dummy variables ($X_2 \cdots X_J$ with X_1 being the base group) and fit a

regression such as the following:

$$\mu_i = \beta_0 + \beta_2 \cdot X_2 \cdots \beta_J \cdot X_J$$

The following statements in a PROC MCMC step fit such a hypothetical regression model:

```
parms beta0 betax2 ... betaxJ;
prior beta: ~ n(0, sd=100);
mu = beta0 + betax2 * x2 + ... betaxJ * xJ;
...
```

Equivalently, you can also treat this model as a random-effects model such as the following, where β_j are random effects for each category in X:

$$\mu_i = \beta_0 + \beta_j \text{ for } j = 1, \ldots, J$$

However, this random-effects model is over-parameterized. The ZERO= option rids the model with one random-effects parameter of choice and fixes it to be zero. The following example statements fit such a hypothetical random-effects model:

```
parms beta0;
prior beta0 ~ n(0, sd=100);
random beta ~ n(0, sd=100) subject=x zero=first;
mu = beta0 + beta;
...
```

The specification ZERO=FIRST sets the first random-effects parameter to 0, implying $\beta_1 = 0$. This random-effects parameter corresponds to the first category in the SUBJECT= variable. The category is what the first observation of the SUBJECT= variable takes.

The specification ZERO=LAST sets the last random-effects parameter to be 0, implying $\beta_J = 0$. This random-effects parameter corresponds to the last category in the SUBJECT= variable. The category is not necessarily the same category that the last observation of the SUBJECT= variable takes because the SUBJECT= variable does not need to be sorted.

The specification ZERO='*formatted-value*' sets the random-effects parameter for the category (in the SUBJECT= variable) with a formatted value that matches '*formatted-value*' to 0. For example, ZERO='3' sets $\beta_3 = 0$.

The CONSTRAINT(VALUE=*value*) option works similarly to the ZERO= option. You can assign an arbitrary *value* to any one of the random-effects parameter. For example, the specification CONSTRAINT(VALUE=0)=FIRST is equivalent to ZERO=FIRST.

ICOND=*variable-list* | *numeric-list*

ISTATES=*variable-list* | *numeric-list*

specifies the initial conditions (or initial states) of the lag or lead variable of the random effect when the subject indices are out of the range of the subjects. (For more information about rules of constructing lag and lead variables in PROC MCMC, see the section "Access Lag and Lead Variables" on page 6306.) This works similarly to the ICOND= option in the MODEL statement, except that the index is done

according to a subject variable, not observations. The initial conditions can be model parameters, functions of model parameters, or constants. By default, *numeric-list* is set to 0.

The ICOND= option in a RANDOM statement sets the initial conditions for all lag or lead variables (of the associated random effect) that appear in the program, not just those that appear in the RANDOM statement. Suppose you have a maximum L number of lag variables and a maximum M number of lead variables of the random effect mu in the program, and there are n clusters. The program has the following vector of variables that need to be resolved during simulation:

$$\mu_{L+1}, \ldots, \mu_0, \mu_1, \ldots, \mu_n, \mu_{n+1}, \ldots, \mu_{n+M}$$

Of these variables, n ($\mu_1, \ldots, \mu_n$) are random-effects parameters and the remaining $L+M$ are initial conditions that are specified in the ICOND= option. The *variable-list* (or the *number-list*) should be a vector of length $L+M$, which can be greater than the number of lag/lead random-effect variables that appear in a program. If you provide a partial list that contains fewer than $L+M$ states, PROC MCMC fills the remaining vector with the value of 0.

INITIAL=*SAS-data-set* | *constant* | *numeric-list*

specifies the initial values of the random-effects parameters. By default, PROC MCMC uses the same option as specified in the INIT= option to generate initial values for the random-effects parameter: either it uses the mode of the prior density or it randomly draws a sample from that distribution. You can start the Markov chain at different places by providing a *SAS-data-set*, a constant, or a *numeric-list* for multivariate random-effects parameters.

If you use a *SAS-data-set*, the data set must consist of variable names that agree with the random-effects parameters in the model (see the NAMESUFFIX= option for the naming convention of the random-effects parameters). The easiest way to find the names of the internally created parameter names is to run a default analysis with a very small number of simulations and check the variable names in the OUTPOST= data set. You can provide a subset of the initial values in the *SAS-data-set* and PROC MCMC will use the default mechanism to fill in the rest of the random-effects parameters.

For example, the following statement creates a data set with initial values for the random-effects parameters u_1, u_2, and u_3:

```
data RandomInit;
   input u_1 u_2 u_3;
   datalines;
2.3 3 -3
;
```

The following RANDOM statement takes the values in the RandomInit data set to be the initial values of the corresponding random-effects parameters in the model:

```
random u ~ normal(0,var=s2u) subject=index init=randominit;
```

Specifying a *constant* assigns that constant as the initial value to all random-effects parameters in the statement. For example, the following statement assigns the value 5 to be used as an initial value for all u_i in the model:

```
random u ~ normal(0,var=s2u) subject=index init=5;
```

If you have multiple effects, you can provide a list of numbers, where the length of the list the same as the dimension of your random-effects array. Each number is then given to all corresponding random-effects parameters in order. For example, the following statement assigns the value 2 to be used as an initial value for all $w1_i$ and the value 3 to be used for all $w2_i$ in the model:

```
array w[2] w1 w2;
random w ~ mvn(mu, cov) subject=index init=(2 3);
```

If you use the GENERAL or DGENERAL functions in the RANDOM statement, you must provide initial values for these parameters.

MONITOR= (*symbol-list* | *number-list* | **RANDOM(***number***))**

outputs analysis for selected random-effects parameters. You can choose to monitor the random-effects parameters by listing the effect names or effect indices, or you can have them randomly selected by PROC MCMC.

To monitor all random-effects parameters, you specify the effect name in the MONITOR= option:

```
random u ~ normal(0,var=s2u) subject=index monitor=(u);
```

You have three options for monitoring a subset of the random-effects parameters. You can provide a list of the parameter names, you can provide a number list of the parameter indices, or you can have PROC MCMC randomly choose a subset of parameters for you.

For example, if you want to monitor analysis for parameters u_1 through u_10, u_23, and u_57, you can provide the names as follows:

```
random u ~ normal(0,var=s2u) subject=index monitor=(u_1-u_10 u_23 u_57);
```

The naming convention in the *symbol-list* must agree with the NAMESUFFIX= option, which controls how the parameter names of the *random-effect* are created. By default, NAMESUFFIX=SUBJECT, and the *symbol-list* must use suffixes that correspond to the formatted values in the SUBJECT= data set variable. With the NAMESUFFIX=POSITION option, the *symbol-list* must use suffixes that agree with the input order of the SUBJECT= variable. If the SUBJECT= variable has a character value, you cannot use the hyphen (-) in the *symbol-list* to indicate a range of variables.

To monitor the same list of random-effects parameters, you can provide their indices:

```
random u ~ normal(0,var=s2u) subject=index monitor=(1 to 10 by 1 23 57);
```

PROC MCMC can also randomly choose a subset of the parameters to monitor:

```
random u ~ normal(0,var=s2u) subject=index monitor=(random(12));
```

The sequence of the random indices is controlled by the SEED= option in the PROC MCMC statement.

By default, PROC MCMC does not monitor any random-effects parameters. When you specify this option, it takes the specification of the STATISTICS= and PLOTS= options in the PROC MCMC statement. By default, PROC MCMC outputs all the posterior samples of all random-effects parameters to the OUTPOST= output data set. You can use the NOOUTPOST option to suppress the saving of the random-effects parameters.

NAMESUFFIX=*option*

specifies how the names of the random-effects parameters are internally created from the SUBJECT= *variable* that is specified in the RANDOM statement. PROC MCMC creates the names by concatenating the *random-effect* symbol with an underscore and a series of numbers or characters. The following *options* control the type of methods that are used in such construction:

SUBJECT

constructs the parameter names by appending the formatted values of the SUBJECT= *variable* in the input data set.

POSITION

constructs the parameter names by appending the numbers 1, 2, 3, and so on, where the number indicates the order in which the SUBJECT= *variable* appears in the data set.

For example, suppose you have an input data set with four observations and the SUBJECT= variable zipcode has four values (with three of them unique): 27513, 01440, 27513, and 15217. The following SAS statement creates three random-effects parameters named u_27513, u_01440, and u_15217:

```
random u ~ normal(0,var=10) subject=zipcode namesuffix=subject;
```

On the other hand, using NAMESUFFIX=POSITION creates three parameters named u_1, u_2, and u_3:

```
random u ~ normal(0,var=10) subject=zipcode namesuffix=position;
```

By default, NAMESUFFIX=SUBJECT.

NOOUTPOST

suppresses the output of the posterior samples of random-effects parameters to the OUTPOST= data set. In models with a large number of random-effects parameters (for example, tens of thousands), PROC MCMC can run faster if it does not save the posterior samples of the random-effects parameters.

When you specify both the NOOUTPOST option and the MONITOR= option, PROC MCMC outputs the list of variables that are monitored.

The maximum number of variables that can be saved to an OUTPOST= data set is 32,767. If you run a large-scale random-effects model with the number of parameters exceeding the limit, the NOOUTPOST option is evoked automatically and PROC MCMC does not save the random-effects parameter draws to the posterior output data set. You can use the MONITOR= option to select a subset of the parameters to store in the OUTPOST= data set.

UDS Statement

UDS *subroutine-name* **(***subroutine-argument-list***) ;**

UDS stands for user defined sampler. The UDS statement enables you to use a separate algorithm, other than the default random walk Metropolis, to update parameters in the model. The purpose of the UDS statement is to give you a greater amount of flexibility and better control over the updating schemes of the Markov chain. Multiple UDS statements are allowed.

For the UDS statement to work properly, you have to do the following:

- write a subroutine by using PROC FCMP (see the FCMP Procedure in the *Base SAS Procedures Guide*) and save it to a SAS catalog (see the example in this section). The subroutine must update some parameters in the model. These are the UDS parameters. The subroutine is called the UDS subroutine.
- declare any UDS parameters in the PARMS statement with a sampling option, as in </ UDS > (see the section "PARMS Statement" on page 6249).
- specify the prior distributions for all UDS parameters, using the PRIOR statements.

NOTE: All UDS parameters must appear in three places: the UDS statement, the PARMS statement, and the PRIOR statement. Otherwise, PROC MCMC exits.

To obtain a valid Markov chain, a UDS subroutine must update a parameter from its full posterior conditional distribution and not the posterior marginal distribution. The posterior conditional is something that you need to provide. This conditional is implicitly based on a prior distribution. PROC MCMC has no means to verify that the implied prior in the UDS subroutine is the same as the prior that you specified in the PRIOR statement. You need to make sure that the two distributions agree; otherwise, you will get misleading results.

The priors in the PRIOR statements do not directly affect the sampling of the UDS parameters. They could affect the sampling of the other parameters in the model, which, in turn, changes the behavior of the Markov chain. You can see this by noting cases where the hyperparameters of the UDS parameters are model parameters; the priors should be part of the posterior conditional distributions of these hyperparameters, and they cannot be omitted.

Some additional information is listed to help you better understand the UDS statement:

- Most features of the SAS programming language can be used in subroutines processed by PROC FCMP (see the FCMP Procedure in the *Base SAS Procedures Guide*).
- The UDS statement does not support FCMP functions—a FCMP function returns a value, while a subroutine does not. A subroutine updates some of its subroutine arguments. These arguments are called OUTARGS arguments.
- The UDS parameters cannot be in the same block as other parameters. The optional argument </ UDS > in the PARMS statement prevents parameters that use the default Metropolis from being mixed with those that are updated by the UDS subroutines.
- You can put all the UDS parameters in the same PARMS statement or have a separate UDS statement for each of them.

- The same subroutine can be used in multiple UDS statements. This feature comes in handy if you have a generic sampler that can be applied to different parameters.
- PROC MCMC updates the UDS parameters by calling the UDS subroutines directly. At every iteration, PROC MCMC first samples parameters that use the Metropolis algorithm, then the UDS parameters. Sampling of the UDS parameters proceeds in the order in which the UDS statements are listed.
- A UDS subroutine accepts any symbols in the program as well as any input data set variables as its arguments.
- Only the OUTARGS arguments in a UDS subroutine are updated in PROC MCMC. You can modify other arguments in the subroutine, but the changes are not global in PROC MCMC.
- If a UDS subroutine has an argument that is a SAS data set variable, PROC MCMC steps through the data set while updating the UDS parameters. The subroutine is called once per observation in the data set for every iteration.
- If a UDS subroutine does not have any arguments that are data set variables, PROC MCMC does not access the data set while executing the subroutine. The subroutine is called once per iteration.
- To reduce the overhead in calling the UDS subroutine and accessing the data set repeatedly, you might consider reading all the input data set variables into arrays and using the arrays as the subroutine arguments. See the section "BEGINCNST/ENDCNST Statement" on page 6234 about how to use the BEGINCNST and ENDCNST statements to store data set variables.

For an example that uses the UDS statement, see "Example 80.19: Implement a New Sampling Algorithm" on page 6464.

Details: MCMC Procedure

How PROC MCMC Works

PROC MCMC is a simulation-based procedure that applies a variety of sampling algorithms to the program at hand. The default sampling methods include conjugate sampling (from full conditional), direct sampling from the marginal distribution, inverse cumulative distribution function, random walk Metropolis with normal proposal, and discretized random walk Metropolis with normal proposal. You can request alternate sampling algorithms, such as random walk Metropolis with t distribution proposal, discretized random walk Metropolis with symmetric geometric proposal, and the slice sampling algorithm.

PROC MCMC applies the more efficient sampling algorithms first, whenever possible. When a parameter does not appear in the conditional distributions of other random variables in the program, PROC MCMC generates samples directly from its prior distribution (which is also its marginal distribution). This usually occurs in data-independent Monte Carlo simulation programs (see "Example 80.1: Simulating Samples From a Known Density" on page 6362 for an example) or missing data problems, where the missing response variables are generated directly from the conditional sampling distribution (or the conditional likelihood). When conjugacy is detected, PROC MCMC uses random number generators to draw values from the full conditional distribution. (For information about detecting conjugacy, see the section "Conjugate Sampling"

on page 6272.) In other situations, PROC MCMC resorts to the random walk Metropolis with normal proposal to generate posterior samples for continuous parameters and a discretized version for discrete parameters. See the section "Metropolis and Metropolis-Hastings Algorithms" on page 156 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for details about the Metropolis algorithm. For the actual implementation details of the Metropolis algorithm in PROC MCMC, such as tuning of the covariance matrices, see the section "Tuning the Proposal Distribution" on page 6269.

A key component of the Metropolis algorithm is the calculation of the objective function. In most cases, the objective function that PROC MCMC uses in a Metropolis step is the logarithm of the joint posterior distribution, which is calculated with the inclusion of all data and parameters. The rest of this section describes how PROC MCMC calculates the objective function for parameters that use the Metropolis algorithm.

Model Parameters

To calculate the log of the posterior density, PROC MCMC assumes that all observations in the data set are independent,

$$\log(p(\theta|\mathbf{y})) = \log(\pi(\theta)) + \sum_{i=1}^{n} \log(f(y_i|\theta))$$

where θ is a parameter or a vector of parameters that are defined in the PARMS statements (referred to as the *model parameters*). The term $\log(\pi(\theta))$ is the sum of the log of the prior densities specified in the PRIOR and HYPERPRIOR statements. The term $\log(f(y_i|\theta))$ is the log likelihood specified in the MODEL statement. The MODEL statement specifies the log likelihood for a single observation in the data set.

If you want to model dependent data—that is, $\log(f(\mathbf{y}|\theta)) \neq \sum_i \log(f(y_i|\theta))$—you can use the JOINTMODEL option in the PROC MCMC statement. See the section "Modeling Joint Likelihood" on page 6304 for more details.

The statements in PROC MCMC are similar to DATA step statements; PROC MCMC evaluates every statement in order for each observation. At the beginning of the data set, the log likelihood is set to be 0. As PROC MCMC steps through the data set, it cumulatively adds the log likelihood for each observation. Statements between the BEGINNODATA and ENDNODATA statements are evaluated only at the first and the last observations. At the last observation, the log of the prior and hyperprior distributions is added to the sum of the log likelihood to obtain the log of the posterior distribution.

Calculation of the $\log(p(\theta|\mathbf{y}))$ objective function involves a complete pass through the data set, making it potentially computationally expensive. If $\theta = \{\theta_1, \theta_2\}$ is multidimensional, you can choose to update a portion of the parameters at each iteration step by declaring them in separate PARMS statements (see the section "Blocking of Parameters" on page 6266 for more information). PROC MCMC updates each block of parameters while holding others constant. The objective functions that are used in each update are the same as the log of the joint posterior density:

$$\log(p(\theta_1|\mathbf{y}, \theta_2)) = \log(p(\theta_2|\mathbf{y}, \theta_1)) = \log(p(\theta|\mathbf{y}))$$

In other words, PROC MCMC does not derive the conditional distribution explicitly for each block of parameters, and it uses the full joint distribution in the Metropolis step for every block update.

Random-Effects Models

For programs that require RANDOM statements, PROC MCMC includes the sum of the density evaluation of the random-effects parameters in the calculation of the objective function for θ,

$$\log(p(\theta|\boldsymbol{\gamma},\mathbf{y})) = \log(\pi(\theta)) + \sum_{j=1}^{J}\log(\pi(\gamma_j|\theta)) + \sum_{i=1}^{n}\log(f(y_i|\theta,\boldsymbol{\gamma}))$$

where $\boldsymbol{\gamma} = \{\gamma_1,\ldots,\gamma_J\}$ are random-effects parameters and $\pi(\gamma_j|\theta)$ is the prior distribution of the random-effects parameters. The likelihood function can be conditional on $\boldsymbol{\gamma}$, but the prior distributions of θ, which must be independent of $\boldsymbol{\gamma}$, cannot.

The objective function used in the Metropolis step for the random-effects parameter γ_j contains only the portion of the data that belong to the jth cluster:

$$\log(p(\gamma_j|\theta,\mathbf{y})) = \log(\pi(\gamma_j|\theta)) + \sum_{i\in\{j\text{th cluster}\}}\log(f(y_i|\theta,\gamma_j))$$

The calculation does not include $\log(\theta)$, the prior density piece, because that is a known constant. Evaluation of this objective function involves only a portion of the data set, making it more computationally efficient. In fact, updating every random-effects parameters in a single RANDOM statement involves only one pass through the data set.

You can have multiple RANDOM statements in a program, which adds more pieces to the posterior calculation, such as

$$\log(p(\theta|\boldsymbol{\gamma},\boldsymbol{\alpha},\mathbf{y})) = \log(\pi(\theta)) + \sum_{j=1}^{J}\log(\pi(\gamma_j|\theta)) + \sum_{k=1}^{K}\log(\pi(\alpha_k|\theta)) + \sum_{i=1}^{n}\log(f(y_i|\theta,\boldsymbol{\gamma},\boldsymbol{\alpha}))$$

where $\boldsymbol{\alpha} = \{\alpha_1,\ldots,\alpha_K\}$ is another random effect. The random effects $\boldsymbol{\gamma}$ and $\boldsymbol{\alpha}$ can form their own hierarchy (as in a nested model), or they can enter the program in a non-nested fashion. The objective functions for γ_j and α_k are calculated using only observations that belong to their respective clusters.

Models with Missing Values

Missing values in the response variables of the MODEL statement are treated as random variables, and they add another layer in the conditional updates in the simulation. Suppose that

$$\mathbf{y} = \{\mathbf{y}_{\mathbf{obs}},\mathbf{y}_{\mathbf{mis}}\}$$

The response variable $\mathbf{y}$ consists of n_1 observed values $\mathbf{y}_{\mathbf{obs}}$ and n_2 missing values $\mathbf{y}_{\mathbf{mis}}$. The log of the posterior distribution is thus formed by

$$\log(p(\theta|\boldsymbol{\gamma},\mathbf{y}_{\mathbf{mis}},\mathbf{y}_{\mathbf{obs}})) = \log(\pi(\theta))+\sum_{j=1}^{J}\log(\pi(\gamma_j|\theta))+\sum_{i=1}^{n_2}\log(f(y_{\mathbf{mis},i}|\theta,\boldsymbol{\gamma}))\sum_{i=1}^{n_1}\log(f(y_{\mathbf{obs},i}|\theta,\boldsymbol{\gamma}))$$

where the expression is evaluated at the drawn $\boldsymbol{\gamma}$ and $\mathbf{y}_{\mathbf{obs}}$ values.

The conditional distribution of the random-effects parameter γ_j is

$$\log(p(\gamma_j|\theta,\mathbf{y})) = \log(\pi(\gamma_j|\theta)) + \sum_{i\in\{j\text{th cluster}\}}\log(f(y_i|\theta,\gamma_j))$$

where the y_i are either the observed or the imputed values of the response variable.

The missing values are usually sampled directly from the sampling distribution and do not require the Metropolis sampler. When a response variable takes on a GENERAL function, the objective function is simply the likelihood function: $\log\left(f(y_{\mathbf{mis},i}|\theta,\gamma_j)\right)$.

Blocking of Parameters

In a multivariate parameter model, if all k parameters are proposed with one joint distribution $q(\cdot|\cdot)$, acceptance or rejection would occur for all of them. This can be rather inefficient, especially when parameters have vastly different scales. A way to avoid this difficulty is to allocate the k parameters into d blocks and update them separately. The PARMS statement puts model parameters in separate blocks, and each block of parameters is updated sequentially in the procedure.

Suppose you want to sample from a multivariate distribution with probability density function $p(\theta|\mathbf{y})$ where $\theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$. Now suppose that these k parameters are separated into d blocks—for example, $p(\theta|\mathbf{x}) = f_d(z)$ where $z = \{z_1, z_2, \ldots, z_d\}$, where each z_j contains a nonempty subset of the $\{\theta_i\}$, and where each θ_i is contained in one and only one z_j. In the MCMC context, the z's are blocks of parameters. In the blocked algorithm, a proposal consists of several parts. Instead of proposing a simultaneous move for all the θ's, a proposal is made for the θ_i's in z_1 only, then for the θ_i's in z_2, and so on for d subproposals. Any accepted proposal can involve any number of the blocks moving. The parameters do not necessarily all move at once as in the all-at-once Metropolis algorithm.

Formally, the blocked Metropolis algorithm is as follows. Let w_j be the collection of θ_i that are in block z_j, and let $q_j(\cdot|w_j)$ be a symmetric multivariate distribution that is centered at the current values of w_j.

1. Let $t = 0$. Choose points for all w_j^t. A point can be an arbitrary point as long as $p(w_j^t|\mathbf{y}) > 0$.
2. For $j = 1, \ldots, d$:
   a) Generate a new sample, $w_{j,\text{new}}$, using the proposal distribution $q_j(\cdot|w_j^t)$.
   b) Calculate the following quantity:

$$r = \min\left\{\frac{p(w_{j,\text{new}}|w_1^t, \ldots, w_{j-1}^t, w_{j+1}^{t-1}, \ldots, w_d^{t-1}, \mathbf{y})}{p(w_j^t|w_1^t, \ldots, w_{j-1}^t, w_{j+1}^{t-1}, \ldots, w_d^{t-1}, \mathbf{y})}, 1\right\}.$$

   c) Sample u from the uniform distribution $U(0,1)$.
   d) Set $w_j^{t+1} = w_{j,\text{new}}$ if $r < u$; $w_j^{t+1} = w_j^t$ otherwise.
3. Set $t = t + 1$. If $t < T$, the number of desired samples, go back to Step 2; otherwise, stop.

With PROC MCMC, you can sample all parameters simultaneously by putting them all in a single PARMS statement, you can sample parameters individually by putting each parameter in its own PARMS statement, or you can sample certain subsets of parameters together by grouping each subset in its own PARMS statements. For example, if the model you are interested in has five parameters, alpha, beta, gamma, phi, sigma, the all-at-once strategy is as follows:

```
parms alpha beta gamma phi sigma;
```

The one-at-a-time strategy is as follows:

```
parms alpha;
parms beta;
parms gamma;
parms phi;
parms sigma;
```

A two-block strategy could be as follows:

```
parms alpha beta gamma;
parms phi sigma;
```

The exceptions to the previously described blocking strategies are parameters that are sampled directly (either from their full conditional or marginal distributions) and parameters that are array-based (with multivariate prior distributions). In these cases, the parameters are taken out of an existing block and are updated individually. You can use the sampling options in the PARMS statement to override the default behavior.

One of the greatest challenges in MCMC sampling is achieving good mixing of the chains—the chains should quickly traverse the support of the stationary distribution. A number of factors determine the behavior of a Metropolis sampler; blocking is one of them, so you want to be extremely careful when you choose a good design. Generally speaking, forming blocks of parameters has its advantages, but it is not true that the larger the block the faster the convergence.

When simultaneously sampling a large number of parameters, the algorithm might find it difficult to achieve good mixing. As the number of parameters gets large, it is much more likely to have (proposal) samples that fall well into the tails of the target distribution, producing too small a test ratio. As a result, few proposed values are accepted and convergence is slow. On the other hand, when the algorithm samples each parameter individually, the computational cost increases linearly. Each block of Metropolis parameters requires one additional pass through the data set, so a five-block updating strategy could take five times longer than a single-block updating strategy. In addition, there is a chance that the chain might mix far too slowly because the conditional distributions (of θ_i given all other θ's) might be very "narrow," as a result of posterior correlation among the parameters. When that happens, it takes a long time for the chain to fully explore that dimension alone. There are no theoretical results that can help determine an optimal "blocking" for an arbitrary parametric model. A rule followed in practice is to form small groups of correlated parameters that belong to the same context in the formulation of the model. The best mixing is usually obtained with a blocking strategy somewhere between the all-at-once and one-at-a-time strategies.

Sampling Methods

When suitable, PROC MCMC chooses the optimal sampling method for each parameter. That involves direct sampling either from the conditional posterior via conjugacy (see the section "Conjugate Sampling" on page 6272) or via the marginal posterior (see the section "Direct Sampling" on page 6272). Alternatively, PROC MCMC samples according to Table 80.5. Each block of parameters is classified by the nature of the prior distributions. "Continuous" means all priors of the parameters in the same block have a continuous distribution. "Discrete" means all priors are discrete. "Mixed" means that some parameters are continuous and others are discrete. Parameters that have binary priors are treated differently, as indicated in the table.

Table 80.5 Sampling Methods in PROC MCMC

Blocks	Default Method	Alternative Method
Continuous	Multivariate normal (MVN)	Multivariate *t* (MVT); slice sampler
Discrete (other than binary)	Binned MVN	Binned MVT or symmetric geometric
Mixed	MVN	MVT
Binary (single dimensional)	Inverse CDF	
Binary (multidimensional)	Independence sampler	

For a block of continuous parameters, PROC MCMC uses a multivariate normal distribution as the default proposal distribution. In the tuning phase, PROC MCMC finds an optimal scale c and a tuning covariance matrix Σ.

For a discrete block of parameters, PROC MCMC uses a discretized multivariate normal distribution as the default proposal distribution. The scale c and covariance matrix $\mathbf{\Sigma}$ are tuned. Alternatively, you can use an independent symmetric geometric proposal distribution. The density has form $\frac{p(1-p)^{|\theta|}}{2(1-p)}$ and has variance $\frac{(2-p)(1-p)}{p^2}$. In the tuning phase, the procedure finds an optimal proposal probability p for every parameter in the block.

You can change the proposal distribution, from the normal to a *t* distribution. You can either use the PROC option PROPDIST=T(*df*) or PARMS statement option </ T(*df*) > to make the change. The *t* distributions have thicker tails, and they can propose to the tail areas more efficiently than the normal distribution. It can help with the mixing of the Markov chain if some of the parameters have a skewed tails. See "Example 80.6: Nonlinear Poisson Regression Models" on page 6392. The independence sampler (see the section "Independence Sampler" on page 160 in Chapter 8, "Introduction to Bayesian Analysis Procedures") is used for a block of binary parameters. The inverse CDF method is used for a block that consists of a single binary parameter.

For parameters with continuous prior distributions, you can use the slice sampler as an alternative sampling algorithm. To do so, specify the SLICE option in the PARMS. When you specify the SLICE option, all parameters are updated individually. PROC MCMC does not support a multivariate version of the slice sampler. For more in information about the slice sampler, see the section"Slice Sampler" on page 159 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

The sampling algorithms for the random-effects parameters are chosen in a similar fashion. The preferred algorithms are the direct method either from the full conditional or the marginal. When these are not attainable, Metropolis with normal proposal becomes the default for continuous random-effects parameters,

and discrete Metropolis with normal proposal becomes the default for discrete random-effects parameters. You can use the ALGORITHM= option in the RANDOM statement to choose the slice sampler or discrete Metropolis with symmetric geometric as the alternatives.

The sampling preference of the missing data variables is the same as the random-effects parameters. The reserve sampling algorithm is the Metropolis. There is no alternative sampling method available for the missing data variables.

Tuning the Proposal Distribution

One key factor in achieving high efficiency of a Metropolis-based Markov chain is finding a good proposal distribution for each block of parameters. This process is referred to as tuning. The tuning phase consists of a number of loops. The minimum number of loops is controlled by the option MINTUNE=, with a default value of 2. The option MAXTUNE= controls the maximum number of tuning loops, with a default value of 24. Each loop lasts for NTU= iterations, where by default NTU= 500. At the end of every loop, PROC MCMC examines the acceptance probability for each block. The acceptance probability is the percentage of NTU= proposals that have been accepted. If the probability falls within the acceptance tolerance range (see the section "Scale Tuning" on page 6269), the current configuration of $c/\mathbf{\Sigma}$ or p is kept. Otherwise, these parameters are modified before the next tuning loop.

Continuous Distribution: Normal or *t* Distribution

A good proposal distribution should resemble the actual posterior distribution of the parameters. Large sample theory states that the posterior distribution of the parameters approaches a multivariate normal distribution (see Gelman et al. 2004, Appendix B, and Schervish 1995, Section 7.4). That is why a normal proposal distribution often works well in practice. The default proposal distribution in PROC MCMC is the normal distribution: $q_j(\theta_{\text{new}}|\theta^t) = \text{MVN}(\theta_{\text{new}}|\theta^t, c^2\mathbf{\Sigma})$. As an alternative, you can choose a multivariate t distribution as the proposal distribution. It is a good distribution to use if you think that the posterior distribution has thick tails and a t distribution can improve the mixing of the Markov chain. See "Example 80.6: Nonlinear Poisson Regression Models" on page 6392.

Scale Tuning

The acceptance rate is closely related to the sampling efficiency of a Metropolis chain. For a random walk Metropolis, high acceptance rate means that most new samples occur right around the current data point. Their frequent acceptance means that the Markov chain is moving rather slowly and not exploring the parameter space fully. On the other hand, a low acceptance rate means that the proposed samples are often rejected; hence the chain is not moving much. An efficient Metropolis sampler has an acceptance rate that is neither too high nor too low. The scale c in the proposal distribution $q(\cdot|\cdot)$ effectively controls this acceptance probability. Roberts, Gelman, and Gilks (1997) showed that if both the target and proposal densities are normal, the optimal acceptance probability for the Markov chain should be around 0.45 in a single dimensional problem, and asymptotically approaches 0.234 in higher dimensions. The corresponding optimal scale is 2.38, which is the initial scale set for each block.

Due to the nature of stochastic simulations, it is impossible to fine-tune a set of variables such that the Metropolis chain has the exact desired acceptance rate. In addition, Roberts and Rosenthal (2001) empirically demonstrated that an acceptance rate between 0.15 and 0.5 is at least 80% efficient, so there is really no need to fine-tune the algorithms to reach acceptance probability that is within small tolerance of the optimal values. PROC MCMC works with a probability range, determined by the PROC options TARGACCEPT

± ACCEPTTOL. The default value of TARGACCEPT is a function of the number of parameters in the model, as outlined in Roberts, Gelman, and Gilks (1997). The default value of ACCEPTTOL= is 0.075. If the observed acceptance rate in a given tuning loop is less than the lower bound of the range, the scale is reduced; if the observed acceptance rate is greater than the upper bound of the range, the scale is increased. During the tuning phase, a scale parameter in the normal distribution is adjusted as a function of the observed acceptance rate and the target acceptance rate. The following updating scheme is used in PROC MCMC [1]:

$$c_{\text{new}} = \frac{c_{\text{cur}} \cdot \Phi^{-1}(p_{\text{opt}}/2)}{\Phi^{-1}(p_{\text{cur}}/2)}$$

where c_{cur} is the current scale, p_{cur} is the current acceptance rate, p_{opt} is the optimal acceptance probability.

Covariance Tuning

To tune a covariance matrix, PROC MCMC takes a weighted average of the old proposal covariance matrix and the recent observed covariance matrix, based on NTU samples in the current loop. The TUNEWT=*w* option determines how much weight is put on the recently observed covariance matrix. The formula used to update the covariance matrix is as follows:

$$\text{COV}_{\text{new}} = w\,\text{COV}_{\text{cur}} + (1 - w)\text{COV}_{\text{old}}$$

There are two ways to initialize the covariance matrix:

- The default is an identity matrix multiplied by the initial scale of 2.38 (controlled by the PROC option SCALE=) and divided by the square root of the number of estimated parameters in the model. It can take a number of tuning phases before the proposal distribution is tuned to its optimal stage, since the Markov chain needs to spend time learning about the posterior covariance structure. If the posterior variances of your parameters vary by more than a few orders of magnitude, if the variances of your parameters are much different from 1, or if the posterior correlations are high, then the proposal tuning algorithm might have difficulty with forming an acceptable proposal distribution.
- Alternatively, you can use a numerical optimization routine, such as the quasi-Newton method, to find a starting covariance matrix. The optimization is performed on the joint posterior distribution, and the covariance matrix is a quadratic approximation at the posterior mode. In some cases this is a better and more efficient way of initializing the covariance matrix. However, there are cases, such as when the number of parameters is large, where the optimization could fail to find a matrix that is positive definite. In that case, the tuning covariance matrix is reset to the identity matrix.

A side product of the optimization routine is that it also finds the *maximum a posteriori* (MAP) estimates with respect to the posterior distribution. The MAP estimates are used as the initial values of the Markov chain.

If any of the parameters are discrete, then the optimization is performed conditional on these discrete parameters at their respective fixed initial values. On the other hand, if all parameters are continuous, you can in some cases skip the tuning phase (by setting MAXTUNE=0) or the burn-in phase (by setting NBI=0).

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

Discrete Distribution: Symmetric Geometric

By default, PROC MCMC uses the normal density as the proposal distribution in all Metropolis random walks. For parameters that have discrete prior distributions, PROC MCMC discretizes proposed samples. You can choose an alternative symmetric geometric proposal distribution by specifying the option DISCRETE=GEO.

The density of the symmetric geometric proposal distribution is as follows:

$$\frac{p_g(1-p_g)^{|\theta|}}{2(1-p_g)}$$

where the symmetry centers at θ. The distribution has a variance of

$$\sigma^2 = \frac{(2-p_g)(1-p_g)}{p_g^2}$$

Tuning for the proposal p_g uses the following formula:

$$\frac{\sigma_{\text{new}}}{\sigma_{\text{cur}}} = \frac{\Phi^{-1}(p_{\text{opt}}/2)}{\Phi^{-1}(p_{\text{cur}}/2)}$$

where σ_{new} is the standard deviation of the new proposal geometric distribution, σ_{cur} is the standard deviation of the current proposal distribution, p_{opt} is the target acceptance probability, and p_{cur} is the current acceptance probability for the discrete parameter block.

The updated p_g is the solution to the following equation that is between 0 and 1 :

$$\sqrt{\frac{(2-p_g)(1-p_g)}{p_g^2}} = \frac{\sigma_{\text{cur}} \cdot \Phi^{-1}(p_{\text{opt}}/2)}{\Phi^{-1}(p_{\text{cur}}/2)}$$

Binary Distribution: Independence Sampler

Blocks consisting of a single parameter with a binary prior do not require any tuning; the inverse-CDF method applies. Blocks that consist of multiple parameters with binary prior are sampled by using an independence sampler with binary proposal distributions. See the section "Independence Sampler" on page 160 in Chapter 8, "Introduction to Bayesian Analysis Procedures." During the tuning phase, the success probability p of the proposal distribution is taken to be the probability of acceptance in the current loop. Ideally, an independence sampler works best if the acceptance rate is 100%, but that is rarely achieved. The algorithm stops when the probability of success exceeds the TARGACCEPTI=*value*, which has a default value of 0.6.

Direct Sampling

The word "direct" is reserved for sampling that is done directly from the prior distribution of a model or a random-effects parameter or from the sampling distribution of a missing data variable. If the parameter is updated via sampling from its full conditional posterior distribution, the sampling method is referred to as conjugate sampling. (See the section "Conjugate Sampling" on page 6272.)

Whenever a parameter does not appear in the hierarchy of another parameter in the model, PROC MCMC samples directly from its distribution. For a model parameter or a random-effects parameter, this distribution is its prior distribution. For a missing data variable, this distribution is the sampling distribution of the response variable. Therefore, direct sampling takes place most frequently in data-independent Monte Carlo simulations or the sampling of missing response variables.

Conjugate Sampling

Conjugate prior is a family of prior distributions in which the prior and the posterior distributions are of the same family of distributions. For example, if you model an independently and identically distributed random variable y_i using a normal likelihood with known variance σ^2,

$$y_i \sim \text{normal}(\mu, \sigma^2)$$

a normal prior on μ

$$\mu \sim \text{normal}(\mu_0, \sigma_0^2)$$

is a conjugate prior because the posterior distribution of μ is also a normal distribution given $\text{y} = \{y_i\}$, σ^2, μ_0, and σ_0^2 :

$$\mu|\text{y} \sim \text{normal}\left(\left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1} \cdot \left(\frac{\mu_0}{\sigma_0^2} + \frac{n \cdot \bar{\text{y}}}{\sigma^2}\right), \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1}\right)$$

Conjugate sampling is efficient because it enables the Markov chain to obtain samples from the target distribution directly. When appropriate, PROC MCMC uses conjugate sampling methods to draw conditional posterior samples. Table 80.6 lists scenarios that lead to conjugate sampling in PROC MCMC.

Table 80.6 Conjugate Sampling in PROC MCMC

Family	Parameter	Prior
Normal with known μ	Variance σ^2	Inverse gamma family
Normal with known μ	Precision τ	Gamma family
Normal with known scale parameter (σ^2, σ, or τ)	Mean μ	Normal
Multivariate normal with known Σ	Mean $\boldsymbol{\mu}$	Multivariate normal
Multivariate normal with known $\boldsymbol{\mu}$	Covariance Σ	Inverse Wishart
Multinomial	$\mathbf{p}$	Dirichlet
Binomial/binary	p	Beta
Poisson	λ	Gamma family

In most cases, Family in Output 80.6 refers to the likelihood function. However, it does not necessarily have to be the case. The Family is a distribution that is conditional on the parameter of interest, and it can appear in any level of the hierarchical model, including on the random-effects level.

PROC MCMC can detect conjugacy only if the model parameter (not a function or a transformation of the model parameter) is used in the prior and Family distributions. For example, the following statements lead to a conjugate sampler being used on the parameter mu:

```
parm mu;
prior mu ~ n(0, sd=1000);
model y ~ n(mu, var=s2);
```

However, if you modify the program slightly in the following way, although the conjugacy still holds in theory, PROC MCMC cannot detect conjugacy on mu because the parameter enters the normal likelihood function through the symbol w:

```
parm mu;
prior mu ~ n(0, sd=1000);
w = mu;
model y ~ n(w, var=s2);
```

In this case, PROC MCMC resorts to the default sampling algorithm, which is a random walk Metropolis based on a normal kernel.

Similarly, the following statements also prevent PROC MCMC from detecting conjugacy on the parameter mu:

```
parm mu;
prior mu ~ n(0, sd=1000);
model y ~ n(mu + 2, var=s2);
```

In a normal family, an often-used and often-confused conjugate prior on the variance is the inverse gamma distribution, and a conjugate prior on the precision is the gamma distribution. See "Gamma and Inverse Gamma Distributions" on page 6332 for typical usages of these prior distributions.

When conjugacy is detected in a model, PROC MCMC performs a numerical optimization on the joint posterior distribution at the start of the MCMC simulation. If the only sampling methods required in the program are conjugate samplers or direct samplers, PROC MCMC omits this optimization step. To turn off this optimization routine, use the PROPCOV=IND option in the PROC MCMC statement.

Initial Values of the Markov Chains

There are three types of parameters in a PROC MCMC program: the *model parameters* in the PARMS statement, the *random-effects parameters* in the RANDOM statement, and the *missing data variables* in the MODEL statement. The last category is used to model missing values in the input data set.

When the model parameters and random-effects parameters have missing initial values, PROC MCMC generates initial values based on the prior distributions. PROC MCMC either uses the mode value (the default) or draws a random number (if the INIT=RANDOM option is specified). For distributions that do not have modes, such as the uniform distribution, PROC MCMC uses the mean instead. In general, PROC MCMC avoids using starting values that are close to the boundary of support of the prior distribution. For example, the exponential prior has a mode at 0, and PROC MCMC starts an initial value at the mean. This

avoids some potential numerical problems. If you use the GENERAL or DGENERAL function in the PRIOR statements, you must provide initial values for those parameters.

With missing data variables, PROC MCMC uses the sample average of the nonmissing values (of the response variable) as the initial value. If all values of a particular variable are missing, PROC MCMC resorts to using the mode value or a random number from the sampling distribution (the likelihood), depending on the specification of the INIT= option.

To assign a different set of initial values to the model parameters, you use either the PARMS statements or programming statements within the BEGINCNST and ENDCNST statements. See the section "Assignments of Parameters" on page 6274 for more information about how to assign parameter values within the BEGINCNST and ENDCNST statements.

To assign initial values to the random-effects parameters, you can use the INIT= option in the RANDOM statement. Either you can give a constant value to all random-effects parameters that are associated with that statement (for example, use **init=3**), or you can assign values individually by providing a data set that stores different values for different parameters.

A mirroring INIT= option in the MODEL statement enables you to assign different initial values to the missing data variables.

If you use the PROPCOV= optimization option in the PROC MCMC statement, PROC MCMC starts the tuning at the optimized values. PROC MCMC overwrites the initial values that you might have provided at the beginning of the Markov chain unless you use the option INIT=REINIT.

Assignments of Parameters

In general, you cannot alter the values of any model parameters in PROC MCMC. For example, the following assignment statement produces an error:

```
parms alpha;
alpha = 27;
```

This restriction prevents incorrect calculation of the posterior density—assignments of parameters in the program would override the parameter values generated by PROC MCMC and lead to an incorrect value of the density function.

However, you can modify parameter values and assign initial values to parameters within the block defined by the BEGINCNST and ENDCNST statements. The following syntax is allowed:

```
parms alpha;
begincnst;
   alpha = 27;
endcnst;
```

The initial value of alpha is 27. Assignments within the BEGINCNST/ENDCNST block override initial values specified in the PARMS statement. For example, with the following statements, the Markov chain starts at alpha = 27, not 23.

```
parms alpha 23;
begincnst;
   alpha = 27;
endcnst;
```

This feature enables you to systematically assign initial values. Suppose that z is an array parameter of the same length as the number of observations in the input data set. You want to start the Markov chain with each z_i having a different value depending on the data set variable y. The following statements set $z_i = |y|$ for the first half of the observations and $z_i = 2.3$ for the rest:

```
/* a rather artificial input data set. */
data inputdata;
   do ind = 1 to 10;
      y = rand('normal');
      output;
   end;
run;

proc mcmc data=inputdata;
   array z[10];
   begincnst;
      if ind <= 5 then z[ind] = abs(y);
      else z[ind] = 2.3;
   endcnst;
   parms z:;
   prior z: ~ normal(0, sd=1);
   model general(0);
run;
```

Elements of z are modified as PROC MCMC executes the programming statements between the BEGINCNST and ENDCNST statements. This feature could be useful when you use the GENERAL function and you find that the PARMS statements are too cumbersome for assigning starting values.

Standard Distributions

The section "Univariate Distributions" on page 6276 (Table 80.7 through Table 80.36) lists all univariate distributions that PROC MCMC recognizes. The section "Multivariate Distributions" on page 6287 (Table 80.37 through Table 80.41) lists all multivariate distributions that PROC MCMC recognizes. With the exception of the multinomial distribution, all these distributions can be used in the MODEL, PRIOR, and HYPERPRIOR statements. The multinomial distribution is supported only in the MODEL statement. The RANDOM statement supports a limited number of distributions; see Table 80.4 for the complete list.

See the section "Using Density Functions in the Programming Statements" on page 6291 for information about how to use distributions in the programming statements. To specify an arbitrary distribution, you can use the GENERAL and DGENERAL functions. See the section "Specifying a New Distribution" on page 6290 for more details. See the section "Truncation and Censoring" on page 6294 for tips about how to work with truncated distributions and censoring data.

Univariate Distributions

Table 80.7 Beta Distribution

PROC specification	**beta**(a, b)
Density	$\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}$
Parameter restriction	$a > 0, b > 0$
Range	$\begin{cases} [0,1] & \text{when } a = 1, b = 1 \\ [0,1) & \text{when } a = 1, b \neq 1 \\ (0,1] & \text{when } a \neq 1, b = 1 \\ (0,1) & \text{otherwise} \end{cases}$
Mean	$\frac{a}{a+b}$
Variance	$\frac{ab}{(a+b)^2(a+b+1)}$
Mode	$\begin{cases} \frac{a-1}{a+b-2} & a > 1, b > 1 \\ 0 \text{ and } 1 & a < 1, b < 1 \\ 0 & \begin{cases} a < 1, b \geq 1 \\ a = 1, b > 1 \end{cases} \\ 1 & \begin{cases} a \geq 1, b < 1 \\ a > 1, b = 1 \end{cases} \\ \text{does not exist uniquely} & a = b = 1 \end{cases}$
Random number	If $\min(a, b) > 1$, see (Cheng 1978); if $\max(a, b) < 1$, see (Atkinson and Whittaker 1976) and (Atkinson 1979); if $\min(a, b) < 1$ and $\max(a, b) > 1$, see (Cheng 1978); if $a = 1$ or $b = 1$, use the inversion method; if $a = b = 1$, use a uniform random number generator.

Table 80.8 Binary Distribution

PROC specification	**binary**(p)
Density	$p^{\theta}(1-p)^{1-\theta}$
Parameter restriction	$0 \leq p \leq 1$
Range	$\begin{cases} \{0\} & \text{when } p = 0 \\ \{1\} & \text{when } p = 1 \\ \{0,1\} & \text{otherwise} \end{cases}$
Mean	(p)
Variance	$p(1-p)$
Mode	$\begin{cases} \{1\} & \text{when } p = 1 \\ \{0\} & \text{otherwise} \end{cases}$

Random number	Generate $u \sim \text{uniform}(0, 1)$. If $u \leq p$, $\theta = 1$; else, $\theta = 0$.

Table 80.9 Binomial Distribution

PROC specification	**binomial**(n, p)
Density	$\left(\begin{array}{c} n \\ \theta \end{array} \right) p^{\theta}(1-p)^{n-\theta}$
Parameter restriction	$n = 0, 1, 2, \ldots, 0 \leq p \leq 1$
Range	$\theta \in \{0, \ldots, n\}$
Mean	np
Variance	$np(1-p)$
Mode	$\lfloor (n+1)p \rfloor$

Table 80.10 Cauchy Distribution

PROC specification	**cauchy**(a, b)
Density	$\frac{1}{\pi}\left(\frac{b}{b^2+(\theta-a)^2}\right)$
Parameter restriction	$b > 0$
Range	$\theta \in (-\infty, \infty)$
Mean	Does not exist.
Variance	Does not exist.
Mode	a
Random number	Generate $u_1, u_2 \sim \text{uniform}(0, 1)$; let $v = 2u_2 - 1$. Repeat the procedure until $u_1^2 + v^2 < 1$. $y = v/u_1$ is a draw from the standard Cauchy, and $\theta = a + by$ (Ripley 1987).

Table 80.11 χ^2 Distribution

PROC specification	**chisq**(ν)
Density	$\frac{1}{\Gamma(\nu/2)2^{\nu/2}}\theta^{(\nu/2)-1}e^{-\theta/2}$
Parameter restriction	$\nu > 0$
Range	$\theta \in [0, \infty)$ if $\nu = 2$; $(0, \infty)$ otherwise.
Mean	ν
Variance	2ν
Mode	$\nu - 2$ if $\nu \geq 2$; does not exist otherwise.
Random number	χ^2 is a special case of the gamma distribution: $\theta \sim \text{gamma}(\nu/2, \text{scale=2})$ is a draw from the χ^2 distribution.

Table 80.12 Exponential χ^2 Distribution

PROC specification	**expchisq**(ν)
Density	$\frac{1}{\Gamma(\nu/2)2^{\nu/2}} \exp(\theta)^{\nu/2} \exp(-\exp(\theta)/2)$
Parameter restriction	$\nu > 0$
Range	$\theta \in (-\infty, \infty)$
Mode	$\log(\nu)$
Random number	Generate $x_1 \sim \chi^2(\nu)$, and $\theta = \log(x_1)$ is a draw from the exponential χ^2 distribution.
Relationship to the χ^2 distribution	$\theta \sim \chi^2(\nu) \Leftrightarrow \log(\theta) \sim \exp \chi^2(\nu)$

Table 80.13 Exponential Exponential Distribution

PROC specification	**expexpon**(scale = b)	**expexpon**(iscale = β)
Density	$\frac{1}{b} \exp(\theta) \exp(-\exp(\theta)/b)$	$\beta \exp(\theta) \exp(-\exp(\theta) \cdot \beta)$
Parameter restriction	$b > 0$	$\beta > 0$
Range	$\theta \in (-\infty, \infty)$	Same
Mode	$\log(b)$	$\log(1/\beta)$
Random number	Generate $x_1 \sim$ expon(scale=b), and $\theta = \log(x_1)$ is a draw from the exponential exponential distribution. Note that an exponential exponential distribution is not the same as the double exponential distribution.	
Relationship to the exponential distribution	$\theta \sim \text{expon}(b) \Leftrightarrow \log(\theta) \sim \text{expExpon}(b)$	

Table 80.14 Exponential Gamma Distribution

PROC specification	**expgamma**(a, scale = b)	**expgamma**(a, iscale = β)
Density	$\frac{1}{b^a \Gamma(a)} e^{a\theta} \exp(-e^{\theta}/b)$	$\frac{\beta^a}{\Gamma(a)} e^{a\theta} \exp(-e^{\theta} \cdot \beta)$
Parameter restriction	$a > 0, b > 0$	$a > 0, \beta > 0$
Range	$\theta \in (-\infty, \infty)$	Same
Mode	$\log(ab)$	$\log(a/\beta)$
Random number	Generate $x_1 \sim \text{gamma}(a, \text{scale} = b)$, and $\theta = \log(x_1)$ is a draw from the exponential gamma distribution.	
Relationship to the Γ distribution	$\theta \sim \text{gamma}(a, b) \Leftrightarrow \log(\theta) \sim \text{expGamma}(a, b)$	

Table 80.15 Exponential Inverse χ^2 Distribution

PROC specification	**expichisq**(ν)
Density	$\frac{1}{\Gamma(\frac{\nu}{2})2^{\nu/2}} \exp(-\nu\theta/2) \exp(-1/(2\exp(\theta)))$
Parameter restriction	$\nu > 0$
Range	$\theta \in (-\infty, \infty)$
Mode	$-\log(\nu)$
Random number	Generate $x_1 \sim i\chi^2(\nu)$, and $\theta = \log(x_1)$ is a draw from the exponential inverse χ^2 distribution.
Relationship to the $i\chi^2$ distribution	$\theta \sim i\chi^2(\nu) \Leftrightarrow \log(\theta) \sim \exp i\chi^2(\nu)$

Table 80.16 Exponential Inverse Gamma Distribution

PROC specification	**expigamma**(a, scale = b)	**expigamma**(a, iscale = β)
Density	$\frac{b^a}{\Gamma(a)} \exp(-\alpha\theta) \exp(-b/\exp(\theta))$	$\frac{1}{\beta^\alpha \Gamma(a)} \exp(-\alpha\theta) \exp(-\frac{1}{\beta \exp(\theta)})$
Parameter restriction	$a > 0, b > 0$	$a > 0, \beta > 0$
Range	$\theta \in (-\infty, \infty)$	Same
Mode	$-\log(a/b)$	$-\log(a\beta)$
Random number	Generate $x_1 \sim$ igamma(a, scale $= b$), and $\theta = \log(x_1)$ is a draw from the exponential inverse gamma distribution.	
Relationship to the $i\Gamma$ distribution	$\theta \sim$ igamma$(a, b) \Leftrightarrow \log(\theta) \sim$ eigamma(a, b)	

Table 80.17 Exponential Scaled Inverse χ^2 Distribution

PROC specification	**expsichisq**(ν, s)
Density	$\frac{(\frac{\nu}{2})^{\nu/2}}{\Gamma(\frac{\nu}{2})} s^\nu \exp(-\nu\theta/2) \exp(-\nu s^2/(2\exp(\theta)))$
Parameter restriction	$\nu > 0, s > 0$
Range	$\theta \in (-\infty, \infty)$
Mode	$\log(s^2)$
Random number	Generate $x_1 \sim si\chi^2(\nu, s)$, and $\theta = \log(x_1)$ is a draw from the exponential scaled inverse χ^2 distribution.
Relationship to the $si\chi^2$ distribution	$\theta \sim si\chi^2(\nu, s) \Leftrightarrow \log(\theta) \sim \exp si\chi^2(\nu, s)$

Table 80.18 Exponential Distribution

PROC specification	**expon**(scale = b)	**expon**(iscale = β)
Density	$\frac{1}{b}e^{-\theta/b}$	$\beta e^{-\beta\theta}$
Parameter restriction	$b > 0$	$\beta > 0$
Range	$\theta \in [0, \infty)$	Same
Mean	b	$1/\beta$
Variance	b^2	$1/\beta^2$
Mode	0	0
Random number	The exponential distribution is a special case of the gamma distribution: $\theta \sim$ gamma(1, scale $= b$) is a draw from the exponential distribution.	

Table 80.19 Gamma Distribution

PROC specification	**gamma**(a, scale = b)	**gamma**(a, iscale = β)
Density	$\frac{1}{b^a\Gamma(a)}\theta^{a-1}e^{-\theta/b}$	$\frac{\beta^a}{\Gamma(a)}\theta^{a-1}e^{-\beta\theta}$
Parameter restriction	$a > 0, b > 0$	$a > 0, \beta > 0$
Range	$\theta \in [0, \infty)$ if $a = 1$; $(0, \infty)$ otherwise.	Same
Mean	ab	a/β
Variance	ab^2	a/β^2
Mode	$(a-1)b$ if $a \geq 1$	$(a-1)/\beta$ if $a \geq 1$
Random number	See (McGrath and Irving 1973).	

Table 80.20 Geometric Distribution

PROC specification	**geo**(p)
Density *	$p(1-p)^\theta$
Parameter restriction	$0 < p \leq 1$
Range	$\theta \in \begin{cases} \{0, 1, 2, \ldots\} & 0 < p < 1 \\ \{0\} & p = 1 \end{cases}$
Mean	$\frac{1-p}{p}$
Variance	$\frac{1-p}{p^2}$
Mode	0
Random number	Based on samples obtained from a Bernoulli distribution with probability p until the first success.

*The random variable θ is the total number of failures in an experiment *before* the first success. This density function is not to be confused with another popular formulation, $p(1-p)^{\theta-1}$, which counts the total number of trials *until* the first success.

Table 80.21 Inverse χ^2 Distribution

PROC specification	**ichisq**(ν)
Density	$\frac{1}{\Gamma(\nu/2)2^{\nu/2}}\theta^{-(\nu/2+1)}e^{-1/(2\theta)}$
Parameter restriction	$\nu > 0$
Range	$\theta \in (0, \infty)$
Mean	$\frac{1}{\nu-2}$ if $\nu > 2$
Variance	$\frac{2}{(\nu-2)^2(\nu-4)}$ if $\nu > 4$
Mode	$\frac{1}{\nu+2}$
Random number	Inverse χ^2 is a special case of the inverse gamma distribution: $\theta \sim$ igamma($\nu/2$, iscale $= 2$) is a draw from the inverse χ^2 distribution.

Table 80.22 Inverse Gamma Distribution

PROC specification	**igamma**(a, scale = b)	**igamma**(a, iscale = β)
Density	$\frac{b^a}{\Gamma(a)}\theta^{-(a+1)}e^{-b/\theta}$	$\frac{1}{\beta^a\Gamma(a)}\theta^{-(a+1)}e^{-1/\beta\theta}$
Parameter restriction	$a > 0, b > 0$	$a > 0, \beta > 0$
Range	$\theta \in (0, \infty)$	Same
Mean	$\frac{b}{a-1}$ if $a > 1$	$\frac{1}{\beta(a-1)}$ if $a > 1$
Variance	$\frac{b^2}{(a-1)^2(a-2)}$	$\frac{1}{\beta^2(a-1)^2(a-2)}$
Mode	$\frac{b}{a+1}$	$\frac{1}{\beta(a+1)}$
Random number	Generate $x_1 \sim$ gamma(a, scale $= b$), and $\theta = 1/x_1$ is a draw from the igamma(a, iscale $= b$) distribution.	
Relationship to the gamma distribution	$\theta \sim$ gamma(a, iscale $= b$) $\Leftrightarrow 1/\theta \sim$ igamma(a, scale $= b$)	

Table 80.23 Laplace (Double Exponential) Distribution

PROC specification	**laplace**(a, scale = b)	**laplace**(a, iscale = β)
Density	$\frac{1}{2b}e^{-\lvert\theta-a\rvert/b}$	$\frac{\beta}{2}e^{-\beta\lvert\theta-a\rvert}$
Parameter restriction	$b > 0$	$\beta > 0$
Range	$\theta \in (-\infty, \infty)$	Same
Mean	a	a

Variance	$2b^2$	$2/\beta^2$
Mode	a	a
Random number	Inverse CDF. $F(\theta) = \begin{cases} \frac{1}{2}\exp\left(-\frac{a-\theta}{b}\right) & \theta < a \\ 1 - \frac{1}{2}\exp\left(-\frac{\theta-a}{b}\right) & \theta \geq a \end{cases}$. Generate $u_1, u_2 \sim$ uniform(0, 1). If $u_1 < 0.5, \theta = a + b\log(u_2)$; else $\theta = a - b\log(u_2)$. θ is a draw from the Laplace distribution.	

Table 80.24 Logistic Distribution

PROC specification	**logistic**(a, b)
Density	$\frac{\exp\left(-\frac{\theta-a}{b}\right)}{b\left(1+\exp\left(-\frac{\theta-a}{b}\right)\right)^2}$
Parameter restriction	$b > 0$
Range	$\theta \in (-\infty, \infty)$
Mean	a
Variance	$\frac{\pi^2 b^2}{3}$
Mode	a
Random number	Inverse CDF method with $F(\theta) = \left(1 + \exp(-\frac{\theta-a}{b})\right)^{-1}$. Generate $u \sim$ uniform(0, 1), and $\theta = a - b\log(1/u - 1)$ is a draw from the logistic distribution.

Table 80.25 Lognormal Distribution

PROC specification	**lognormal**$(\mu,$ sd = $s)$	**lognormal**$(\mu,$ var = $v)$	**lognormal**$(\mu,$ prec = $\tau)$
Density	$\frac{1}{\theta s\sqrt{2\pi}}\exp\left(-\frac{(\log\theta-\mu)^2}{2s^2}\right)$	$\frac{1}{\theta\sqrt{2\pi v}}\exp\left(-\frac{(\log\theta-\mu)^2}{2v}\right)$	$\frac{1}{\theta}\sqrt{\frac{\tau}{2\pi}}\exp\left(-\frac{\tau(\log\theta-\mu)^2}{2}\right)$
Parameter restriction	$s > 0$	$v > 0$	$\tau > 0$
Range	$\theta \in (0, \infty)$	Same	Same
Mean	$\exp(\mu + s^2/2)$	$\exp(\mu + v/2)$	$\exp(\mu + 1/(2\tau))$
Variance	$\exp(2(\mu + s^2))$ $-\exp(2\mu + s^2)$	$\exp(2(\mu + v))$ $-\exp(2\mu + v)$	$\exp(2(\mu + 1/\tau))$ $-\exp(2\mu + 1/\tau)$
Mode	$\exp(\mu - s^2)$	$\exp(\mu - v)$	$\exp(\mu - 1/\tau)$
Random number	Generate $x_1 \sim$ normal(0, 1), and $\theta = \exp(\mu + sx_1)$ is a draw from the lognormal distribution.		

Table 80.26 Negative Binomial Distribution

PROC specification	**negbin**(n, p)
Density	$\begin{pmatrix} \theta+n-1 \\ n-1 \end{pmatrix} p^n(1-p)^\theta$
Parameter restriction	$n = 1, 2, \ldots,$ and$0 < p \leq 1$
Range	$\theta \in \begin{cases} \{0, 1, 2, \ldots\} & 0 < p < 1 \\ \{0\} & p = 1 \end{cases}$
Mean	round$\left(\frac{n(1-p)}{p}\right)$
Variance	$\frac{n(1-p)}{p^2}$
Mode	$\begin{cases} 0 & n = 1 \\ \text{round}\left(\frac{(n-1)(1-p)}{p}\right) & n > 1 \end{cases}$
Random number	Generate $x_1 \sim$ gamma$(n, 1)$, and $\theta \sim$ Poisson$(x_1 \cdot (1-p)/p)$ (Fishman 1996).

Table 80.27 Normal Distribution

PROC specification	**normal**$(\mu,$ sd = $s)$	**normal**$(\mu,$ var = $v)$	**normal**$(\mu,$ prec = $\tau)$
Density	$\frac{1}{s\sqrt{2\pi}} \exp\left(-\frac{(\theta-\mu)^2}{2s^2}\right)$	$\frac{1}{\sqrt{2\pi v}} \exp\left(-\frac{(\theta-\mu)^2}{2v}\right)$	$\sqrt{\frac{\tau}{2\pi}} \exp\left(-\frac{\tau(\theta-\mu)^2}{2}\right)$
Parameter restriction	$s > 0$	$v > 0$	$\tau > 0$
Range	$\theta \in (-\infty, \infty)$	Same	Same
Mean	μ	Same	Same
Variance	s^2	v	$1/\tau$
Mode	μ	Same	Same

Table 80.28 NormalCAR Distribution

PROC specification	**normalcar**(**neighbors=**, **num=**, sd= $s)$	**normalcar**(**neighbors=**, **num=**, var= $v)$	**normalcar**(**neighbors=**, **num=**, prec= $\tau)$
Density	$\theta_i \vert \theta_{-i} \sim N(\sum_{j \in N(i)} \theta_j / m_i, s^2)$	$\theta_i \vert \theta_{-i} \sim N(\sum_{j \in N(i)} \theta_j / m_i, v)$	$\theta_i \vert \theta_{-i} \sim N(\sum_{j \in N(i)} \theta_j / m_i, 1/\tau)$
Notation	i is the area or site index, m_i is the number of neighbors to i, and $N(i)$ is the set of neighbors to i.		
Parameter restriction	$s > 0$	$v > 0$	$\tau > 0$
Range	$\theta \in (-\infty, \infty)$	Same	Same

Mean	average of neighbors	Same	Same
Variance	s^2	v	$1/\tau$
Mode	average of neighbors	Same	Same

Table 80.29 Pareto Distribution

PROC specification	**pareto**(a, b)
Density	$\frac{a}{b}\left(\frac{b}{\theta}\right)^{a+1}$
Parameter restriction	$a > 0, b > 0$
Range	$\theta \in [b, \infty)$
Mean	$\frac{ab}{a-1}$ if $a > 1$
Variance	$\frac{b^2 a}{(a-1)^2(a-2)}$ if $a > 2$
Mode	b
Random number	Inverse CDF method with $F(\theta) = 1 - (b/\theta)^a$. Generate $u \sim$ uniform$(0, 1)$, and $\theta = \frac{b}{u^{1/a}}$ is a draw from the Pareto distribution.
Useful transformation	$x = 1/\theta$ is Beta$(a, 1)\mathbf{I}\{x < 1/b\}$.

Table 80.30 Poisson Distribution

PROC specification	**poisson**(λ)
Density	$\frac{\lambda^\theta}{\theta!}\exp(-\lambda)$
Parameter restriction	$\lambda \geq 0$
Range	$\theta \in \begin{cases} \{0, 1, \ldots\} & \text{if } \lambda > 0 \\ \{0\} & \text{if } \lambda = 0 \end{cases}$
Mean	λ
Variance	λ, if $\lambda > 0$
Mode	round(λ)

Table 80.31 Scaled Inverse χ^2 Distribution

PROC specification	**sichisq**(v, s^2)
Density	$\frac{(s^2 v/2)^{v/2}}{\Gamma(v/2)}\theta^{-(v/2+1)}e^{-vs^2/(2\theta)}$
Parameter restriction	$v > 0, s > 0$
Range	$\theta \in (0, \infty)$
Mean	$\frac{v}{v-2}s^2$ if $v > 2$
Variance	$\frac{2v^2}{(v-2)^2(v-4)}s^4$ if $v > 4$

Mode	$\frac{\nu}{\nu+2}s^2$
Random number	Scaled inverse χ^2 is a special case of the inverse gamma distribution: $\theta \sim \text{igamma}(\nu/2, \text{scale} = (\nu s^2)/2)$ is a draw from the scaled inverse χ^2 distribution.

Table 80.32 *t* **Distribution**

PROC specification	**t**(μ, sd = s, ν)	**t**(μ, var = v, ν)	**t**(μ, prec = τ, ν)
Density	$\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})s\sqrt{\nu\pi}}(1+\frac{(\theta-\mu)^2}{\nu s^2})^{-\frac{\nu+1}{2}}$	$\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\nu\pi v}}(1+\frac{(\theta-\mu)^2}{\nu v})^{-\frac{\nu+1}{2}}$	$\frac{\Gamma(\frac{\nu+1}{2})\sqrt{\tau}}{\Gamma(\frac{\nu}{2})\sqrt{\nu\pi}}(1+\frac{\tau(\theta-\mu)^2}{\nu})^{-\frac{\nu+1}{2}}$
Parm restriction	$s > 0, \nu > 0$	$v > 0, \nu > 0$	$\tau > 0, \nu > 0$
Range	$\theta \in (-\infty, \infty)$	Same	Same
Mean	μ if $\nu > 1$	Same	Same
Variance	$\frac{\nu}{\nu-2}s^2$ if $\nu > 2$	$\frac{\nu}{\nu-2}v$ if $\nu > 2$	$\frac{\nu}{\nu-2}\frac{1}{\tau}$ if $\nu > 2$
Mode	μ	Same	Same
Random number	$x_1 \sim \text{normal}(0,1), x_2 \sim \chi^2(d)$, and $\theta = m + \sigma x_1\sqrt{d/x_2}$ is a draw from the t distribution.		

Table 80.33 Table (Categorical) Distribution

PROC specification	**table**(**p**), where $\mathbf{p} = \{p_i\}$, for $i = 1, 2, \ldots, k$
Density	$f(\theta = i) = p_i$
Parameter restriction	$\sum_i^k p_i = 1$ with all $p_i > 0$
Range	$\theta \in \{1, 2, \ldots, k\}$
Mode	i such that $p_i = \max(p_1, \ldots, p_k)$
Random number	Inverse CDF method with $F(\theta = i) = \sum_{j=1}^{i} p_j$.

Table 80.34 Uniform Distribution

PROC specification	**uniform**(a, b)
Density	$\begin{cases} \frac{1}{a-b} & \text{if } a > b \\ \frac{1}{b-a} & \text{if } b > a \\ 1 & \text{if } a = b \end{cases}$
Parameter restriction	none
Range	$\theta \in [a, b]$

Mean	$\frac{a+b}{2}$
Variance	$\frac{\lvert b-a\rvert^2}{12}$
Mode	Does not exist
Random number	Mersenne Twister (Matsumoto and Kurita 1992, 1994; Matsumoto and Nishimura 1998)

Table 80.35 Wald Distribution

PROC specification	**wald**(μ, λ)
Density	$\sqrt{\frac{\lambda}{2\pi\theta^3}} \exp\left(\frac{-\lambda(\theta-\mu)^2}{2\mu^2\theta}\right)$
Parameter restriction	$\mu > 0, \lambda > 0$
Range	$\theta \in (0, \infty)$
Mean	μ
Variance	μ^3/λ
Mode	$\mu\left[\left(1+\frac{9\mu^2}{4\lambda^2}\right)^{1/2} - \frac{3\mu}{2\lambda}\right]$
Random number	Generate $v_0 \sim \chi^2_{(1)}$. Let $x_1 = \mu + \frac{\mu^2 v_0}{2\lambda} - \frac{\mu}{2\lambda}\sqrt{4\mu\lambda v_0 + \mu^2 v_0^2}$ and $x_2 = \mu^2/x_1$. Perform a Bernoulli trial, $w \sim$ Bernoulli$(\frac{\mu}{\mu+x_1})$. If $w = 1$, choose $\theta = x_1$; otherwise, choose $\theta = x_2$ (Michael, Schucany, and Haas 1976).

Table 80.36 Weibull Distribution

PROC specification	**weibull**(μ, c, σ)
Density	$\exp\left(-\left(\frac{\theta-\mu}{\sigma}\right)^c\right) \frac{c}{\sigma}\left(\frac{\theta-\mu}{\sigma}\right)^{c-1}$
Parameter restriction	$c > 0, \sigma > 0$
Range	$\theta \in [\mu, \infty)$ if $c = 1$; (μ, ∞) otherwise
Mean	$\mu + \sigma\Gamma(1 + 1/c)$
Variance	$\sigma^2[\Gamma(1 + 2/c) - \Gamma^2(1 + 1/c)]$
Mode	$\mu + \sigma(1 - 1/c)^{1/c}$ if $c > 1$
Random number	Inverse CDF method with $F(\theta) = 1 - \exp\left(-\left(-\frac{\theta-\mu}{\sigma}\right)^c\right)$. Generate $u \sim$ uniform$(0, 1)$, and $\theta = \mu + \sigma \cdot (-\ln u)^{1/c}$ is a draw from the Weibull distribution.

Multivariate Distributions

Table 80.37 Dirichlet Distribution

PROC specification	$\boldsymbol{\theta} \sim \mathbf{dirich}(\boldsymbol{\alpha})$, where $\boldsymbol{\theta} = \{\theta_i\}, \boldsymbol{\alpha} = \{\alpha_i\}$, for $i = 1, 2, \ldots, k$
Density	$\frac{\Gamma(\alpha_0)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} \theta_i^{\alpha_i - 1}$, where $\alpha_0 = \sum_{i=1}^{k} \alpha_i$
Parameter restriction	$\alpha_i > 0$
Range	$\theta_i > 0, \sum_{i=1}^{k} \theta_i = 1$
Mean	α_j / α_0
Mode	$(\alpha_j - 1) / (\alpha_0 - k)$

Table 80.38 Inverse Wishart Distribution

PROC specification	$\boldsymbol{\theta} \sim \mathbf{iwishart}(\nu, \mathbf{S})$, both $\boldsymbol{\theta}$ and $\mathbf{S}$ are $k \times k$ matrices
Density	$\left(2^{\frac{\nu k}{2}} \pi^{\frac{k(k-1)}{4}} \prod_{i=1}^{k} \Gamma\left(\frac{\nu+1-i}{2}\right)\right)^{-1} \lvert\mathbf{S}\rvert^{\frac{\nu}{2}} \lvert\boldsymbol{\theta}\rvert^{-\frac{\nu+k+1}{2}} \exp\left(-\frac{1}{2}\mathrm{tr}(\mathbf{S}\boldsymbol{\theta}^{-1})\right)$
Parameter restriction	$\mathbf{S}$ must be symmetric and positive definite; $\nu > k - 1$
Range	$\boldsymbol{\theta}$ is symmetric and positive definite
Mean	$\mathbf{S} / (\nu - k - 1)$
Mode	$\mathbf{S} / (\nu + k + 1)$

Table 80.39 Multivariate Normal Distribution

PROC specification	$\boldsymbol{\theta} \sim \mathbf{mvn}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\theta} = \{\theta_k\}, \boldsymbol{\mu} = \{\mu_k\}$, for $i = 1, 2, \ldots, k$, and $\boldsymbol{\Sigma}$ is a $k \times k$ variance matrix
Density	$\exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu})\right) \Big/ \sqrt{(2\pi)^k \lvert\boldsymbol{\Sigma}\rvert}$
Parameter restriction	$\boldsymbol{\Sigma}$ must be symmetric and positive definite
Range	$-\infty < \theta_i < \infty$
Mean	$\boldsymbol{\mu}$
Mode	$\boldsymbol{\mu}$

Table 80.40 Autoregressive Multivariate Normal Distribution

PROC specification	$\boldsymbol{\theta} \sim \mathbf{MVNAR}(\boldsymbol{\mu}, \text{sd=}\sigma, \rho)$	$\boldsymbol{\theta} \sim \mathbf{MVNAR}(\boldsymbol{\mu}, \text{var=}\sigma^2, \rho)$	$\boldsymbol{\theta} \sim \mathbf{MVNAR}(\boldsymbol{\mu}, \text{prec=}1/\sigma^2, \rho)$

Density	$\exp\left(-\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu})'(\sigma^2\boldsymbol{\Sigma})^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})\right) \Big/ \sqrt{(2\pi)^k \left\lvert(\sigma^2\boldsymbol{\Sigma})\right\rvert}$ where $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & \rho & \rho^2 & \rho^3 & \cdots & \rho^k \\ \rho & 1 & \rho & \rho^2 & \cdots & \rho^{k-1} \\ \rho^2 & \rho & 1 & \rho & \cdots & \rho^{k-2} \\ \rho^3 & \rho^2 & \rho & 1 & \cdots & \rho^{k-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^k & \rho^{k-1} & \rho^{k-2} & \rho^{k-3} & \cdots & 1 \end{bmatrix}$
Parameter restriction	$\sigma > 0$ and $-1 < \rho < 1$
Range	$-\infty < \theta_i < \infty$
Mean	$\boldsymbol{\mu}$
Mode	$\boldsymbol{\mu}$
Special Case	When $\rho = 0$, the distribution simplifies to $\mathbf{mvn}(\boldsymbol{\mu}, \sigma^2 \cdot \mathbf{I}_k)$, where $\mathbf{I}_k$ denotes the $k \times k$ identity matrix

Table 80.41 Multinomial Distribution

PROC specification	$\boldsymbol{\theta} \sim \mathbf{multinom}(\mathbf{p})$, where $\boldsymbol{\theta} = \{\theta_i\}$ and $\mathbf{p} = \{p_i\}$, for $i = 1, 2, \ldots, k$
Density	$\frac{n!}{\theta_1 \cdots \theta_k} p_1^{\theta_1} \cdots p_k^{\theta_k}$, where $\sum_i^k \theta_i = n$
Parameter restriction	$\sum_i^k p_i = 1$ with all $p_i > 0$
Range	$\theta_i \in \{0, \ldots, n\}$, nonnegative integers
Mean	$n \cdot \mathbf{p}$

Usage of Multivariate Distributions

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcmvn.sas.)

The following simple example illustrates the usage of the multivariate distributions in PROC MCMC. Suppose you are interested in estimating the mean and covariance of multivariate data using this multivariate normal model:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \text{MVN}\left(\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix}\right)$$

where

$$\boldsymbol{\mu} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} 2.4 & 3 \\ 3 & 8.1 \end{pmatrix}$$

You can use the following independent prior on μ and Σ:

$$\begin{aligned}\mu &\sim \text{MVN}\left(\mu_0 = \begin{pmatrix}0\\0\end{pmatrix}, \Sigma_0 = \begin{pmatrix}100 & 0\\0 & 100\end{pmatrix}\right)\\ \Sigma &\sim \text{iWishart}\left(\nu = 2, \text{S} = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}\right)\end{aligned}$$

The following IML procedure statements simulate 100 random multivariate normal samples:

```
title 'An Example that Uses Multivariate Distributions';
proc iml;
   N = 100;
   Mean = {1 2};
   Cov =  {2.4 3, 3 8.1};
   call randseed(1);
   x = RANDNORMAL( N, Mean, Cov );

   SampleMean = x[:,];
   n = nrow(x);
   y = x - repeat( SampleMean, n );
   SampleCov = y`*y / (n-1);
   print SampleMean Mean, SampleCov Cov;

   cname = {"x1", "x2"};
   create inputdata from x [colname = cname];
   append from x;
   close inputdata;
quit;
```

Figure 80.13 prints the sample mean and covariance of the simulated data, in addition to the true mean and covariance matrix.

Figure 80.13 Simulated Multivariate Normal Data

An Example that Uses Multivariate Distributions

SampleMean		Mean	
0.9987751	2.115693	1	2

SampleCov		Cov	
2.8252975	3.7190704	2.4	3
3.7190704	9.2916805	3	8.1

The following PROC MCMC statements estimate the posterior mean and covariance of the multivariate normal data:

```
proc mcmc data=inputdata seed=17 nmc=3000 diag=none;
   ods select PostSumInt;
   array data[2] x1 x2;
   array mu[2];
   array Sigma[2,2];
   array mu0[2] (0 0);
```

```
   array Sigma0[2,2] (100 0 0 100);
   array S[2,2] (1 0 0 1);
   parm mu Sigma;
   prior mu ~ mvn(mu0, Sigma0);
   prior Sigma ~ iwish(2, S);
   model data ~ mvn(mu, Sigma);
run;
```

To use the multivariate distribution, you must specify parameters (or random variables in the MODEL statement) in an array form. The first ARRAY statement creates an one-dimensional array data, which contains two numeric variables, x1 and x2, from the input data set. The data variable is your response variable. The subsequent statements defines two array-parameters (mu and Sigma) and three constant array-hyperparameters (mu0, Sigma0, and S). The PARMS statement declares mu and Sigma to be model parameters. The two PRIOR statements specify the multivariate normal and inverse Wishart distributions as the prior for mu and Sigma, respectively. The MODEL statement specifies the multivariate normal likelihood with data as the random variable, mu as the mean, and Sigma as the covariance matrix.

Figure 80.14 lists the estimated posterior statistics for the parameters.

Figure 80.14 Estimated Mean and Covariance

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
mu1	3000	0.9941	0.1763	0.6338	1.3106
mu2	3000	2.1135	0.3112	1.4939	2.7165
Sigma1	3000	2.8726	0.4084	2.1001	3.6723
Sigma2	3000	3.7573	0.6418	2.5791	5.0223
Sigma3	3000	3.7573	0.6418	2.5791	5.0223
Sigma4	3000	9.3987	1.3224	7.0155	12.0969

Specifying a New Distribution

To work with a new density that is not listed in the section "Standard Distributions" on page 6275, you can use the GENERAL and DGENERAL functions. The letter "D" stands for discrete. The new distributions have to be specified on the logarithm scale.

Suppose you want to use the inverse beta distribution:

$$p(\alpha|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)+\Gamma(b)} \cdot \alpha^{(a-1)} \cdot (1+\alpha)^{-(a+b)}$$

The following statements in PROC MCMC define the density on its log scale:

```
a = 3; b = 5;
const = lgamma(a + b) - lgamma(a) - lgamma(b);
lp = const + (a - 1) * log(alpha) - (a + b) * log(1 + alpha);
prior alpha ~ general(lp);
```

The symbol lp is the expression for the log of an inverse beta (a = 3, b = 5). The function **general(lp)** assigns that distribution to alpha. The constant term, const, can be omitted because the Markov simulation requires only the log of the density kernel.

You can use the GENERAL function to specify a distribution for a single variable or for multiple variables. It is important to emphasize that the argument *lp* is an expression for the log of the joint distribution for these variables. On the contrary, any standard distribution is applied separately to each random variable in the statement.

When you use the GENERAL function in the MODEL statement, you do not need to specify the dependent variable on the left of the tilde ∼. The log-likelihood function takes the dependent variable into account; hence, there is no need to explicitly state the dependent variable in the MODEL statement. However, in the PRIOR and RANDOM statements, you need to explicitly state the parameter names and a tilde with the GENERAL function.

You can specify any distribution function by using the GENERAL and DGENERAL functions as long as the distribution function is programmable with SAS statements. When the function is used in the PRIOR statements, you must supply initial values in either the PARMS statement or within the BEGINCNST and ENDCNST statements. See the sections "PARMS Statement" on page 6249 and "BEGINCNST/ENDCNST Statement" on page 6234. When the function is used in the RANDOM statement, you must use the INITIAL= option in the RANDOM statement to supply initial values

NOTE: PROC MCMC does not verify that the GENERAL function you specify is a valid distribution—that is, an integrable density. You must use the function with caution.

Using Density Functions in the Programming Statements

Density Functions in PROC MCMC

PROC MCMC has a number of internally defined log-density functions for univariate and multivariate distributions. These functions have the basic form of LPDF*dist*(*x*, *parm-list*), where *dist* is the name of the distribution (see Table 80.42 for univariate distributions and Table 80.43 for multivariate distributions). The argument *x* is the random variable, and *parm-list* is the list of parameters.

In addition, the univariate functions allow for optional boundary arguments, such as LPDF*dist*(*x*, *parm-list*, < *lower* >, < *upper* >), where *lower* and *upper* are optional but positional boundary arguments. With the exception of the Bernoulli and uniform distribution, you can specify limits on all univariate distributions.

To set a lower bound on the normal density:

```
lpdfnorm(x, 0, 1, -2);
```

To set just an upper bound, specify a missing value for the lower bound argument:

```
lpdfnorm(x, 0, 1, ., 2);
```

Leaving both limits out gives you the unbounded density. You can also specify both bounds:

```
lpdfnorm(x, 0, 1);
lpdfnorm(x, 0, 1, -3, 4);
```

See Table 80.42 for the function names of univariate distributions and Table 80.43 for multivariate distributions.

Table 80.42 Logarithm of Univariate Density Functions in PROC MCMC

Distribution Name	Function Call
Beta	**lpdfbeta(***x, a, b,*< *lower*>, < *upper*>**);**
Binary	**lpdfbern(***x, p***);**
Binomial	**lpdfbin(***x, n,p,* < *lower*>, < *upper*>**);**
Cauchy	**lpdfcau(***x, loc, scale,* < *lower*>, < *upper*>**);**
χ^2	**lpdfchisq(***x, df,*< *lower*>, < *upper*>**);**
Exponential χ^2	**lpdfechisq(***x, df,* < *lower*>, < *upper*>**);**
Exponential gamma	**lpdfegamma(***x, sp,scale,* < *lower*>, < *upper*>**);**
Exponential exponential	**lpdfeexpon(***x, scale,*< *lower*>, < *upper*>**);**
Exponential inverse χ^2	**lpdfeichisq(***x, df,* < *lower*>, < *upper*>**);**
Exponential inverse gamma	**lpdfeigamma(***x, sp, scale,* < *lower*>, < *upper*>**);**
Exponential scaled inverse χ^2	**lpdfesichisq(***x, df, scale,* < *lower*>, < *upper*>**);**
Exponential	**lpdfexpon(***x, scale,* < *lower*>, < *upper*>**);**
Gamma	**lpdfgamma(***x, sp, scale,* < *lower*>, < *upper*>**);**
Geometric	**lpdfgeo(***x, p,* < *lower*>, < *upper*>**);**
Inverse χ^2	**lpdfichisq(***x, df,* < *lower*>, < *upper*>**);**
Inverse gamma	**lpdfigamma(***x, sp, scale,* < *lower*>, < *upper*>**);**
Laplace	**lpdfdexp(***x, loc, scale,* < *lower*>, < *upper*>**);**
Logistic	**lpdflogis(***x, loc, scale,* < *lower*>, < *upper*>**);**
Lognormal	**lpdflnorm(***x, loc, sd,* < *lower*>, < *upper*>**);**
Negative binomial	**lpdfnegbin(***x, n, p,* < *lower*>, < *upper*>**);**
Normal	**lpdfnorm(***x, mu, sd,* < *lower*>, < *upper*>**);**

Table 80.42 *continued*

Distribution Name	Function Call
Pareto	**lpdfpareto(***x, sp, scale, < lower >, < upper >***);**
Poisson	**lpdfpoi(***x, mean, < lower >, < upper >***);**
Scaled inverse χ^2	**lpdfsichisq(***x, df, scale, < lower >, < upper >***);**
t	**lpdft(***x, mu, sd, df, < lower >,< upper >***);**
Uniform	**lpdfunif(***x, a, b***);**
Wald	**lpdfwald(***x, mean, scale, < lower >, < upper >***);**
Weibull	**lpdfwei(***x, loc, sp, scale, < lower >, < upper >***);**

In the multivariate log-density functions, arrays must be used in place for the random variable and parameters in the model.

Table 80.43 Logarithm of Multivariate Density Functions in PROC MCMC

Distribution Name	Function Call
Dirichlet	**lpdfdirich(***x_array, alpha_array***);**
Inverse Wishart	**lpdfiwish(***x_array, df, S_array***);**
Multivariate normal	**lpdfmvn(***x_array, mu_array, cov_array***);**
Multinomial	**lpdfmnom(***x_array, p_array***);**

Standard Distributions, the LOGPDF Functions, and the LPDF*dist* Functions

Standard distributions listed in the section "Standard Distributions" on page 6275 are *names* only, and they can be used only in the MODEL, PRIOR, and HYPERPRIOR statements to specify either a prior distribution or a conditional distribution of the data given parameters. They do not return any values, and you cannot use them in the programming statements.

The LOGPDF functions are DATA step functions that compute the logarithm of various probability density (mass) functions. For example,

logpdf(*"beta", x, 2, 15***);**

returns the log of a beta density with parameters a = 2 and b = 15, evaluated at *x*. All the LOGPDF functions are supported in PROC MCMC.

The LPDF*dist* functions are unique to PROC MCMC. They compute the logarithm of various probability density (mass) functions. The functions are the same as the LOGPDF functions when it comes to calculating the log density. For example,

lpdfbeta(*x, 2,15***);**

returns the same value as

logpdf(*"beta", x, 2, 15***);**

The LPDF*dist* functions cover a greater class of probability density functions, and the univariate distribution functions take the optional but positional boundary arguments. There are no corresponding LCDF*dist* or LSDF*dist* functions in PROC MCMC. To work with the cumulative probability function or the survival functions, you need to use the LOGCDF and the LOGSDF DATA step functions.

Truncation and Censoring

Truncated Distributions

To specify a truncated distribution, you can use the LOWER= and/or UPPER= options. Almost all of the univariate standard distributions, including the GENERAL and DGENERALfunctions, take these optional truncation arguments. The binary, the uniform, and the tabled distributions do not support these truncation options. Multivariate distributions, such as the multivariate normal, do not support these options neither.

For example, you can specify the following:

```
prior alpha ~ normal(mean = 0, sd = 1, lower = 3, upper = 45);
```

or

```
parms beta;
a = 3; b = 7;
ll = (a + 1) * log(b / beta);
prior beta ~ general(ll, upper = b + 17);
```

The preceding statements state that if beta is less than b+17, the log of the prior density is ll, as calculated by the equation; otherwise, the log of the prior density is missing—the log of zero.

When the same distribution is applied to multiple parameters in a PRIOR statement, the LOWER= and UPPER= truncations apply to all parameters in that statement. For example, the following statements define a Poisson density for theta and gamma:

```
parms theta gamma;
lambda = 7;
l1 = theta * log(lambda) - lgamma(1 + theta);
l2 = gamma * log(lambda) - lgamma(1 + gamma);
ll = l1 + l2;
prior theta gamma ~ dgeneral(ll, lower = 1);
```

The LOWER=1 condition is applied to both theta and gamma, meaning that for the assignment to ll to be meaningful, both theta and gamma have to be greater than 1. If either of the parameters is less than 1, the log of the joint prior density becomes a missing value.

PROC MCMC calculates the normalizing constant in all truncated distributions (with exception to the GENERAL and the DGENERAL functions), and you can use parameters in the LOWER= or UPPER= option.

Note that if you use either the GENERAL or DGENERAL function, you must compute the normalizing constant in cases where it is required. A truncated distribution has the probability distribution

$$p(\theta|a < \theta < b) = \frac{p(\theta)}{F(b) - F(a)}$$

where $p(\cdot)$ is the density function and $F(\cdot)$ is the cumulative distribution function. In SAS functions, $p(\cdot)$ is the probability density function and $F(\cdot)$ is the cumulative distribution function. The following example shows how to construct a truncated gamma prior on theta, with SHAPE=3, SCALE=2, LOWER=A, and UPPER=B:

```
lp = logpdf('gamma', theta, 3, 2)
        - log(cdf('gamma', a, 3, 2) - cdf('gamma', b, 3, 2));
prior theta ~ general(lp);
```

This density specification is different from the following more naive definition, without taking into account the normalizing constant:

```
lp = logpdf('gamma', theta, 3, 2);
prior theta ~ general(lp, lower=a, upper=b);
```

If a or b is a parameter, you get very different results from the two formulations.

Censoring

You can use either of two approaches to model censored data. One is to specify the marginal distribution, and the other is to treat the censored data as missing values.

Suppose you partition the data into four categories: uncensored (with observation x), left-censored (with observation xl), right-censored (with observation xr), and interval-censored (with observations xl and xr). The likelihood is the normal distribution with mean mu and standard deviation s. The following statements construct the corresponding log likelihood for the observed data:

```
if uncensored then
   ll = logpdf('normal', x, mu, s);
else if leftcensored then
   ll = logcdf('normal', xl, mu, s);
else if rightcensored then
   ll = logsdf('normal', xr, mu, s);
else /* this is the case of interval censored. */
   ll = log(cdf('normal', xr, mu, s) - cdf('normal', xl, mu, s));
model general(ll);
```

Alternatively, you can treat censored data as missing values and impute the values in the Markov chain. In the following statement, the CLOWER= and CUPPER= options are the censoring indicators:

```
model x ~ normal(mu, sd=1, clower=xl, cupper=xr);
```

Missing x values become parameters, and PROC MCMC samples according to the censoring information. Specify the MISSING=ACMODELY option in the PROC MCMC statement if the xl or xr variables contain missing values. This option enables PROC MCMC to draw missing response variables without discarding observations that have missing covariates. By default, PROC MCMC models missing values but throws away observations that have missing values in nonresponse variables.

See "Example 80.17: Normal Regression with Interval Censoring" on page 6456.

Some Useful SAS Functions

Table 80.44 Some Useful SAS Functions

SAS Function	Definition
abs(*x***);**	$\lvert x \rvert$
airy(*x***);**	Returns the value of the AIRY function.
beta(*x1, x2***);**	$\int_0^1 z^{x1-1}(1-z)^{x2-1}dz$
call logistic(*x***);**	$\frac{\exp(x)}{1+\exp(x)}$
call softmax(*x1, . . . , xn***);**	Each element is replaced by $\exp(x_j)/\sum \exp(x_j)$
call stdize(*x1, . . . , xn***);**	Standardize values
cdf();	Cumulative distribution function
cdf(*'normal', x, 0, 1***);**	Standard normal cumulative distribution function
comb(*x1, x2***);**	$\frac{x1!}{x2!(x1-x2)!}$
constant(*'.'***);**	Calculate commonly used constants
cos(*x***);**	cosine(x)
css(*x1, . . . , xn***);**	$\sum_i (x_i - \bar{x})^2$
cv(*x1, . . . , xn***);**	std(x) / mean(x) * 100
dairy(*x***);**	Derivative of the AIRY function
dimN(*m***);**	Returns the numbers of elements in the Nth dim of array *m*
x1 eq x2	Returns 1 if x1 = x2; 0 otherwise
x1x2**	$x1^{x2}$
geomean(*x1, . . . , xn***);**	$\exp\left(\frac{\log(x_1)+\cdots+\log(x_n)}{n}\right)$
difN(*x***);**	Returns differences between the argument and its Nth lag
digamma(*x1***);**	$\frac{\Gamma'(x1)}{\Gamma(x1)}$
erf(*x***);**	$\frac{2}{\sqrt{\pi}}\int_0^x \exp(-z^2)dz$
erfc(*x***);**	1 – erf(x)
fact(*x***);**	$x!$
floor(*x***);**	Greatest integer $\leq x$
gamma(*x***);**	$\int_0^\infty z^{x-1}\exp(-1)dz$
harmean(*x1, . . . , xn***);**	$\frac{n}{1/x_1+\cdots 1/x_n}$
ibessel(*nu, x, kode***);**	Modified Bessel function of order *nu* evaluated at x
jbessel(*nu, x***);**	Bessel function of order *nu* evaluated at x
lagN(*x***);**	Returns values from a queue

Table 80.44 *continued*

SAS Function	Definition
largest(*k, x1, . . . , xn***);**	Returns the kth largest element
lgamma(*x***);**	$\ln(\Gamma(x))$
lgamma(*x+1***);**	$\ln(x!)$
log(*x, logN(x)***);**	$\ln(x)$
logbeta(*x1, x2***);**	lgamma(x_1) + lgamma(x_2) - lgamma($x_1 + x_2$)
logcdf();	Log of a left cumulative distribution function
logpdf();	Log of a probability density (mass) function
logsdf();	Log of a survival function
max(*x1, x2***);**	Returns x_1 if $x_1 > x_2$; x_2 otherwise
mean(*of x1–xn***);**	$\sum_i x_i / n$
median(*of x1–xn***);**	Returns the median of nonmissing values
min(*x1, x2***);**	Returns x_1 if $x_1 < x_2$; x_2 otherwise
missing(*x***);**	Returns 1 if x is missing; 0 otherwise
mod(*x1, x2***);**	Returns the remainder from x_1/x_2
n(*x1, . . . , xn***);**	Returns number of nonmissing values
nmiss(*of y1–yn***);**	Number of missing values
quantile();	Computes the quantile from a specific distribution
pdf();	Probability density (mass) functions
perm(*n, r***);**	$\frac{n!}{(n-r)!}$
put();	Returns a value that uses a specified format
round(*x***);**	Rounds x
rms(*of x1–xn***);**	$\sqrt{\frac{x_1^2+\cdots x_n^2}{n}}$
sdf();	Survival function
sign(*x***);**	Returns –1 if $x < 0$; 0 if $x = 0$; 1 if $x > 0$
sin(*x***);**	sine(x)
smallest(*s, x1, . . . , en* **);**	Returns the sth smallest component of $x_1, \ldots, x_n$
sortn(*of x1-xn***);**	Sorts the values of the variables
sqrt(*x***);**	$\sqrt{x}$
std(*x1, . . . , xn)* **);**	Standard deviation of $x_1, \ldots, x_n$ (n-1 in denominator)
sum(*of x:***);**	$\sum_i x_i$
trigamma(*x***);**	Derivative of the DIGAMMA(x) function
uss(*of x1–xn***);**	Uncorrected sum of squares

Here are examples of some commonly used transformations:

- logit

```
mu = beta0 + beta1 * z1;
call logistic(mu);
```

- log

```
w = beta0 + beta1 * z1;
mu = exp(w);
```

- probit

```
w = beta0 + beta1 * z1;
mu = cdf(`normal', w, 0, 1);
```

- cloglog

```
w = beta0 + beta1 * z1;
mu = 1 - exp(-exp(w));
```

Matrix Functions in PROC MCMC

The MCMC procedure provides you with a number of CALL routines for performing simple matrix operations on declared arrays. With the exception of FILLMATRIX, IDENTITY, and ZEROMATRIX, the CALL routines listed in Table 80.45 do not support matrices or arrays that contain missing values.

Table 80.45 Matrix Functions in PROC MCMC

CALL Routine	Description
ADDMATRIX	Performs an element-wise addition of two matrices or of a matrix and a scalar.
CHOL	Calculates the Cholesky decomposition for a particular symmetric matrix.
DET	Calculates the determinant of a specified matrix, which must be square.
ELEMMULT	Performs an element-wise multiplication of two matrices.
FILLMATRIX	Replaces all of the element values of the input matrix with the specified value. You can use this routine with multidimensional numeric arrays.
IDENTITY	Converts the input matrix to an identity matrix. Diagonal element values of the matrix are set to 1, and the rest of the values are set to 0.
INV	Calculates a matrix that is the inverse of the input matrix. The input matrix must be a square, nonsingular matrix.
MULT	Calculates the matrix product of two input matrices.
SUBTRACTMATRIX	Performs an element-wide subtraction of two matrices or of a matrix and a scalar.
TRANSPOSE	Returns the transpose of a matrix.
ZEROMATRIX	Replaces all of the element values of the numeric input matrix with 0.

ADDMATRIX CALL Routine

The ADDMATRIX CALL routine performs an element-wise addition of two matrices or of a matrix and a scalar.

The syntax of the ADDMATRIX CALL routine is

CALL ADDMATRIX (*X*, *Y*, *Z***) ;**

where

X specifies a scalar or an input matrix with dimensions $m \times n$ (that is, $X[m, n]$)

Y specifies a scalar or an input matrix with dimensions $m \times n$ (that is, $Y[m, n]$)

Z specifies an output matrix with dimensions $m \times n$ (that is, $Z[m, n]$)

such that

$$Z = X + Y$$

CHOL CALL Routine

The CHOL CALL routine calculates the Cholesky decomposition for a particular symmetric matrix.

The syntax of the CHOL CALL routine is

CALL CHOL (*X*, *Y* <, *validate*>**) ;**

where

X specifies a symmetric positive-definite input matrix with dimensions $m \times m$ (that is, X[m, m])

Y is a variable that contains the Cholesky decomposition and specifies an output matrix with dimensions $m \times m$ (that is, $Y[m, m]$)

validate specifies an optional argument that can increase the processing speed by avoiding error checking:

If *validate* = 0 or is not specified, then the matrix *X* is checked for symmetry.

If *validate* = 1, then the matrix *X* is assumed to be symmetric.

such that

$$X = YY^*$$

where *Y* is a lower triangular matrix with strictly positive diagonal entries and Y^* denotes the conjugate transpose of *Y*.

Both input and output matrices must be square and have the same dimensions. If *X* is symmetric positive-definite, *Y* is a lower triangle matrix. If *X* is not symmetric positive-definite, *Y* is filled with missing values.

DET CALL Routine

The determinant, the product of the eigenvalues, is a single numeric value. If the determinant of a matrix is zero, then that matrix is singular (that is, it does not have an inverse). The routine performs an LU decomposition and collects the product of the diagonals.

The syntax of the DET CALL routine is

CALL DET (*X***, *a*) ;**

where

X specifies an input matrix with dimensions $m \times m$ (that is, $X[m, m]$)

a specifies the returned determinate value

such that

$a = |X|$

ELEMMULT CALL Routine

The ELEMMULT CALL routine performs an element-wise multiplication of two matrices.

The syntax of the ELEMMULT CALL routine is

CALL ELEMMULT (*X***, *Y*, *Z*) ;**

where

X specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$)

Y specifies an input matrix with dimensions $m \times n$ (that is, $Y[m, n]$)

Z specifies an output matrix with dimensions $m \times n$ (that is, $Z[m, n]$)

FILLMATRIX CALL Routine

The FILLMATRIX CALL routine replaces all of the element values of the input matrix with the specified value. You can use the FILLMATRIX CALL routine with multidimensional numeric arrays.

The syntax of the FILLMATRIX CALL routine is

CALL FILLMATRIX (*X***, *Y*) ;**

where

X specifies an input numeric matrix

Y specifies the numeric value that is used to fill the matrix

IDENTITY CALL Routine

The IDENTITY CALL routine converts the input matrix to an identity matrix. Diagonal element values of the matrix are set to 1, and the rest of the values are set to 0.

The syntax of the IDENTITY CALL routine is

CALL IDENTITY (*X***) ;**

where

X specifies an input matrix with dimensions $m \times m$ (that is, $X[m, m]$)

INV CALL Routine

The INV CALL routine calculates a matrix that is the inverse of the input matrix. The input matrix must be a square, nonsingular matrix.

The syntax of the INV CALL routine is

CALL INV (*X***, *Y***) ;**

where

X specifies an input matrix with dimensions $m \times m$ (that is, $X[m, m]$)

Y specifies an output matrix with dimensions $m \times m$ (that is, $Y[m, m]$)

MULT CALL Routine

The MULT CALL routine calculates the matrix product of two input matrices.

The syntax of the MULT CALL routine is

CALL MULT (*X***, *Y*, *Z*) ;**

where

X specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$)

Y specifies an input matrix with dimensions $n \times p$ (that is, $Y[n, p]$)

Z specifies an output matrix with dimensions $m \times p$ (that is, $Z[m, p]$)

The number of columns for the first input matrix must be the same as the number of rows for the second matrix. The calculated matrix is the last argument.

SUBTRACTMATRIX CALL Routine

The SUBTRACTMATRIX CALL routine performs an element-wide subtraction of two matrices or of a matrix and a scalar.

The syntax of the SUBTRACTMATRIX CALL routine is

CALL SUBTRACTMATRIX (*X*, *Y*, *Z***) ;**

where

X specifies a scalar or an input matrix with dimensions $m \times n$ (that is, $X[m, n]$)

Y specifies a scalar or an input matrix with dimensions $m \times n$ (that is, $Y[m, n]$)

Z specifies an output matrix with dimensions $m \times n$ (that is, $Z[m, n]$)

such that

$$Z = X - Y$$

TRANSPOSE CALL Routine

The TRANSPOSE CALL routine returns the transpose of a matrix.

The syntax of the TRANSPOSE CALL routine is

CALL TRANSPOSE (*X*, *Y***) ;**

where

X specifies an input matrix with dimensions $m \times n$ (that is, $X[m, n]$)

Y specifies an output matrix with dimensions $n \times m$ (that is, $Y[n, m]$)

ZEROMATRIX CALL Routine

The ZEROMATRIX CALL routine replaces all of the element values of the numeric input matrix with 0. You can use the ZEROMATRIX CALL routine with multidimensional numeric arrays.

The syntax of the ZEROMATRIX CALL routine is

CALL ZEROMATRIX (*X***) ;**

where

X specifies a numeric input matrix.

Create Design Matrix

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcdes.sas.)

PROC MCMC does not support a CLASS statement; therefore you need to construct the right design matrix (with dummy or indicator variables) prior to calling PROC MCMC. The best tool to use is the TRANSREG procedure (see Chapter 126, "The TRANSREG Procedure"). This procedure offers both indicator and effects coding methods. You can specify any categorical variables in the CLASS expansion, and use the ZERO= option to select a reference category. You can also specify any other data set variables (predictors, the responses, and so on) to the output data set in the ID statement.

For example, the following statements create a data set that contains two categorical variables (City and G), and two continuous variables (x and resp):

```
title 'Create Design Matrix';
data categorical;
   input City$ G$ x resp @@;
   datalines;
Chicago F 69.0 112.5  Chicago F 56.5  84.0
Chicago M 65.3  98.0  Chicago M 59.8  84.5
NewYork M 62.8 102.5  NewYork M 63.5 102.5
NewYork F 57.3  83.0  NewYork M 57.5  85.0
;
```

Suppose you are interested in creating a design matrix that uses dummy variable coding for the categorical variables City, G and their interaction City * G. You can use the following PROC TRANSREG statements:

```
proc transreg data=categorical design;
   model class(city g city*g / zero=last);
   id x resp;
   output out=input_mcmc(drop=_: Int:);
run;
```

The DESIGN option specifies that the primary goal is to code the design matrix. The MODEL statement indicates the variable of interest. The CLASS option in the MODEL statement expands the variables of interest to a list of "dummy" variables. The ZERO=LAST option sets the reference level. The ID statement includes x and resp in the OUT= data set. And the OUTPUT statement creates a new data set Input_MCMC that stores the design matrix and original variables from the original data set.

A quick call of the PRINT procedure shows the output from the PROC TRANSREG call:

```
proc print data=input_mcmc;
run;
```

Figure 80.15 prints the design matrix that is generated by PROC TRANSREG. The Input_mcmc data set contains all the variables from the original Categorical data set, in addition to corresponding dummy variables (CityChicago, GF, and CityChicagoGF) for the categorical variables.

Figure 80.15 Design Matrix Generated by PROC TRANSREG

Create Design Matrix

Obs	CityChicago	GF	CityChicagoGF	City	G	x	resp
1	1	1	1	Chicago	F	69.0	112.5
2	1	1	1	Chicago	F	56.5	84.0
3	1	0	0	Chicago	M	65.3	98.0
4	1	0	0	Chicago	M	59.8	84.5
5	0	0	0	NewYork	M	62.8	102.5
6	0	0	0	NewYork	M	63.5	102.5
7	0	1	0	NewYork	F	57.3	83.0
8	0	0	0	NewYork	M	57.5	85.0

You can now proceed to call PROC MCMC using this input data set Input_mcmc and the corresponding dummy variables.

PROC TRANSREG automatically creates a macro variable, &_TRGIND, which contains a list of variable names that it creates. The `%put &_trgind;` statement prints the following:

```
CityChicago GF CityChicagoGF
```

The macro variable &_TRGIND can come handy if you want to build a regression model; you can refer to &_TRGIND in the following way:

```
proc mcmc data=input_mcmc;
   array data[5] 1 &_trgind x;
   array beta[5] beta0-beta4;
   ...;
   call mult(beta, data, mu);
   ...;
```

The first ARRAY statement defines a one-dimensional array of length 5, and it takes on five values: a constant 1 and variables CityChicago, GF, CityChicagoGF, and x. The second ARRAY statement defines an array of beta, which are the model parameters. Later in the program, you can use the CALL MULT function to calculate the regression mean and store the value in the symbol mu.

Modeling Joint Likelihood

PROC MCMC assumes that the input observations are independent and that the joint log likelihood is the sum of individual log-likelihood functions. You specify the log likelihood of one observation in the MODEL statement. PROC MCMC evaluates that function for each observation in the data set and cumulatively sums them up. If observations are not independent of each other, this summation produces the incorrect log likelihood.

There are two ways to model dependent data. You can either use the DATA step LAG function or use the PROC option JOINTMODEL. The LAG function returns values of a variable from a queue. As PROC MCMC steps through the data set, the LAG function queues each data set variable, and you have access to the current value as well as to all previous values of any variable. If the log likelihood for observation x_i depends

only on observations 1 to i in the data set, you can use this SAS function to construct the log-likelihood function for each observation. Note that the LAG function enables you to access observations from different rows, but the log-likelihood function in the MODEL statement must be generic enough that it applies to all observations. See "Example 80.14: Time Independent Cox Model" on page 6437 and "Example 80.15: Time Dependent Cox Model" on page 6444 for how to use this LAG function.

A second option is to create arrays, store all relevant variables in the arrays, and construct the joint log likelihood for the entire data set instead of for each observation. Following is a simple example that illustrates the usage of this option. For a more realistic example that models dependent data, see "Example 80.14: Time Independent Cox Model" on page 6437 and "Example 80.15: Time Dependent Cox Model" on page 6444.

```
/* allocate the sample size. */
data exi;
   call streaminit(17);
   do ind = 1 to 100;
      y = rand("normal", 2.3, 1);
      output;
   end;
run;
```

The log-likelihood function for each observation is as follows:

$$\log(f(y_i|\mu,\sigma)) = \log(\phi(y_i;\mu,\text{var}=\sigma^2))$$

The joint log-likelihood function is as follows:

$$\log(f(\mathbf{y}|\mu,\sigma)) = \sum_i \log(\phi(y_i;\mu,\text{var}=\sigma^2))$$

The following statements fit a simple model with an unknown mean (mu) in PROC MCMC, with the variance in the likelihood assumed known. The MODEL statement indicates a normal likelihood for each observation y.

```
proc mcmc data=exi seed=7 outpost=p1;
   parm mu;
   prior mu ~ normal(0, sd=10);
   model y ~ normal(mu, sd=1);
run;
```

The following statements show how you can specify the log-likelihood function for the entire data set:

```
data a;
run;

proc mcmc data=a seed=7 outpost=p2 jointmodel;
   array data[1] / nosymbols;
   begincnst;
      rc = read_array("exi", data, "y");
      n = dim(data, 1);
   endcnst;

   parm mu;
   prior mu ~ normal(0, sd=10);
   ll = 0;
```

```
   do i = 1 to n;
      ll = ll + lpdfnorm(data[i], mu, 1);
   end;
   model general(ll);
run;
```

The JOINTMODEL option indicates that the function used in the MODEL statement calculates the log likelihood for the entire data set, rather than just for one observation. Given this option, PROC MCMC no longer steps through the input data during the simulation. Consequently, you can no longer use any data set variables to construct the log-likelihood function. Instead, you store the data set in arrays and use arrays instead of data set variables to calculate the log likelihood.

The ARRAY statement allocates a temporary array (data). The READ_ARRAY function selects the y variable from the exi data set and stores it in the data array. See the section "READ_ARRAY Function" on page 6235. In the programming statements, you use a DO loop to construct the joint log likelihood. The expression ll in the GENERAL function now takes the value of the joint log likelihood for all data.

You can run the following statements to see that two PROC MCMC runs produce identical results.

```
proc compare data=p1 compare=p2;
   var mu;
run;
```

Access Lag and Lead Variables

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmclag.sas.)

There are two types of random variables in PROC MCMC that are indexed: the response (MODEL statement) is indexed by observations, and the random effect (RANDOM statement) is indexed by the SUBJECT= option variable. As the procedure steps through the input data set, the response or the random-effects symbols are filled with values of the current observation or the random-effects parameters in the current subject. Often you might want to access lag or lead variables across an index. For example, the likelihood function for y_i can depend on y_{i-1} in an autoregressive model, or the prior distribution for μ_j can depend on μ_k, where $k \neq j$, in a dynamic linear model. In these situations, you can use the following rules to construct symbols to access values from other observations or subjects:

rv.*Li*: the *i*th lag of the variable rv. This looks back to the past.

rv.*Ni*: the *i*th lead of the variable rv. This looks forward to the future.

The construction is allowed for random variables that are associated with an index, either a response variable or a random-effects variable. You concatenate the variable name, a dot, either the letter *L* (for "lag") or the letter *N* (for "next"), and a lag or a lead number. PROC MCMC resolves these variables according to the indices that are associated with the random variable, with respect to the current observation.

For example, the following RANDOM statement specifies a first-order Markov dependence in the random effect mu that is indexed by the subject variable time:

```
random mu ~ normal(mu.l1,sd=1) subject=time;
```

This corresponds to the prior

$$\mu_t \sim \text{normal}(\mu_{t-1}, \text{sd} = 1)$$

At each observation, PROC MCMC fills in the symbol mu with the random-effects parameter μ_t that belongs to the current cluster t. To fill in the symbol mu.l1, the procedure looks back and finds a lag-1 random-effects parameter, μ_{t-1}, from the last cluster *t-1*. As the procedure moves forward in the input data set, these two symbols are constantly updated, as appropriate.

When the index is out of range, such as t–1 when t is 1, PROC MCMC fills in the missing state from the ICOND= option in either the MODEL or RANDOM statement. The following example illustrates how PROC MCMC fills in the values of these lag and lead variables as it steps through the data set.

Assume that the random effect mu has five levels, indexed by sub $= \{\text{a}, \text{b}, \text{c}, \text{d}, \text{e}\}$. The model contains two lag variables and one lead variable (mu.l1, mu.l2, and mu.n2):

```
mn = (mu.l1 + mu.l2 + mu.n2) / 3
random mu ~ normal(mn, sd=1) subject=time icond=(alpha beta gamma kappa);
```

In this setup, instead of a list of five random-effects parameters that the variable mu can be assigned values to, there is now a list of nine variables for the variables mu, mu.l1, mu.l2, and mu.n2. The list lines up in the following order:

$$\text{alpha}, \text{beta}, \mu_\text{a}, \mu_\text{b}, \mu_\text{c}, \mu_\text{d}, \mu_\text{e}, \text{gamma}, \text{kappa}$$

PROC MCMC finds relevant symbol values according to this list, as the procedure steps through different subject cluster. The process is illustrated in Table 80.46.

Table 80.46 Processing Lag and Lead Variables

sub	mu	mu.l2	mu.l1	mu.n2
a	μ_a	alpha	beta	μ_c
b	μ_b	beta	μ_a	μ_d
c	μ_c	μ_a	μ_b	μ_e
d	μ_d	μ_b	μ_c	gamma
e	μ_e	μ_c	μ_d	kappa

For observations in cluster a, PROC MCMC sets the random-effects variable mu to μ_a, looks back two lags and fills in mu.l2 with alpha, looks back one lag and fills in mu.l1 with beta, and looks forward two leads and fills in mu.n2 with μ_c. As the procedure moves to observations in cluster b, mu becomes μ_b, mu.l2 becomes beta, mu.l1 becomes μ_a, and mu.n2 becomes μ_d. For observations in the last cluster, cluster e, mu becomes μ_e, mu.l2 becomes μ_c, mu.l1 becomes μ_d, and mu.n2 is filled with kappa.

The following example fits a simple first-order dynamic linear model, in which the data set contains a time index and the response variable y:

```
data dlm;
   input time y;
   datalines;
1   1.353412529
2   4.840739953
3   1.604892523
4   6.8947921
5   3.509644288
6   4.020173553
7   3.842884451
8   4.49057276
9   2.204570502
10   4.007351323
11   2.005515044
12   2.781756057
;
```

You can fit the following model to the data:

$$
\begin{aligned}
Y_t &\sim \text{normal}(\mu_t, \text{var}=\sigma_y^2) \\
\mu_t &\sim \text{normal}(\mu_{t-1}, \text{var}=\sigma_\mu^2) \\
\mu_0 &= \alpha \\
\alpha &\sim \text{normal}(0, \text{var}=10) \\
\sigma_y^2, \sigma_\mu^2 &\sim \text{igamma}(\text{shape}=3, \text{scale}=2)
\end{aligned}
$$

The following PROC MCMC statements estimate parameters from this dynamic linear model:

```
proc mcmc data=dlm outpost=dlmO nmc=20000 seed=23;
   ods select PostSumInt;
   parms alpha 0;
   parms var_y 1 var_mu 1;
   prior alpha ~ n(0, sd=10);
   prior var_y var_mu ~ igamma(shape=3, scale=2);
   random mu ~ n(mu.l1,var=var_mu) s=time icond=(alpha) monitor=(mu);
   model y~n(mu, var=var_y);
run;
```

The key component is the mu.l1 specification in the RANDOM statement, where the prior for mu depends on its lag-1 value. The ICOND= option specifies the initial condition of mu and assigns it to be alpha, which is a parameter in the model.

Figure 80.16 lists the estimated posterior statistics for the parameters.

Figure 80.16 Posterior Summary Statistics of the Dynamic Linear Model

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
alpha	20000	2.6498	1.2924	-0.0555	5.0366
var_y	20000	1.7400	0.8337	0.5651	3.3498
var_mu	20000	0.8299	0.5720	0.2109	1.9582
mu_1	20000	2.6899	0.9253	0.8380	4.5390
mu_2	20000	3.4175	0.7622	1.9336	4.9541
mu_3	20000	3.3820	0.7431	1.8543	4.7950
mu_4	20000	4.3364	0.8297	2.7406	6.0198
mu_5	20000	3.9308	0.7194	2.4967	5.3248
mu_6	20000	3.8642	0.7348	2.4117	5.3832
mu_7	20000	3.7716	0.7399	2.3019	5.1525
mu_8	20000	3.6754	0.7316	2.1683	5.0342
mu_9	20000	3.1796	0.7209	1.7485	4.6257
mu_10	20000	3.2237	0.7355	1.7802	4.6253
mu_11	20000	2.8074	0.7839	1.2985	4.3701
mu_12	20000	2.8006	0.8758	1.1286	4.5473

Compartment Models

Pharmacokinetics (PK) is a branch of medicine that models the movement of a drug through the body (Gabrielsson and Weiner 2006). PK is sometimes referred to as the study of what the body does to a drug. Compartment models are basic building blocks of PK models. In a study, a body is divided in to several *compartments*, groups of organs or tissues that are kinetically homogeneous. The main interest of PK is to model how a drug moves through these compartments—for example, to estimate the amount of a drug and further concentrations of the drug that are present in a compartment at any given time. The concentrations of the drug, over time, are typically modeled via a set of differential equations that depend on a variety of variables, such as the amount of drug given, elimination rates, transfer rates, and so on. The sets of differential equations can also depend on how the drug is administered to the body, distributed through the body, and eliminated from the body. These models are known as compartment models, which are divided based on the number of compartments. Not all analytical solutions to multiple-compartment differential equations models are known, but Abuhelwa, Foster, and Upton (2015) and Fisher and Shafer (2007) provide closed-form solutions for those that correspond to the one-, two-, and three- compartments. These are the models that are handled by the CMPTMODEL statement.

Routes of Drug Administration

There are three types of drug administration methods: intravenous bolus, intravenous infusion, and extravascular dose administrations. A bolus medication typically has no or a very short time lag for the drug to enter a compartment. Intravenous infusions are given over a period of time, and the drug enters into the body at a constant rate. In infusion, the two quantities, *rate* and *duration* of the infusion, are of interest and are used in calculating the amount of drug present in the body. The rate and the duration of the infusion are related, so

knowing one determines the other. For compartment models that have infusion type of administration, it is sufficient to provide either the rate or the duration information.

One-, Two-, and Three-Compartment Models for Intravenous Administration

This section lists a number of assumptions that are made on the set of compartment models that the CMPTMODEL statement supports. The one-, two-, and three-compartment models all have a central compartment and can have one or more peripheral compartments that are linked only to the central compartment but not to each other. After a drug is administered to an administration site, it is distributed to the central compartment and then to other peripheral compartments. The rates at which the drug moves from the central compartment to and from the other peripheral compartments are characterized by transfer rate constants. The rates at which the drug is eliminated from the central or the peripheral compartments are characterized by elimination rate constants.

Schematic representation of the one-, two-, and three- compartment models are given in Figure 80.17, Figure 80.18, and Figure 80.19, respectively. In each figure, Dose is the dosage of the drug that is given intravenously, and k_{10} represents the rate at which the drug is eliminated from compartment 1. In Figure 80.18 and Figure 80.19, k_{12} represents the transfer rate constant from compartment 1 to compartment 2 and k_{21} represents the transfer rate constant from compartment 2 to compartment 1. Similarly, in Figure 80.19, k_{13} represents the transfer rate constant from compartment 1 to compartment 3 and k_{31} represents the transfer rate constant from compartment 3 to compartment 1. In these scenarios, compartment 1 is the central compartment and compartments 2 and 3 are the peripheral compartments. In compartment models, a drug can be administered only to and eliminated only from the central compartment.

Figure 80.17 One-Compartment Model

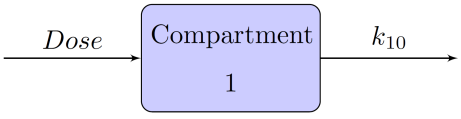


Figure 80.18 Two-Compartment Model

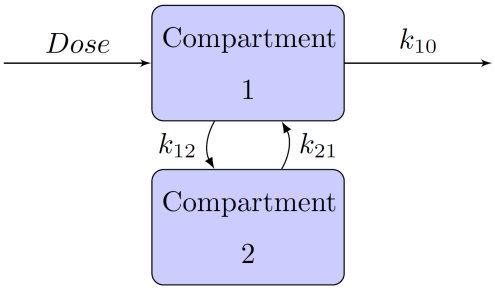
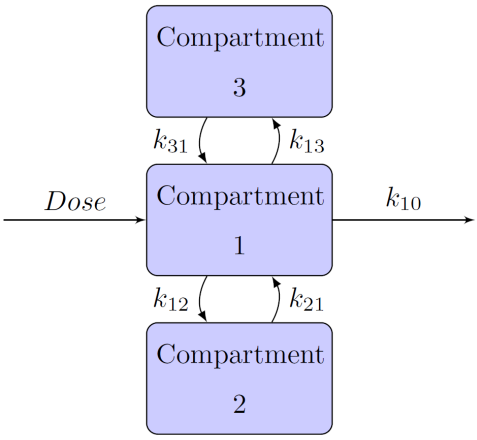


Figure 80.19 Three-Compartment Model



One-, Two-, and Three-Compartment Models for Extravascular Administration

Extravascular administrations, such as oral administration, are different from bolus and infusion in the sense that there is an absorption phase before the drug enters the central compartment. Schematic representation of the one-, two-, and three- compartment models when the drug is administered using an oral method are given in Figure 80.20, Figure 80.21, and Figure 80.22, respectively. In these models, compartment 0 is also called the *depot* compartment.

Figure 80.20 One-Compartment Model with Absorption Phase

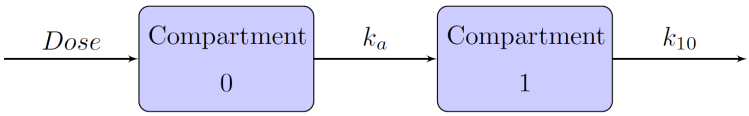


Figure 80.21 Two-Compartment Model with Absorption Phase

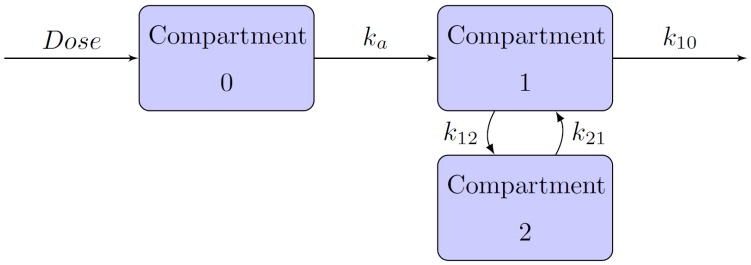
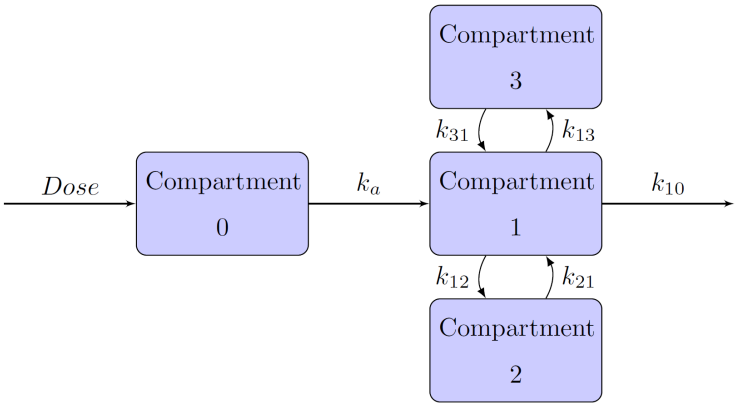


Figure 80.22 Three-Compartment Model with Absorption Phase



Compartment Models Specification

The CMPTMODEL statement in PROC MCMC enables you to fit one-, two-, and three-compartment models that use bolus, infusion, or oral types of drug administration. The CMPTMODEL statement computes the predicted concentrations of the drug for all compartments at each time point.

The CMPTMODEL statement supports a large number of options, which are grouped into three types: required, conditionally-required, and optional options. You use the required options to specify the type of compartment model that you want to fit (the NCOMPS=, ADMTYPE=, and PARMTYPE= options) and to provide the input time variable (in the TIME= option) and the outcome variable (in the PCONC= option) that is the predicted concentration in the first (central) compartment at each time point.

The statement supports the following nine types of compartment models:

1. one-compartment model with bolus dose administration
2. one-compartment model with infusion type of dose administration
3. one-compartment model with oral dose administration
4. two-compartment model with bolus dose administration
5. two-compartment model with infusion type of dose administration
6. two-compartment model with oral dose administration
7. three-compartment model with bolus dose administration
8. three-compartment model with infusion type of dose administration
9. three-compartment model with oral dose administration.

Each of these nine models can be parameterized in two ways: in terms of elimination and transfer rate constants of each compartment, or in terms of clearance and volume parameters of each compartment. The relationship between rate constants and the clearance and volumes of each compartment are:

$$K_{10} = CL1/VOL1$$
$$K_{12} = CL2/VOL1$$
$$K_{13} = CL3/VOL1$$
$$K_{21} = CL2/VOL2$$
$$K_{31} = CL3/VOL3$$

where CL*n* and VOL*n* are the clearance and volume of the *n*th compartment for *n*=1,2,3. You can use the PARMTYPE= option in the CMPTMODEL statement to specify these parameterizations. In total, you can specify 18 types of compartment models by using combinations of the NCOMPS=, ADMTYPE=, and PARMTYPE= options in the CMPTMODEL statement.

The conditionally required options define the specifics needed in some of the compartment models. For example, if you specify a two-compartment model with a bolus type of drug administration in terms of rate constants by using NCOMPS=2, ADMTYPE=IVB, and PARMTYPE=1, then you must specify the K12=, K21=, and K10= options. Here is an example:

```
PROC MCMC data=TwoComp;

...other statements...

erate10= exp(beta1);
trate12= exp(beta2);
trate21= exp(beta3);

CMPTMODEL ncomps=2 admtype=ivb parmtype=1 pconc=pred time=time
          k10=erate10 k12=trate12 k21=trate21
          dose1=ddose scale1=vol/1000;
model conc ~ normal(pred,var=s2);

...other statements...

run;
```

The K12= and K21= options specify the transfer rate constants, and the K10= option specifies the elimination rate constant from the central compartment. In addition, you can specify dosage information (DOSE*n*=) or scaling information (SCALE*n*=) in the syntax. For more information, see the section "CMPTMODEL Statement" on page 6237.

Table 80.47, Table 80.48, and Table 80.49 list all conditionally required options and valid optional options for each of the 18 compartment models.

Table 80.47 One-Compartment Models

Model	Required	Conditionally Required	Optional
One compartment with bolus dose	NCOMPS=1 ADMTYPE=IVB PARMTYPE=1	K10=	DOSE1= SCALE1=
One compartment with bolus dose	NCOMPS=1 ADMTYPE=IVB PARMTYPE=2	CL1= VOL1=	DOSE1= SCALE1=
One compartment with infusion dose	NCOMPS=1 ADMTYPE=INF PARMTYPE=1	K10= RATE= (or) DURN=	DOSE1= SCALE1=
One compartment with infusion dose	NCOMPS=1 ADMTYPE=INF PARMTYPE=2	CL1= VOL1= RATE= (or) DURN=	DOSE1= SCALE1=
One compartment with oral dose	NCOMPS=1 ADMTYPE=ORAL PARMTYPE=1	K10= Ka=	DOSE1= SCALE1=
One compartment with oral dose	NCOMPS=1 ADMTYPE=ORAL PARMTYPE=2	CL1= VOL1= Ka=	DOSE1= SCALE1=

Table 80.48 Two-Compartment Models

Model	Required	Conditionally Required	Optional
Two compartments with bolus dose	NCOMPS=2 ADMTYPE=IVB PARMTYPE=1	K10= K12= K21=	DOSE1= DOSE2= SCALE1= SCALE2= K20= PCONC2=
Two compartments with bolus dose	NCOMPS=2 ADMTYPE=IVB PARMTYPE=2	CL1= VOL1= CL2= VOL2=	DOSE1= DOSE2= SCALE1= SCALE2= K20= PCONC2=
Two compartments with infusion dose	NCOMPS=2 ADMTYPE=INF PARMTYPE=1	K10= K12= K21= RATE= (or) DURN=	DOSE1= SCALE1=
Two compartments with infusion dose	NCOMPS=2 ADMTYPE=INF PARMTYPE=2	CL1= VOL1= CL2= VOL2= RATE= (or) DURN=	DOSE1= SCALE1=
Two compartments with oral dose	NCOMPS=2 ADMTYPE=ORAL PARMTYPE=1	K10= K12= K21= Ka=	DOSE1= DOSE2= SCALE1= SCALE2= K20= PCONC2=
Two compartments with oral dose	NCOMPS=2 ADMTYPE=ORAL PARMTYPE=2	CL1= VOL1= CL2= VOL2= Ka=	DOSE1= DOSE2= SCALE1= SCALE2= K20= PCONC2=

Table 80.49 Three-Compartment Models

Model	Required	Conditionally Required	Optional
Three compartments with bolus dose	NCOMPS=3 ADMTYPE=IVB PARMTYPE=1	K10= K12= K21= K13= K31=	DOSE1= DOSE2= DOSE3= SCALE1= SCALE2= SCALE3= K20= K30= PCONC2= PCONC3=
Three compartments with bolus dose	NCOMPS=3 ADMTYPE=IVB PARMTYPE=2	CL1= VOL1= CL2= VOL2= CL3= VOL3=	DOSE1= DOSE2= DOSE3= SCALE1= SCALE2= SCALE3= K20= K30= PCONC2= PCONC3=
Three compartments with infusion dose	NCOMPS=3 ADMTYPE=INF PARMTYPE=1	K10= K12= K21= K13= K31= RATE= (or) DURN=	DOSE1= SCALE1=
Three compartments with infusion dose	NCOMPS=3 ADMTYPE=INF PARMTYPE=2	CL1= VOL1= CL2= VOL2= CL3= VOL3= RATE= (or) DURN=	DOSE1= SCALE1=
Three compartments with oral dose	NCOMPS=3 ADMTYPE=ORAL PARMTYPE=1	K10= K12= K21= K13= K31= Ka=	DOSE1= DOSE2= DOSE3= SCALE1= SCALE2= SCALE3= K20= K30= PCONC2= PCONC3=
Three compartments with oral dose	NCOMPS=3 ADMTYPE=ORAL PARMTYPE=2	CL1= VOL1= CL2= VOL2= CL3= VOL3= Ka=	DOSE1= DOSE2= DOSE3= SCALE1= SCALE2= SCALE3= K20= K30= PCONC2= PCONC3=

One-Compartment Model with an Oral Dose Administration

Consider the example "Example 80.22: One-Compartment Model with Pharmacokinetic Data" on page 6489, which studies the dispersion of the theophylline drug through a living individual. In this example, Pinheiro and Bates (1995) considered an one-compartment model with an oral dose. The paper mentions the solution to the one-compartment model. Instead of using the explicit solution, you can use the CMPTMODEL statement to fit the same model as follows:

```
proc mcmc data=theoph nmc=10000 seed=27 outpost=theoph0 diag=none;
   array b[2];
   array cov[2,2];
   array S[2,2] (1 0 0 1);
```

```
    parms beta1=-3.22 beta2=0.47 beta3=-2.45;
    parms cov {0.03 0 0 0.4};
    parms s2y;
    prior beta: ~ normal(0, sd=100);
    prior cov ~ iwish(2, S);
    prior s2y ~ igamma(shape=3, scale=2);
    random b ~ mvn(muB, cov) subject=subject;

    cl = exp(beta1 + b1);
    ka = exp(beta2 + b2);
    ke = exp(beta3);
    vl = cl/ke;

    CMPTMODEL ncomps=1 admtype=oral time=time pconc=predConc
              parmtype=1 ka=ka k10=ke dose0=dose scale1=vl;

    model conc ~ normal(predConc,var=s2y);
run;
```

The NCOMPS=1 and ADMTYPE=ORAL options specify a one-compartment model with oral administration. Time is a data set variable that indicates time, and predConc is an outcome variable that contains predicted concentration. The PARMTYPE=1 option requests the compartment model using absorption and elimination rate constants, with KA=ka and K10=ke indicating the absorption and elimination rate constants, respectively. The dosage value for each patient in the depot compartment is specified in DOSE0=option, where dose is a data set variable. Lastly, SCALE1=vl scales the predicted concentrations by vl.

Multiple Doses

In PK field experiments or observational studies, a patient can often receive a drug multiple times, either continually or periodically over a period of time. The patient can also receive multiple types of drug in a study. For example, a patient might receive a bolus injection in the morning and an infusion drug at a constant rate in the evening. Calculation of predicted concentration in each compartment in the presence of these multiple doses or multiple types of dosing is different from single-dosing compartment models. For differential equations of a number of multiple-dosage scenarios and solutions to predicted concentration for the central compartment, see Gabrielsson and Weiner (2006).

The CMPTMODEL statement handles multiple doses in one-, two-, or three-compartment models and computes predicted concentrations in the central compartment. The syntax specification does not change from single-dose models, and the structure and content of the input data set are understood by the statement to fit various types of multiple-dosage models.

Within the biopharmaceutical industry, data in multiple-dosage studies are often structured by following a convention that has been popularized by NONMEM software (Beal et al. 2011). This convention is used to name variables and specify variable values; see Owen and Fiedler-Kelly (2014). If your data are available in a SAS data set that follows this convention, the data set must be converted to a SAS data set that is suitable to be analyzed using PROC MCMC. You can use the autocall macro %PKCONVRT for this purpose.

The %PKCONVRT macro takes two arguments: an input data set (specified by the first argument) and an output data set (specified by the second argument). The organization of the input data set should follow the convention. The output data set can serve as the DATA= data set for PROC MCMC to fit single-dose or multiple-dose models. Here is a simple example:

```
data pk_ex;
input ID  TIME  AMT DV  EVID;
datalines;
1     0.00    60000        .       1
1     0.20    50000        .       1
1     0.25        0    1126.1      0
1     0.50        0     869.9      0
1     0.75        0     883.6      0
1     1.00        0    1244.0      0
1     1.50        0     995.2      0
2     0.00    70000        .       1
2     0.25        0    1126.1      0
2     0.50        0     869.9      0
2     0.75        0     883.6      0
2     1.00        0    1244.0      0

   .... more lines ...
;
run;

%pkconvrt(data=pk_ex, out=out_ex);
```

If you want to fit a two-compartment model for the preceding multiple dosage data, use the following MCMC program with the out_ex data set that is produced from the %PKCONVRT macro:

```
proc mcmc data=out_ex nmc=10000 seed=17071 outpost=post_out;
   array b[4] b1 b2 b3 b4;
   array b0[4] (0 0 0 0);
   array Sigma[4,4];
   array S[4,4] (1 0 0 0, 0 1 0 0, 0 0 1 0, 0 0 0 1);

   parms beta1=2 beta2=2 beta3=2 beta4=2 s2=0.6 Sigma;
   prior beta: ~n(0, sd=10000);
   prior s2 ~ igamma(shape=3, scale=2);
   prior Sigma ~ iwish(4, S);

   random b ~ mvn(b0, Sigma) subject=id;
   cl1 = exp(beta1+b1);
   vl1 = exp(beta2+b2);
   cl2 = exp(beta3+b3);
   vl2 = exp(beta4+b4);

   CMPTMODEL ncomps=2 admtype=ivb parmtype=2 pconc=pred time=time
      vol1=vl1 cl1=cl1 vol2=vl2 cl2=cl2 scale1=vl1;
   model conc ~ normal(pred,var=s2);
run;
```

The CMPTMODEL statement computes the predictions only for the central compartment for multiple dosage data. Therefore, options PCONC2=, PCONC3=, and PCONC0= are not valid in a multiple dose model. In addition, the K20=, K30=, DOSE2=, DOSE3=, SCALE2=, SCALE3=, SCALE0= options are ignored.

The %PKCONVRT autocall macro computes the elapsed time (from the time that a dose is administered to the time that a concentration is measured) in scenarios for both single and multiple continuous doses. The macro writes the computed time in the data set that is specified in the OUT= option. This data set also

contains all the dosage information for all the time points at which the concentrations are measured. When you specify this data set in the PROC MCMC statement and you specify a CMPTMODEL statement, the elapsed times and the dosage information from the data set are used directly in computing the predicted concentrations in the central compartment. As a result, the CMPTMODEL statement ignores any specified DOSE1= and TIME= options. Similarly, for absorption models (ADMTYPE = ORAL), the DOSE0= and TIME= options in the CMPTMODEL statement are ignored and overwritten with the elapsed times and the dosage information from the OUT= data set. For examples that use the %PKCONVRT autocall macro and the CMPTMODEL statement to fit various compartment models, see Kurada and Chen (2018).

In addition, the CMPTMODEL statement supports steady-state dosing scenarios (Owen and Fiedler-Kelly 2014) in one-, two-, and three-compartment models. The %PKCONVRT macro keeps track of the elapsed time and dosing information that are required in the CMPTMODEL statement to fit steady-state dose models. The %PKCONVRT autocall macro supports input data sets that have combinations of steady-state and regular dosing, single and multiple dosing. For example, your data set can contain dosing history information on patients who receive multiple regular bolus doses and others who receive a single steady-state dose. When you use the output data set from the %PKCONVRT macro as input to PROC MCMC, the CMPTMODEL statement properly interprets all these distinct scenarios and computes the predicted concentrations accordingly.

To use the %PKCONVRT autocall macro in conjunction with PROC MCMC for a Bayesian analysis of pharmacokinetic data, specify the DOSEDROP=TRUE option. This option deletes dosing records and keeps measurement records only. By default, the DOSEDROP= option is set to FALSE and the %PKCONVRT macro keeps all records, including dosing records, from the input data set. In these records, the response variable DV is filled with missing values. By default, PROC MCMC treats all missing response values as parameters, draws samples, and incorporates them in the posterior inference. Imputing missing concentrated values at dosing time will change the posterior distribution and lead to a different conclusion than an analysis that uses only observations from measurement records. The DOSEDROP=TRUE option ensures the deletion of the dosing records.

The %PKCONVRT macro creates new variables (such as the _CMPT_ variable) in the OUT= data set. According to SAS conventions, it is best to avoid having variables whose name begins with an underscore _ in the DATA = data set. Also programming variables in the MCMC program that start with _CMPT_ are reserved for the internal computations; hence, avoid programming variables names that begin with _CMPT_ .

CALL ODE and CALL QUAD Subroutines

The CALL ODE subroutine numerically solves a set of first-order ordinary differential equations (ODEs), including piecewise differential equations. The CALL QUAD subroutine calculates multidimensional integrand. You can use them as programming statements in PROC MCMC. These subroutines require that you define an objective function, for either a set of simultaneous differential equations or an integrand function, by using PROC FCMP (see the FCMP procedure in the *Base SAS Procedures Guide*) and call these subroutines in PROC MCMC.

CALL ODE

The CALL ODE subroutine performs numerical integration of first-order vector differential equations of the form $\frac{dy}{dt} = f(t, y(t))$ over the subinterval $t \in [t_i, t_f]$ with the initial values $y(t_i) = y_0$. The subroutine can also be used to solve piecewise differential equations.

You specify the CALL ODE subroutine in PROC MCMC by using the following syntax:

```
CALL ODE("DeqFun", Soln, Init, ti, tf, G1, G2, ... , Gk <, ode_opt>);
```

DeqFun: the name of the PROC FCMP subroutine of a set of simultaneous set of ordinary differential equations, $\frac{dy}{dt} = f(t, y(t))$

Soln: an argument that contains solutions. Soln can be a numeric variable or an array. If it is an array, then the size of the array determines the dimension of the problem. Otherwise, the dimension of the problem is one.

Init: initial values of the variable y. Init can be either a numeric variable or an array. The dimension must match that of Soln.

ti: initial time value of the subinterval *t*

tf: final time value of the subinterval *t*

Gi: input arguments in the DeqFun subroutine

ode_opt: subroutine options. ode_opt is a positional array of size 4, with the following definition:

- ode_opt[1]: convergence accuracy. The default value is 1E–8.
- ode_opt[2]: minimum allowable step size in the integration process. The default value is 1E–12.
- ode_opt[3]: maximum allowable step size in the integration process. The default value is the largest double-precision floating-point number.
- ode_opt[4]: initial step size to start the integration process. The default value is 1E–5.

To specify the DeqFun subroutine by using PROC FCMP, use the following statements:

```
proc fcmp outlib=sasuser.funcs.ODE;
   subroutine DeqFun(t,y[*],dy[*], A1,A2,...,Ak);
   outargs dy;
   dy[1] = -A1*y[1];
   dy[2] = A2*y[1]-Ak-1*y[2];
   ...
   dy[n] = t*y[n] + Ak;
   endsub;
run;
```

The OUTLIB= option specifies an output data set to which the compiled subroutine is written. The first three arguments of the DeqFun subroutine must be (1) the time variable t, (2) the with-respect-to variable y, which can be an array, and (3) the differential equation function variable dy, which can also be an array. The remaining optional input arguments are variables that are required in the construction of the differential equations. You must declare variable dy as the updated variable in the OUTARGS statement.

To include the DeqFun in your PROC MCMC program, you must specify the SAS data set that contains the compiler subroutine:

```
options cmplib=sasuser.funcs;
```

See "Example 80.22: One-Compartment Model with Pharmacokinetic Data" on page 6489 for an example of fitting a pharmacokinetic model by using the CALL ODE subroutine.

The CALL ODE subroutine uses the polyalgorithm of Byrne and Hindmarsh (1975) to provide a numerical solution for the set of ordinary differential equations. Note that you can model any n-order differential equation as a system of first-order differential equations.

Piecewise Differential Equations

Suppose you have a system of piecewise ODEs of the following form:

$$\frac{d\mathbf{y}}{dt} = \begin{cases} f_1(t, \mathbf{y}(t)) & t_1 \le t < t_2 \\ f_2(t, \mathbf{y}(t)) & t_2 \le t < t_3 \\ \vdots & \vdots \\ f_{m-1}(t, \mathbf{y}(t)) & t_{m-1} \le t < t_m \\ f_m(t, \mathbf{y}(t)) & t_m \le t \end{cases}$$

The system of equations is defined over m intervals ($t \in [t_i, t_f]$) with initial values $y(t_j) = y_(t_{j0})$ for $j = 1, 2, \ldots, m$. The variable $\boldsymbol{y}(t)$ can be a vector. The CALL ODE subroutine solves the system of piecewise ODEs in each semiclosed interval $[t_i, t_j)$ for $i < j = 1, 2, \ldots, m$.

The CALL ODE subroutine differentiates between a regular ODE problem and a piecewise ODE problem based on the dimension of the second input argument to the subroutine. That is the Soln argument, which stores the solutions to the ODEs. If Soln is either a numeric variable or a one-dimensional array, the CALL ODE subroutine treats the problem as a regular ODE problem; if Soln is a two-dimensional array, the CALL ODE subroutine treats it as a piecewise ODE problem and solves accordingly.

You specify the CALL ODE subroutine in PROC MCMC to solve piecewise ODEs by using the following syntax:

```
CALL ODE("DeqFun", Soln, ., ti, tf, G1, G2, ... , Gk,
         ode_grid, "InitFun", H1, H2, ..., Hl <, ode_opt>);
```

DeqFun: the name of the PROC FCMP subroutine of a system of ODEs

Soln: an argument that contains solutions. Soln must be an m (number of intervals) $\times$ n (number of equations) array. Upon completion, the CALL ODE subroutine fills in the last row of the Soln array (Soln$[m, 1], \ldots,$ Soln$[m, n]$) with the final solutions.

. (period): the third positional argument takes a missing value, or a period (.). In a regular ODE problem, the third argument is reserved for initial values, which are fixed per solution. In a piecewise problem, initial values can depend on solutions to previous set of ODEs, and they are set through a separate PROC FCMP subroutine (see description of InitFun).

ti: initial time value of the subinterval t

tf: final time value of the subinterval t

Gi: input arguments in the DeqFun subroutine

ode_grid: numerical array that stores the interval boundaries. You must use the keyword ode_grid for this array, and the array should be placed after all the Gi variables. Elements in ode_grid must be sorted in ascending order. The dimension of this array should be *m*, the number of rows in Soln.

InitFun: the name of the PROC FCMP subroutine that specifies the initial values of the ODEs at different intervals. The subroutine requires an output array argument that should be filled with the required initial values for each interval. The InitFun subroutine should come after the ode_grid variable.

Hi: input arguments in the InitFun subroutine

ode_opt: subroutine options

The following example statements construct a two-interval piecewise ODE in the DeqFun subroutine, set initial values in the InitFun subroutine, and call the general ODE solver in PROC MCMC:

```
proc fcmp outlib=sasuser.funcs.ODE;
   subroutine DeqFun(t,y[*],dy[*],t1,beta,alpha);
   outargs dy;
   if (t <= t1) then do;
       dy[1] = -y[1];
       dy[2] = beta * y[1] - y[2];
       dy[3] = - y[3];
   end; else do;
       dy[1] = 0;
       dy[2] = alpha * y[1] + y[2];
       dy[3] = alpha * y[3] * y[3];
   end;
   endsub;
run;

proc fcmp outlib=sasuser.funcs.ODE;
   subroutine InitFun(init[*,*],d,Soln[*,*]);
   outargs init;
   init[1,1] = d;
   init[1,2] = d;
   init[1,3] = d;
   init[2,1] = Soln[1,1];
   init[2,2] = Soln[1,2];
   init[2,3] = Soln[1,3];
   endsub;
run;

options cmplib=sasuser.funcs;

proc mcmc ...;
   array Soln[2, 3];
   array ode_grid[2];

   ode_grid[1]=0;
   ode_grid[2]=ti;
   call ode("DeqFun", Soln, ., 0, tf, ti, beta, alpha,
            ode_grid, "InitFun", dose, Soln);
   ...;
run;
```

The DeqFun subroutine specifies three differential equations in two intervals, $t \leq t_2$ and $t > t_2$. The InitFun subroutine specifies the initial values for the first set of equations as d (an input variable) and specifies the initial values for the second set of equations as solutions to the first set of equations. In the PROC MCMC statements, Soln is an array of size 2×3. The ode_grid array has two elements: the first element is the lower bound (0 in this case), and the second element is ti (for example, a data set variable). The DeqFun and InitFun subroutines are passed to the ODE solver. Three variables (ti, beta, and alpha) are input arguments to the DeqFun subroutine. Two variables (dose and Soln) are input arguments to the InitFun subroutine. The solution array, Soln, is passed to the InitFun subroutine and enables the ODE solver to use solutions to the first set of ODEs as the initial values for the second set. The CALL ODE subroutine fills the second row of the Soln array (Soln[2,1], Soln[2,2], and Soln[2,3]) with the final solution to the entire system of the ODEs upon completion.

CALL QUAD

The CALL QUAD subroutine is a numerical integrator that performs the integration in a twofold fashion based on the dimension of the problem. In a unidimensional problem, the subroutine uses the adaptive Romberg type of integration techniques; in a multidimensional problem, it uses the Laplace approximation. You specify the CALL QUAD subroutine syntax in PROC MCMC by using the following syntax:

```
CALL QUAD("IntFun",Res, LL, UL, G1, G2, ... , Gk <, quad_init> <, quad_opt>);
```

IntFun: the name of the PROC FCMP subroutine of an integrand function

Res: contains the approximated integral value

LL: lower limit of the integral. LL must be an array if the problem is multidimensional.

UL: upper limit of the integral. UL must be an array if the problem is multidimensional.

Gi: input arguments in the IntFun subroutine

quad_init: starting values that are used in optimizing the IntFun function in Laplace approximation. This optional array is applicable only in a multidimensional problem, and you must use the keyword quad_init for this array. By default, the optimization starts at the average values of LL and UL. When you use the variable quad_init, it must precede the integration option array quad_opt.

quad_opt: integration options. You must use the keyword quad_opt for this array. For a unidimensional integration problem, quad_opt must be a positional array of size 4:

- quad_opt[1]: relative accuracy. The default value is 1E–7.
- quad_opt[2]: the approximate location of a maximum of the integrand. The default value is the centered location between the LL and UL limits.
- quad_opt[3]: approximate estimate of any scale in the integrand along the independent variables. The default value is 1.
- quad_opt[4]: the number of refinements allowed in order to achieve the required accuracy

In a multidimensional integration problem, quad_opt is a positional array of size 2:

- quad_opt[1]: relative accuracy. The default value is 1E–7.

- quad_opt[2]: optimization technique:
  - 1: Quasi-Newton
  - 2: Newton-Raphson
  - 3: Newton-Raphson with ridging
  - 4: Nelder-Mead simplex method

To specify the subroutine IntFun in PROC FCMP, you can use the following example:

```
proc fcmp outlib=sasuser.funcs.QUAD;
   subroutine IntFun(y[*],fy,pi);
   outargs fy;
   fy = (exp(-(y[1]*y[1]+y[2]*y[2])/2))/(2*pi);
   endsub;
run;
```

The first argument to IntFun must be the with-respect-to vector in the integral. The variable y can be a scalar or an array, depending on your problem. The second argument must be the variable that contains the integrand value, which should be declared as the updated variable in the OUTARGS statement. The remaining input arguments are optional variables, if needed in constructing the integrand function.

To include the IntFun subroutine in your PROC MCMC program, you must specify the SAS data set that contains the compiler subroutine:

```
options cmplib=sasuser.funcs;

proc mcmc ...;
   array LL[2] (-100 -100);
   array UL[2] ( 100  100);

   call quad("IntFun", Res, LL, UL, pi);
   ...;
run;
```

In the PROC MCMC statements, LL and UL are arrays that contain the lower and upper limits of the integral. The variable pi is an input argument to the IntFun subroutine. The solution is stored in Res.

In case of one-dimensional integration, the CALL QUAD subroutine uses adaptive Romberg-type integration techniques. (See Rice 1973; Sikorsky 1982; Sikorsky and Stenger 1984; Stenger 1973b, a.) Many adaptive numerical integration methods (Ralston and Rabinowitz 1978) start at one end of the interval and proceed toward the other end, working on subintervals while locally maintaining a certain prescribed precision. This is not the case with the CALL QUAD subroutine. The CALL QUAD subroutine is an adaptive global-type integrator that produces a quick, rough estimate of the integration result and then refines the estimate until it achieves the prescribed accuracy. This gives the subroutine an advantage over Gauss-Hermite and Gauss-Laguerre quadratures (Ralston and Rabinowitz 1978; Squire 1987), particularly for infinite and semi-infinite intervals, because those methods perform only a single evaluation.

The Laplace approximation is used in the multidimensional problem. To approximate a k-dimensional integration of an integrand, $h(t)$, between the limits a and b, you can use the approximation formula

$$\int_a^b h(t)dt \cong h(c)\sqrt{\frac{(2\pi)^k}{|H(t)|_{t=c}}}$$

where $-\log[h(t)]$ assumes a minimum over $[a, b]$ at an interior critical point c, and $H(t)$ is the Hessian of $-\log[h(t)]$,

$$H(t) = \frac{\partial^2 \{-\log(h(t))\}}{\partial t \partial t'}$$

The minimum of the negative log integrand is found by using numerical optimization. By default, the CALL QUAD subroutine uses the quasi-Newton method. You can select other methods by using the quad_opt option.

Regenerating Diagnostics Plots

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcren.sas.)

By default, PROC MCMC generates three plots: the trace plot, the autocorrelation plot, and the kernel density plot. Unless ODS Graphics is enabled before calling the procedure, it is hard to generate the same graph afterwards. Directly using the **Stat.MCMC.Graphics.TraceAutocorrDensity** template is not feasible. The easiest way to regenerate the same graph is with the %TADPlot autocall macro. The %TADPlot macro requires you to specify an input data set (which usually is the output data set from a previous PROC MCMC call) and a list of variables that you want to plot.

For more information about enabling and disabling ODS Graphics, see the section "Enabling and Disabling ODS Graphics" on page 687 in Chapter 24, "Statistical Graphics Using ODS."

A simple regression example, with three parameters, is used here for illustrational purposes. For an explanation of the regression model and the data involved, see the section "Simple Linear Regression" on page 6201. The following statements fit a regression model:

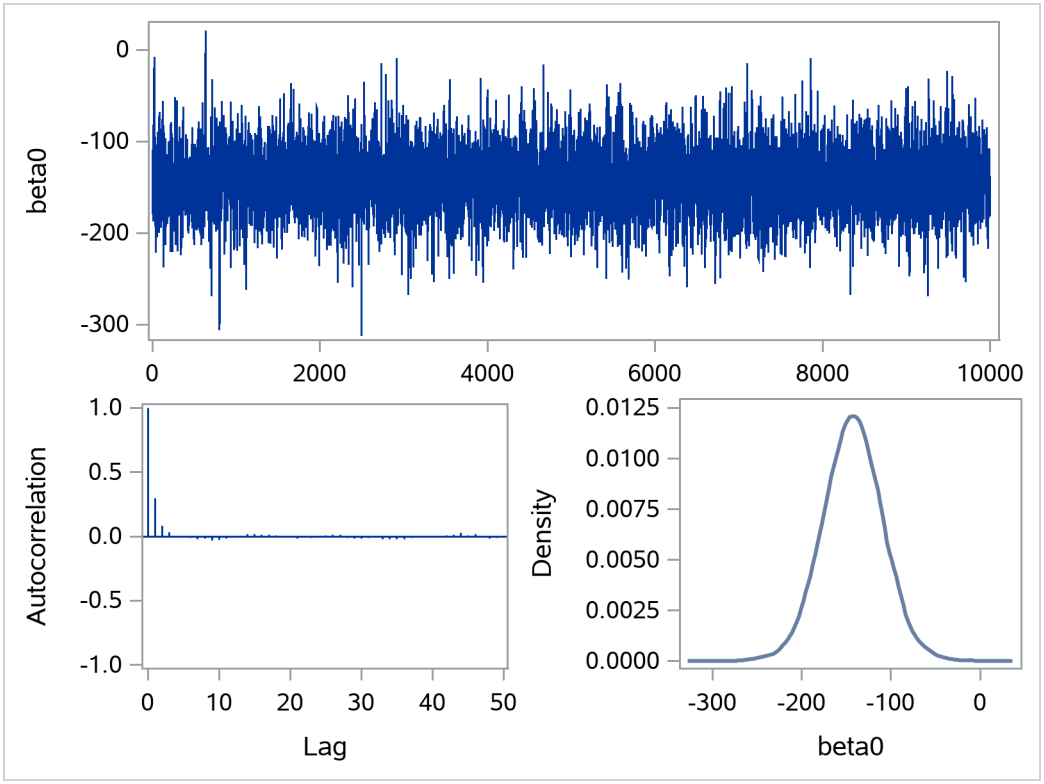
```
ods select none;
proc mcmc data=sashelp.class nmc=50000 thin=5 outpost=classout seed=246810;
   parms beta0 0 beta1 0;
   parms sigma2 1;
   prior beta0 beta1 ~ normal(0, var = 1e6);
   prior sigma2 ~ igamma(3/10, scale = 10/3);
   mu = beta0 + beta1*height;
   model weight ~ normal(mu, var = sigma2);
run;
ods select all;
```

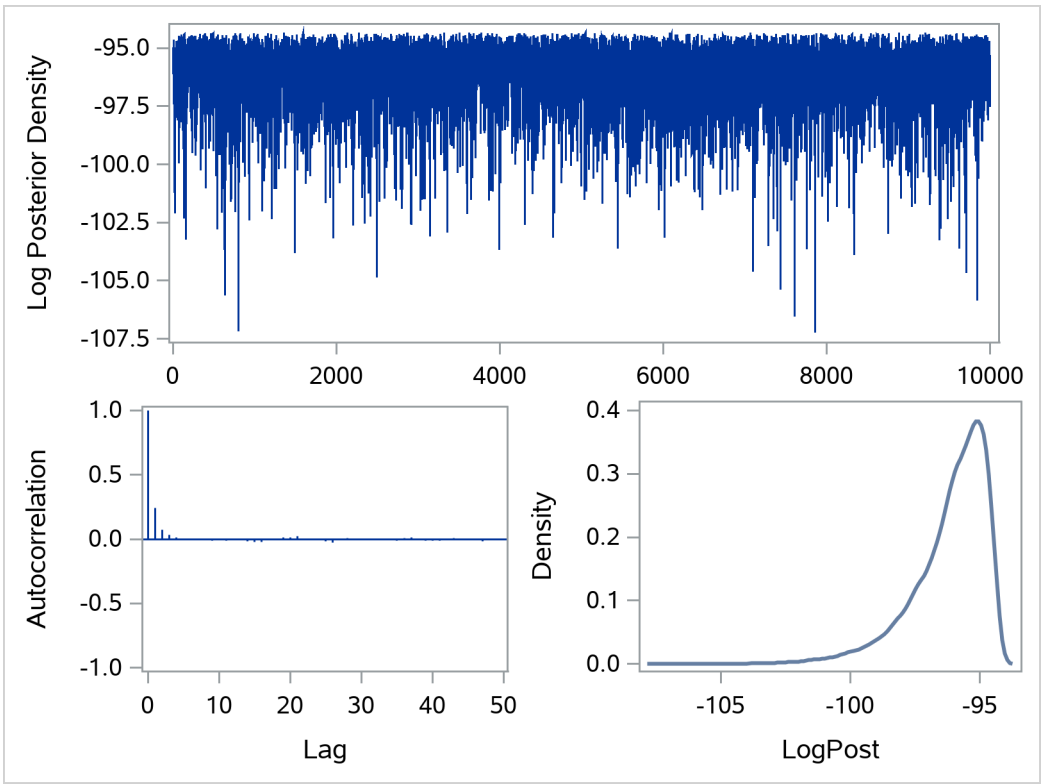
The output data set Classout contains posterior draws for beta0, beta1, and sigma2. It also stores the log of the prior density (LogPrior), log of the likelihood (LogLike), and the log of the posterior density (LogPost). If you want to examine the beta0 and LogPost variable, you can use the following statements to generate the graphs:

```
ods graphics on;
%tadplot(data=classout, var=beta0 logpost);
ods graphics off;
```

Figure 80.23 displays the regenerated diagnostics plots for variables beta0 and Logpost from the data set Classout.

Figure 80.23 Regenerated Diagnostics Plots for beta0 and Logpost





Caterpillar Plot

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmccat.sas.)

The caterpillar plot is a side-by-side bar plot of 95% intervals for multiple parameters. Typically, it is used to visualize and compare random-effects parameters, which can come in large numbers in certain models. You can use the %CATER autocall macro to create a caterpillar plot. The %CATER macro requires you specify an input data set and a list of variables that you want to plot.

A random-effects model that has 21 random-effects parameters is used here for illustrational purpose. For an explanation of the random-effects model and the data involved, see "Example 80.7: Logistic Regression Random-Effects Model" on page 6401. The following statements generate a SAS data set and fit the model:

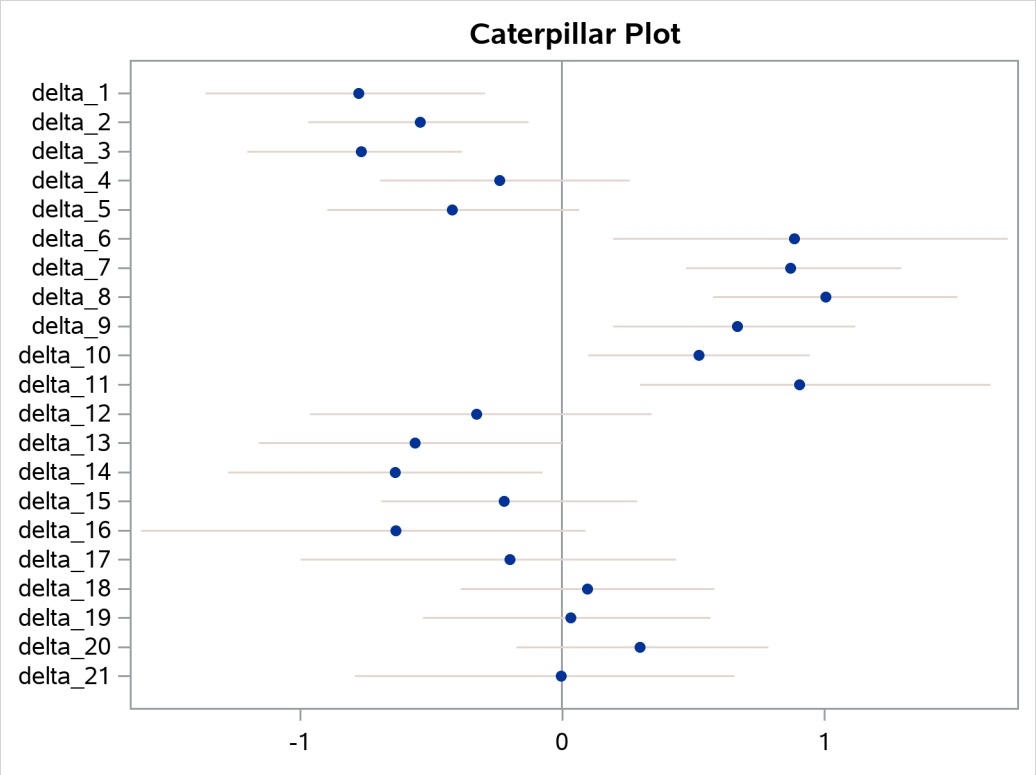
```
title 'Create a Caterpillar Plot';

data seeds;
   input r n seed extract @@;
   ind = _N_;
   datalines;
10  39  0  0    23  62  0  0    23  81  0  0    26  51  0  0
17  39  0  0     5   6  0  1    53  74  0  1    55  72  0  1
32  51  0  1    46  79  0  1    10  13  0  1     8  16  1  0
10  30  1  0     8  28  1  0    23  45  1  0     0   4  1  0
 3  12  1  1    22  41  1  1    15  30  1  1    32  51  1  1
 3   7  1  1
;

ods select none;
proc mcmc data=seeds outpost=postout seed=332786 nmc=20000;
   parms beta0 0 beta1 0 beta2 0 beta3 0 s2 1;
   prior s2 ~ igamma(0.01, s=0.01);
   prior beta: ~ general(0);
   w = beta0 + beta1*seed + beta2*extract + beta3*seed*extract;
   random delta ~ normal(w, var=s2) subject=ind;
   pi = logistic(delta);
   model r ~ binomial(n = n, p = pi);
run;
ods select all;
```

The output data set Postout contains posterior draws for all 21 random-effects parameters, delta_1 ⋯ delta_21. You can use the following statements to generate a caterpillar plot for the 21 parameters:

```
%CATER(data=postout, var=delta:);
```

Figure 80.24 is a caterpillar plot of the random-effects parameters delta_1–delta_21.

Figure 80.24 Caterpillar Plot of the Random-Effects Parameters



If you want to change the display of the caterpillar plot, such as using a different line pattern, color, or size of the markers, you need to first modify the **`Stat.MCMC.Graphics.Caterpillar`** template and then call the %CATER macro again.

You can use the following statements to view the source of the **`Stat.MCMC.Graphics.Caterpillar`** template:

```
proc template;
   path sashelp.tmplmst;
   source Stat.MCMC.Graphics.Caterpillar;
run;
```

Figure 80.25 lists the source statements of the template that is used to generate the template for the caterpillar plot.

Figure 80.25 Source Statements for Stat.MCMC.Graphics.Caterpillar Template

```
define statgraph Stat.MCMC.Graphics.Caterpillar;
   dynamic _OverallMean _VarName _VarMean _XLower _XUpper _byline_ _bytitle_
      _byfootnote_;
   begingraph;
      entrytitle "Caterpillar Plot";
      layout overlay / yaxisopts=(offsetmin=0.05 offsetmax=0.05 display=(line
         ticks tickvalues)) xaxisopts=(display=(line ticks tickvalues));
         referenceline x=_OVERALLMEAN / lineattrs=(color=
            GraphReference:ContrastColor);
         HighLowPlot y=_VARNAME high=_XUPPER low=_XLOWER / lineattrs=
            GRAPHCONFIDENCE;
         scatterplot y=_VARNAME x=_VARMEAN / markerattrs=(size=5 symbol=
            circlefilled);
      endlayout;
      if (_BYTITLE_)
         entrytitle _BYLINE_ / textattrs=GRAPHVALUETEXT;
      else
         if (_BYFOOTNOTE_)
            entryfootnote halign=left _BYLINE_;
         endif;
      endif;
   endgraph;
end;
```

You can use the TEMPLATE procedure (see Chapter 24, "Statistical Graphics Using ODS") to modify the graph template. Subsequent calls to the %CATER macro will use the modified template to make the graph.

Autocall Macros for Postprocessing

Although PROC MCMC provides a number of convergence diagnostic tests and posterior summary statistics, PROC MCMC performs the calculations only if you specify the options in advance. If you wish to analyze the posterior draws of unmonitored parameters or functions of the parameters that are calculated in later DATA step calls, you can use the autocall macros in Table 80.50.

Table 80.50 Postprocessing Autocall Macros

Macro	Description
%ESS	Effective sample sizes
%GEWEKE*	Geweke diagnostic
%HEIDEL*	Heidelberger-Welch diagnostic
%MCSE	Monte Carlo standard errors
%RAFTERY	Raftery diagnostic
%POSTACF	Autocorrelation
%POSTCOR	Correlation matrix
%POSTCOV	Covariance matrix
%POSTINT	Equal-tail and HPD intervals
%POSTSUM	Summary statistics
%SUMINT	Mean, standard deviation, and HPD interval

*The %GEWEKE and %HEIDEL macros use a different optimization routine than that used in PROC MCMC. As a result, there might be numerical differences in some cases, especially when the sample size is small.

Table 80.51 lists options that are shared by all postprocessing autocall macros. See Table 80.52 for macro-specific options.

Table 80.51 Shared Options

Option	Description
DATA=*SAS-data-set*	Input data set that contains posterior samples
VAR=*variable-list*	Specifies the variables on which you want to carry out the calculation.
PRINT=YES | NO	Displays the results. The default is YES.
OUT=*SAS-data-set*	Specifies a name for the output SAS data set to contain the results.

Suppose that the data set that contains posterior samples is called post and that the variables of interest are defined in the macro variable &PARMS. The following statements call the %ESS macro and calculates the effective sample sizes for each variable:

```
%let parms = alpha beta u_1-u_17;
%ESS(data=post, var=&parms);
```

By default, the ESS estimates are displayed. You can choose not to display the result and save the output to a data set with the following statement:

```
%ESS(data=post, var=&parms, print=NO, out=eout);
```

Some of the macros can take additional options, which are listed in Table 80.52.

Table 80.52 Macro-Specific Options

Macro	Option	Description
%ESS	AUTOCORLAG=*numeric*	Specifies the maximum number of autocorrelation lags used in computing the ESS estimates. By default, AUTOCORLAG=MIN(500, NOBS/4), where NOBS is the sample size of the input data set.
	HIST=YES|NO	Displays a histogram of all ESS estimates. The default is NO.
%HEIDEL	SALPHA=*numeric*	Specifies the α level for the stationarity test. By default, SALPHA=0.05.
	HALPHA=*numeric*	Specifies the α level for the halfwidth test. By default, HALPHA=0.05.
	EPS=*numeric*	Specifies a small positive number ϵ such that if the halfwidth is less than ϵ times the sample mean of the remaining iterations, the halfwidth test is passed. By default, EPS=0.1.
%GEWEKE	FRAC1=*numeric*	Specifies the earlier portion of the Markov chain used in the test. By default, FRAC1=0.1.
	FRAC2=*numeric*	Specifies the latter portion of the Markov chain used in the test. By default, FRAC2=0.5.
%MCSE	AUTOCORLAG=*numeric*	Specifies the maximum number of autocorrelation lags used in computing the Monte Carlo standard error estimates. By default, AUTOCORLAG=MIN(500, NOBS/4), where NOBS is the sample size of the input data set.
%RAFTERY	Q=*numeric*	Specifies the order of the quantile of interest. By default, Q=0.025.
	R=*numeric*	Specifies the margin of error for measuring the accuracy of estimation of the quantile. By default, R=0.005.
	S=*numeric*	Specifies the probability of attaining the accuracy of the estimation of the quantile. By default, S=0.95.
	EPS=*numeric*	Specifies the tolerance level for the stationary test. By default, EPS=0.001.
%POSTACF	LAGS=%STR(*numeric-list*)	Specifies autocorrelation lags calculated. The default values are 1, 5, 10, and 50.
%POSTINT	ALPHA=*value*	Specifies the α level ($0 < \alpha < 1$) for the interval estimates. By default, ALPHA=0.05.

For example, the following statement calculates and displays autocorrelation at lags 1, 6, 11, 50, and 100. Note that the lags in the *numeric-list* need to be separated by commas ",".

```
%PostACF(data=post, var=&parms, lags=%str(1 to 15 by 5, 50, 100));
```

Gamma and Inverse Gamma Distributions

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcgsm.sas.)

The gamma and inverse gamma distributions are widely used in Bayesian analysis. With their respective scale and inverse scale parameterizations, they are a frequent source of confusion in the field. This section aims to clarify their parameterizations and common usages.

The gamma distribution is often used as the conjugate prior for the precision parameter ($\tau = 1/\sigma^2$) in a normal distribution. See Table 80.19 in the section "Standard Distributions" on page 6275 for the density definitions. You can specify the distribution in two ways:

- **gamma(shape=, scale=)** which has mean **shape·scale** and variance $\mathbf{shape} \cdot \mathbf{scale}^2$
- **gamma(shape=, iscale=)** which has mean $\frac{\mathbf{shape}}{\mathbf{iscale}}$ and variance $\frac{\mathbf{shape}}{\mathbf{iscale}^2}$

The parameterization of the gamma distribution that is preferred by most Bayesian analysts is to have the same number in both hyperparameter positions, which results in a prior distribution that has mean 1. To do this, you should use the **iscale=** parameterization. In addition, if you choose a small value (for example, 0.01), the prior distribution takes on a large variance (100 in this example). To specify this prior in PROC MCMC, use **gamma(shape=0.01, iscale=0.01)**[2], *not* **gamma(shape=0.01, scale=0.01)**.

If you specify the **scale=** parameterization, as in **gamma(shape=0.01, scale=0.01)**, you would get a prior distribution that has mean 0.0001 and variance 0.000001. This would lead to a completely different posterior inference: the prior would push the precision parameter estimate close to 0, or the variance estimate to a large value.

The inverse gamma distribution is often used as the conjugate prior of the variance parameter (σ^2) in a normal distribution. See Table 80.22 in the section "Standard Distributions" on page 6275 for the density definitions. Similar to the gamma distribution, you can specify the inverse gamma distribution in two ways:

- **igamma(shape=, scale=)**
- **igamma(shape=, iscale=)**

The inverse gamma distribution does not have a mean when the shape parameter is less than or equal to 1 and does not have a variance when the shape parameter is less than or equal to 2.

A gamma prior distribution on the precision is the equivalent to an inverse gamma prior distribution on the variance. The equivalency is the following:

$$\tau \sim \mathbf{gamma(shape=0.01,\ iscale=0.01)} \Leftrightarrow \sigma^2 \sim \mathbf{igamma(shape=0.01,\ scale=0.01)}$$

NOTE: This mnemonic might help you remember the parameterization scheme of the distributions. If you prefer to have identical hyperparameter values in the distribution, you should specify one and only one "i.". When the "i" appears in the **igamma** distribution name for the variance parameter, choose the **scale=**

[2] Specifying the same number at both positions and choosing a small value has been popularized by the WinBUGS software program. The WinBUGS's distribution specification of **dgamma(0.01, 0.01)** is equivalent to specifying **gamma(shape=0.01, iscale=0.01)** in PROC MCMC.

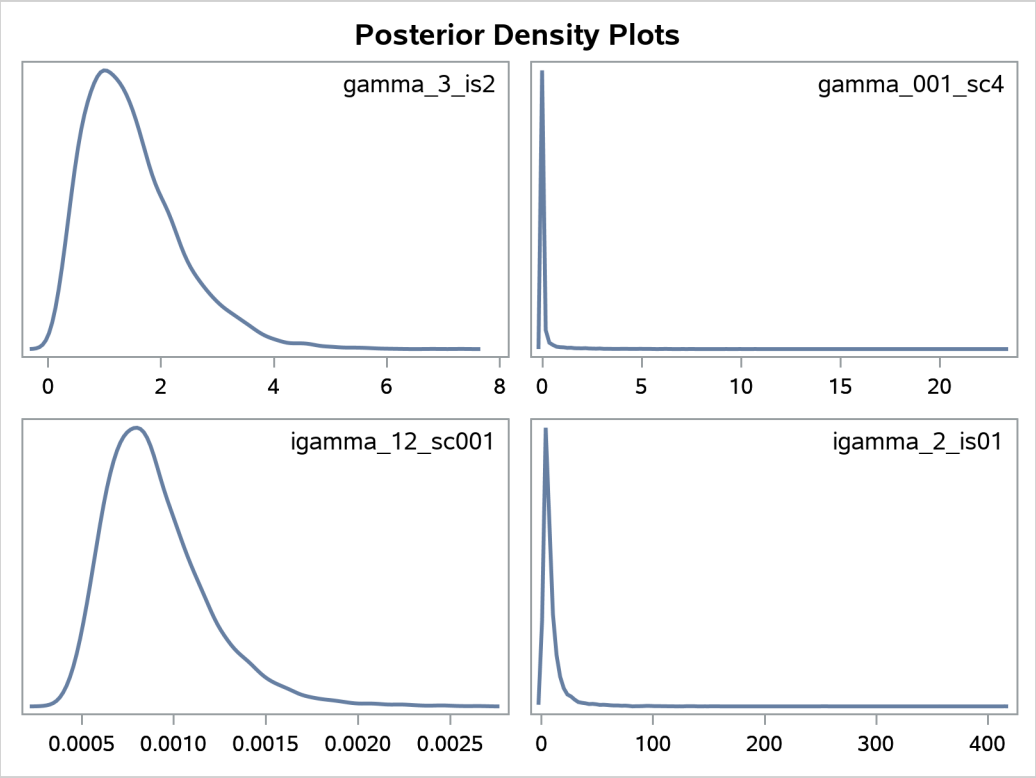
parameterization; when the "i" appears in the **iscale=** parameterization, choose the **gamma** distribution for the precision parameter.

If you are not sure about the choices of other hyperparameter values and what type of prior distributions they induce, you can write a simple PROC MCMC program and see the distributions as in the following example:

```
data a;
run;

ods graphics on;
ods select DensityPanel;
proc mcmc data=a stats=none diag=none nmc=10000 outpost=gout
   plots=density seed=1;
   parms gamma_3_is2 gamma_001_sc4 igamma_12_sc001 igamma_2_is01;
   prior gamma_3_is2     ~  gamma(shape=3, iscale=2);
   prior gamma_001_sc4   ~  gamma(shape=0.01, scale=4);
   prior igamma_12_sc001 ~ igamma(shape=12, scale=0.01);
   prior igamma_2_is01   ~ igamma(shape=2, iscale=0.1);
   model general(0);
run;
ods graphics off;
```

The preceding statements specify four different gamma and inverse gamma distributions with various scale and inverse scale parameter values. The output of kernel density plots of these four prior distributions is shown in Figure 80.26. Note how the X axis scales vary across different distributions.

Figure 80.26 Density Plots of Different Gamma and Inverse Gamma Distributions



Posterior Predictive Distribution

The posterior predictive distribution

$$p(\mathbf{y}_{\text{pred}}|\mathbf{y}) = \int p(\mathbf{y}_{\text{pred}}|\theta)p(\theta|\mathbf{y})d\theta$$

can often be used to check whether the model is consistent with data. For more information about using predictive distribution as a model checking tool, see Gelman et al. 2004, Chapter 6 and the bibliography in that chapter. The idea is to generate replicate data from $p(\mathbf{y}_{\text{pred}}|\mathbf{y})$—call them $\mathbf{y}^i_{\text{pred}}$, for $i = 1, \ldots, M$, where M is the total number of replicates—and compare them to the observed data to see whether there are any large and systematic differences. Large discrepancies suggest a possible model misfit. One way to compare the replicate data to the observed data is to first summarize the data to some test quantities, such as the mean, standard deviation, order statistics, and so on. Then compute the tail-area probabilities of the test statistics (based on the observed data) with respect to the estimated posterior predictive distribution that uses the M replicate $\mathbf{y}_{\text{pred}}$ samples.

Let $T(\cdot)$ denote the function of the test quantity, $T(\mathbf{y})$ the test quantity that uses the observed data, and $T(\mathbf{y}^i_{\text{pred}})$ the test quantity that uses the ith replicate data from the posterior predictive distribution. You calculate the tail-area probability by using the following formula:

$$\Pr(T(\mathbf{y}_{\text{pred}}) > T(\mathbf{y})|\theta)$$

The following example shows how you can use PROC MCMC to estimate this probability.

An Example for the Posterior Predictive Distribution

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcppd.sas.)

This example uses a normal mixed model to analyze the effects of coaching programs for the scholastic aptitude test (SAT) in eight high schools. For the original analysis of the data, see Rubin (1981). The presentation here follows the analysis and posterior predictive check presented in Gelman et al. (2004). The data are as follows:

```
title 'An Example for the Posterior Predictive Distribution';

data SAT;
   input effect se @@;
   ind=_n_;
   datalines;
28.39 14.9  7.94 10.2 -2.75 16.3
 6.82 11.0 -0.64  9.4  0.63 11.4
18.01 10.4 12.16 17.6
;
```

The variable effect is the reported test score difference between coached and uncoached students in eight schools. The variable se is the corresponding estimated standard error for each school. In a normal mixed effect model, the variable effect is assumed to be normally distributed:

$$\text{effect}_i \sim \text{normal}(\mu_i, \text{se}^2) \quad \text{for } i = 1, \ldots, 8$$

The parameter μ_i has a normal prior with hyperparameters (m, v):

$$\mu_i \sim \text{normal}(m, \text{var} = v)$$

The hyperprior distribution on m is a uniform prior on the real axis, and the hyperprior distribution on v is a uniform prior from 0 to infinity.

The following statements fit a normal mixed model and use the PREDDIST statement to generate draws from the posterior predictive distribution.

```
ods exclude all;
proc mcmc data=SAT outpost=out nmc=40000 seed=12;
   parms m 0;
   parms v 1 /slice;
   prior m ~ general(0);
   prior v ~ general(1,lower=0);
   random mu ~ normal(m,var=v) subject=ind monitor=(mu);
   model effect ~ normal(mu,sd=se);
   preddist outpred=pout nsim=5000;
run;
ods exclude none;
```

The ODS EXCLUDE ALL statement disables all ODS output because you are primarily interested in the samples of the predictive distribution. The HYPER, PRIOR, and MODEL statements specify the Bayesian model of interest. The PREDDIST statement generates samples from the posterior predictive distribution and stores the samples in the Pout data set. The predictive variables are named effect_1, . . ., effect_8. When no COVARIATES option is specified, the covariates in the original input data set SAT are used in the prediction. The NSIM= option specifies the number of predictive simulation iterations.

The following statements use the Pout data set to calculate the four test quantities of interest: the average (mean), the sample standard deviation (sd), the maximum effect (max), and the minimum effect (min). The output is stored in the Pred data set.

```
data pred;
   set pout;
   mean = mean(of effect:);
   sd = std(of effect:);
   max = max(of effect:);
   min = min(of effect:);
run;
```

The following statements compute the corresponding test statistics, the mean, standard deviation, and the minimum and maximum statistics on the real data and store them in macro variables. You then calculate the tail-area probabilities by counting the number of samples in the data set Pred that are greater than the observed test statistics based on the real data.

```
proc means data=SAT noprint;
   var effect;
   output out=stat mean=mean max=max min=min stddev=sd;
run;

data _null_;
   set stat;
   call symputx('mean',mean);
```

```
   call symputx('sd',sd);
   call symputx('min',min);
   call symputx('max',max);
run;

data _null_;
   set pred end=eof nobs=nobs;
   ctmean + (mean>&mean);
   ctmin + (min>&min);
   ctmax + (max>&max);
   ctsd + (sd>&sd);
   if eof then do;
      pmean = ctmean/nobs; call symputx('pmean',pmean);
      pmin = ctmin/nobs; call symputx('pmin',pmin);
      pmax = ctmax/nobs; call symputx('pmax',pmax);
      psd = ctsd/nobs; call symputx('psd',psd);
   end;
run;
```

You can plot histograms of each test quantity to visualize the posterior predictive distributions. In addition, you can see where the estimated *p*-values fall on these densities. Figure 80.27 shows the histograms. To put all four histograms on the same panel, you need to use PROC TEMPLATE to define a new graph template. (See Chapter 24, "Statistical Graphics Using ODS.") The following statements define the template **twobytwo**:

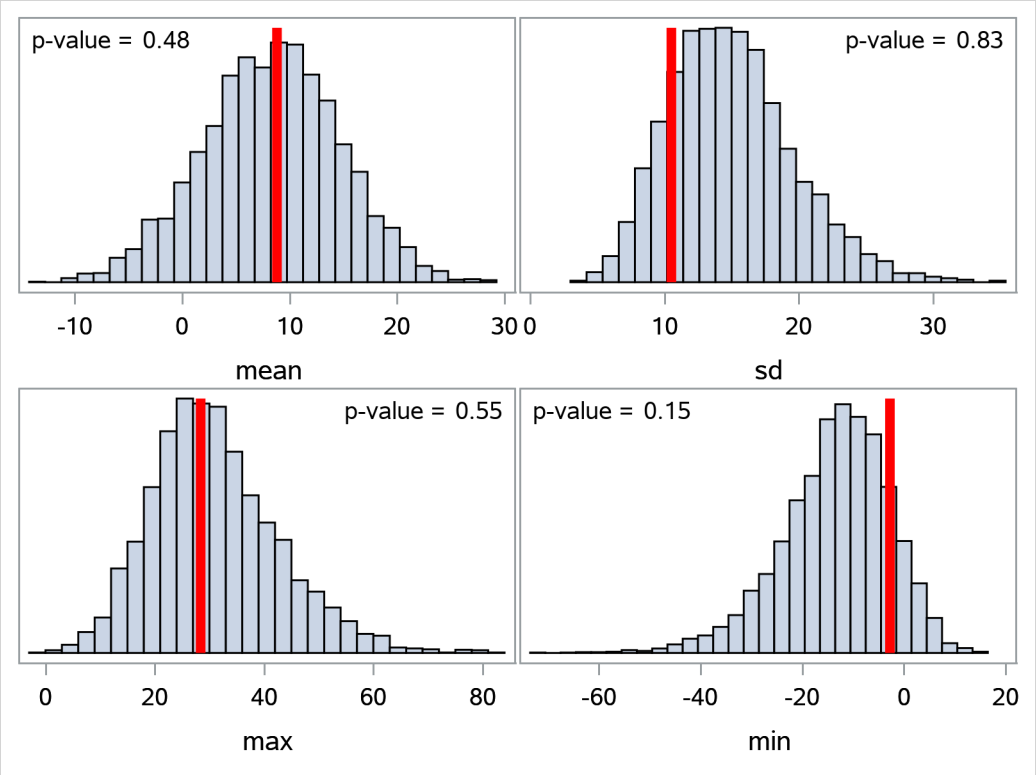
```
proc template;
   define statgraph twobytwo;
      begingraph;
         layout lattice / rows=2 columns=2;
            layout overlay / yaxisopts=(display=none)
                             xaxisopts=(label="mean");
               layout gridded / columns=2 border=false
                                autoalign=(topleft topright);
                  entry halign=right "p-value =";
                  entry halign=left eval(strip(put(&pmean, 12.2)));
               endlayout;
               histogram mean / binaxis=false;
               lineparm x=&mean y=0 slope=. /
                        lineattrs=(color=red thickness=5);
            endlayout;
            layout overlay / yaxisopts=(display=none)
                             xaxisopts=(label="sd");
               layout gridded / columns=2 border=false
                                autoalign=(topleft topright);
                  entry halign=right "p-value =";
                  entry halign=left eval(strip(put(&psd, 12.2)));
               endlayout;
               histogram sd / binaxis=false;
               lineparm x=&sd y=0 slope=. /
                        lineattrs=(color=red thickness=5);
            endlayout;
            layout overlay / yaxisopts=(display=none)
                             xaxisopts=(label="max");
               layout gridded / columns=2 border=false
                                autoalign=(topleft topright);
```

```
                entry halign=right "p-value =";
                entry halign=left eval(strip(put(&pmax, 12.2)));
             endlayout;
             histogram max / binaxis=false;
             lineparm x=&max y=0 slope=. /
                      lineattrs=(color=red thickness=5);
          endlayout;
          layout overlay / yaxisopts=(display=none)
                           xaxisopts=(label="min");
             layout gridded / columns=2 border=false
                              autoalign=(topleft topright);
                entry halign=right "p-value =";
                entry halign=left eval(strip(put(&pmin, 12.2)));
             endlayout;
             histogram min / binaxis=false;
             lineparm x=&min y=0 slope=. /
                      lineattrs=(color=red thickness=5);
          endlayout;
       endlayout;
    endgraph;
 end;
run;
```

You call PROC SGRENDER to create the graph, which is shown in Figure 80.27. (See the SGRENDER procedure in the *SAS ODS Graphics: Procedures Guide*.) There are no extreme *p*-values observed; this supports the notion that the predicted results are similar to the actual observations and that the model fits the data.

```
proc sgrender data=pred template=twobytwo;
run;
```

Figure 80.27 Posterior Predictive Distribution Check for the SAT example



Note that the posterior predictive distribution is not the same as the prior predictive distribution. The prior predictive distribution is $p(\mathbf{y})$, which is also known as the marginal distribution of the data. The prior predictive distribution is an integral of the likelihood function with respect to the prior distribution

$$p(\mathbf{y}_{\text{pred}}) = \int p(\mathbf{y}_{\text{pred}}|\theta)p(\theta)d\theta$$

and the distribution is not conditional on observed data.

Handling of Missing Data

PROC MCMC automatically augments missing values[3] via the use of the MODEL statement. PROC MCMC treats missing values as unknown parameters, assigns distributions to the variables, and incorporates the sampling of the missing data as part of the Markov chain.

You can use the MISSING= option in the PROC MCMC statement to specify how you want PROC MCMC to handle the missing values.

- If you specify MISSING=CC (CC stands for complete cases), PROC MCMC discards all observations that have missing or partial missing values before carrying out the simulation.

[3] A missing value is usually, although not necessarily, represented by a single period (.) in the input data set.

- If you specify MISSING=AC (AC stands for all cases), PROC MCMC neither discards any missing values nor augments them.
- If you specify MISSING=CCMODELY, PROC MCMC treats missing response values as parameters and incorporates them as part of the simulation. But the procedure discards all observations that have missing covariates. A covariate is a data set variable that appears in the program but does not appear to the left of the tilde in the MODEL statement.
- If you specify MISSING=ACMODELY, PROC MCMC samples missing responses without discarding observations that contain missing values in covariates.

Generally speaking, there are three types of missing data models, as discussed by Rubin (1976). Also see Little and Rubin (2002) for a comprehensive treatment of missing data analysis. The rest of this section provides an overview of these three types of missing data models and explains how to use PROC MCMC to fit them.

Missing Completely at Random (MCAR)

Data are said to be MCAR if the probability of a missing value (or the failure of observing a value) does not depend on any other observations in the data set, regardless of whether they are observed or missing. That is, the observed and unobserved values are independent of each other: if y_i is missing, it is MCAR if the probability of observing y_i is independent of other y_j (and other covariates x_i) in the data set. Under this assumption, both the observed and unobserved data are random samples of all the data; hence, fitting a model based only on the observed data does not introduce any biases. This type of analysis is called a complete-case analysis. To carry out a complete-case analysis, you must specify MISSING=CC in the PROC MCMC statement.

Missing at Random (MAR)

Data are said to be MAR if the probability of a missing value can depend on some observed quantities but does not depend on any unobserved data. For example, suppose that x_i are completely observed for all observations and some y_i are missing. MAR states that the probability of observing y_i is independent of other missing y_i (values that could have been observed) and that it depends only on x_i (and, potentially, observed y_i).

The MAR assumption states that the missing y_i are no longer random samples and that they need to be modeled (via the likelihood specification of the missing values). At the same time, the independence assumption of the missing values on the unobserved quantities states that the missing mechanism (usually an binary indicator variable such that $r_i = 1$ if y_i is missing and $r_i = 0$ otherwise) can be ignored and does not need to be taken into account. Hence, MAR is sometimes referred to as *ignorably missing*. It is not the missing values that can be ignored, it is the missing mechanism that can be ignored.

By default, PROC MCMC treats the missing data as MAR (this assumes that you do not input a binary indicator variable r_i and model it specifically): each missing value becomes an extra parameter and PROC MCMC updates it in every iteration. PROC MCMC assumes that both the missing values and observed values arise from the same distribution (which is specified in the MODEL statement),

$$\mathbf{y} = \{\mathbf{y_{obs}}, \mathbf{y_{mis}}\} \sim f(\mathbf{y}|\theta)$$

where $\mathbf{y}$ consists of observed ($\mathbf{y_{obs}}$) and missing ($\mathbf{y_{mis}}$) values, and $f(\mathbf{y}|\theta)$ is the likelihood function with parameters θ.

You can use the MODEL statement to model missing covariates. Using multiple MODEL statements enables you to specify, for example, a marginal distribution for missing values in covariate x and a conditional distribution for the response variable y given x as follows:

```
model x ~ normal(alpha, var=s2_x);
model y ~ normal(beta * x, var=s2_y);
```

In each iteration, PROC MCMC draws samples for every missing value in variable x, then every missing value in variable y, conditional on the drawn values of the x variable.

Missing Not at Random (MNAR)

Data are said to be MNAR if the probability of a missing value depends on unobserved data (or data that could have been observed): the probability that y_i is missing depends on the missing values of other y_i. This is a very general scenario that assumes that the missing mechanism is no longer ignorable (it is sometimes referred to as *nonignorably missing*) and that a model for the missing mechanism is required in order to make correct inferences about the model parameters.

Let $\mathbf{R} = (r_1, \ldots, r_n)$ be the missing value indicator for $\mathbf{Y} = (y_1, \ldots, y_n)$, where $r_i = 1$ if y_i is missing and $r_i = 0$ otherwise. This $\mathbf{R}$ is usually part of an input data set where you preprocess the response variable and create this missing value indicator variable. Modeling MNAR data implies that you must specify a joint likelihood function over $\mathbf{R}$ and $\mathbf{Y}$: $f(\mathbf{R}, \mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})$, where $\mathbf{X}$ represents the covariates and $\boldsymbol{\theta}$ represents the model parameters. This joint distribution can be factored in two ways: a pattern-mixture model and a selection model.

The *selection model* factors the joint distribution $\mathbf{R}$ and $\mathbf{Y}$ into a marginal distribution for $\mathbf{Y}$ and a conditional distribution for $\mathbf{R}$,

$$f(\mathbf{R}, \mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) \propto f(\mathbf{Y}|\mathbf{X}, \boldsymbol{\alpha}) \cdot f(\mathbf{R}|\mathbf{Y}, \mathbf{X}, \boldsymbol{\beta})$$

where $\boldsymbol{\theta} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$, $f(\mathbf{R}|\mathbf{Y}, \mathbf{X}, \boldsymbol{\alpha})$ is usually a binary model with a logit or probit link that involves regression parameters $\boldsymbol{\alpha}$, and $f(\mathbf{Y}|\mathbf{X}, \boldsymbol{\beta})$ is the sampling distribution that generates y_i with model parameters $\boldsymbol{\beta}$.

The *pattern-mixture model* factors the opposite way, a marginal distribution for $\mathbf{R}$ and a conditional distribution for $\mathbf{Y}$,

$$f(\mathbf{R}, \mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) \propto f(\mathbf{R}|\mathbf{X}, \boldsymbol{\gamma}) \cdot f(\mathbf{Y}|\mathbf{R}, \mathbf{X}, \boldsymbol{\delta})$$

where $\boldsymbol{\theta} = (\boldsymbol{\gamma}, \boldsymbol{\delta})$.

You can use PROC MCMC to fit either model by specifying multiple MODEL statements: one for the marginal distribution and one for the conditional distribution. Suppose that the variable r is the missing data indicator, which is modeled using a logit model, and that the response variable y is a Poisson regression that includes the missing variable indicator as one of its covariates. The following statements are a PROC MCMC program that fits a pattern-mixture model:

```
pi = logistic(alpha * x1);
model r ~ binary(pi);
mu = beta0 + beta1 * x2 + beta3 * r;
model y ~ poisson(exp(mu));
```

The first MODEL statement uses a binary model with logit link to model the missing mechanism, and the second MODEL statement models the response variable with a Poisson regression that includes the missing

value indicator as one of its covariates. Each of the two sets of regression has its covariates and regression coefficients. If this hypothetical data set contained missing values in covariates x1 and x2, you could add two more MODEL statements to handle each variable as follows:

```
model x1 ~ normal(mu1, var=s2_x1);
pi = logistic(alpha * x1);
model r ~ binary(pi);
model x2 ~ normal(mu2, var=s2_x2);
mu = beta0 + beta1 * x2 + beta3 * r;
model y ~ poisson(exp(mu));
```

Functions of Random-Effects Parameters

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcmre.sas.)

When you specify a RANDOM statement in a program, PROC MCMC internally creates a random-effects parameter for every unique value in the SUBJECT= variable. You can calculate any transformations of these random-effects parameters by applying SAS functions to the effect, and you can use the transformed variable in the subsequent statements. For example, the following statements perform a logit transformation of an effect:

```
random u ~ normal(mu, var=s2) subject=students;
p = logistic(u);
...
```

The value of the variable p changes with u as the procedure steps through the input data set: for different unique values of the students variable, u takes on a different parameter value, and p changes accordingly.

To save all the transformed values in p to the OUTPOST= data set, you cannot just specify the MONITOR=(p) option in the PROC MCMC statement. With such a specification, PROC MCMC can save only one value of p (usually the value associated with the last observation in the data set); it cannot save all values. To output all transformed values, you must create an array to store every transformation and use the MONITOR= option to save the entire array to the OUTPOST= data set. The difficult part of the programming involves the creation of the correct array index to use in different types of the SUBJECT= variables. The rest of this section describes how to monitor functions of random-effects parameters in different situations.

Indexing Subject Variable

This subsection describes how to monitor transformation of an effect u when the students variable is an indexing subject variable. An indexing subject variable is an integer variable that takes value from one to the total number of unique subjects in a variable. In other words, the variable can be used as an index in a SAS array. The indexing subject variable does not need to be sorted for the example code in this section to work. An example of an indexing variable takes the values of (`1 2 3 4 5 1 2 3 4 5`), where the total number of observation is $n=10$ and the number of unique values is $m=5$.

The following statements create an indexing variable students in the data set a:

```
data a;
   input students @@;
   datalines;
1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10
;
```

The following statements run a random-effects model without any response variables. There are only random-effects parameters in the model; the program calculates the logit transformation of each effect, saves the results to the OUTPOST= data set, and produces Figure 80.28:

```
proc mcmc data=a monitor=(p) diag=none stats=none outpost=a1
   plots=none seed=1;
   array p[10];
   random u ~ n(0, sd=1) subject=students;
   p[students] = logistic(u);
   model general(0);
run;

proc print data=a1(obs=3);
run;
```

The ARRAY statement creates an array p of size 10, which is the number of unique values in students. The p array stores all the transformed values. The RANDOM statement declares u to be an effect with the subject variable students. The P[STUDENTS] assignment statement calculates the logit transformations of u and saves them in appropriate array elements—this is why the students variable must be an indexing variable. Because the students variable used in the p[] array is also the subject variable in the RANDOM statement, PROC MCMC can match each random-effects parameter with the corresponding element in array p. The MONITOR= option monitors all elements in p and saves the output to the a1 data set. The a1 data set contains variables p1–p10. Figure 80.28 shows the first three observations of the OUTPOST= data set.

Figure 80.28 Monitor Functions of Random Effect u

Obs	Iteration	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	u_1	u_2
1	1	0.5050	0.7334	0.5375	0.5871	0.6862	0.4944	0.5740	0.5991	0.6812	0.8678	0.0198	1.0120
2	2	0.5563	0.4254	0.7260	0.5280	0.4593	0.5797	0.5813	0.7360	0.9079	0.3351	0.2261	-0.3006
3	3	0.6132	0.8031	0.4735	0.3135	0.7047	0.5273	0.6761	0.2379	0.2938	0.3986	0.4607	1.4058

Obs	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_10	LogReff	LogLike	LogPost
1	0.1504	0.3521	0.7826	-0.0225	0.2983	0.4019	0.7595	1.8819	-12.2658	0	-12.2658
2	0.9742	0.1120	-0.1630	0.3213	0.3281	1.0253	2.2887	-0.6854	-13.2393	0	-13.2393
3	-0.1060	-0.7837	0.8698	0.1092	0.7361	-1.1641	-0.8772	-0.4112	-12.3984	0	-12.3984

The variable p1 is the logit transformation of the variable u_1, p2 is the logit transformation of the variable u_2, and so on.

The same idea works for a students variable that is unsorted. The following statements create an unsorted indexing variable students with repeated measures in each subject, fit the same model, and produce Figure 80.29:

```
data a;
   input students @@;
   datalines;
   1 1 1 3 5 3 4 5 3 1 5 5 4 4 2 2 2 2 4 3
;
proc mcmc data=a monitor=(p) diag=none stats=none outpost=a1
   plots=none seed=1;
   array p[5];
   random u ~ n(0, sd=1) subject=students;
   p[students] = logistic(u);
   model general(0);
run;

proc print data=a1(obs=3);
run;
```

Figure 80.29 Monitor Functions of Random Effect u When the students Variable is Unsorted

Obs	Iteration	p1	p2	p3	p4	p5	u_1	u_3	u_5	u_4	u_2	LogReff	LogLike	LogPost
1	1	0.5050	0.6862	0.7334	0.5871	0.5375	0.0198	1.0120	0.1504	0.3521	0.7826	-5.4865	0	-5.4865
2	2	0.4944	0.8678	0.5740	0.6812	0.5991	-0.0225	0.2983	0.4019	0.7595	1.8819	-6.7793	0	-6.7793
3	3	0.5563	0.4593	0.4254	0.5280	0.7260	0.2261	-0.3006	0.9742	0.1120	-0.1630	-5.1595	0	-5.1595

There are five random-effects parameters in this example, and the array p also has five elements. The values p1–p5 are the transformations of u_1–u_5, respectively. The u variables are not sorted from u_1 to u_5 because PROC MCMC creates the names according to the order by which the subject variable appears in the input data set. Nevertheless, because students is an indexing variable, the first element p[1] stores the transformation that corresponds to **`students=1`** (which is u_1), the second element p[2] stores the transformation that corresponds to **`students=2`**, and so on.

Non-Indexing Subject Variable

A non-indexing subject variable can take values of character literals (for example, names) or numerals (for example, ZIP code or a person's weight). This section illustrates how to monitor functions of random-effects parameters in these situations.

Suppose you have unsorted character literals as the subject variable:

```
data a;
   input students$ @@;
   datalines;
   smith john john mary kay smith lizzy ben ben dylan
   ben toby abby mary kay kay lizzy ben dylan mary
;
```

A statement such as following does not work anymore because a character variable cannot be used as an array index:

```
p[students] = logistic(u);
```

In this situation, you usually need to do two things: (1) find out the number of unique values in the subject variable, and (2) create a numeric index variable that replaces the students array index. You can use the following statements to do the first task:

```
proc sql noprint;
   select count(distinct(students)) into :nuniq from a;
quit;
%put &nuniq;
```

The PROC SQL call counts the distinct values in the students variable and saves the count to the macro variable &nuniq. The macro variable is used later to specify the element size of the p array. In this example, the a data set contains 20 observations and 9 unique elements (the value of &nuinq).

The following statements create an Index variable in the a data set that is in the order by which the students names appear in the data set:

```
proc freq data=a order=data noprint;
   tables students / out=_f(keep=students);
run;

proc print data=_f;
run;

data a(drop=n);
   set a;
   do i = 1 to nobs until(students=n);
      set _f(keep=students rename=(students=n)) point=i nobs=nobs;
      Index = i;
   end;
run;

proc print data=a;
run;
```

The PROC FREQ call identifies the unique students names and saves them to the _f data set, which is displayed in Figure 80.30.

Figure 80.30 Unique Names in the Variable Students

Obs	students
1	smith
2	john
3	mary
4	kay
5	lizzy
6	ben
7	dylan
8	toby
9	abby

The DATA step steps through the a data set and creates an Index variable to match the order in which the

students names appear in the data set. The new a data set[4] is displayed in Figure 80.31.

Figure 80.31 New Index Variable in the a Data Set

Obs	students	Index
1	smith	1
2	john	2
3	john	2
4	mary	3
5	kay	4
6	smith	1
7	lizzy	5
8	ben	6
9	ben	6
10	dylan	7
11	ben	6
12	toby	8
13	abby	9
14	mary	3
15	kay	4
16	kay	4
17	lizzy	5
18	ben	6
19	dylan	7
20	mary	3

Student smith is the first subject, and his Index value is one. The same student appears again in the sixth observation, which is given the same Index value. Now, this Index variable can be used to index the p array, in a similar fashion as demonstrated in previous programs:

```
data _f;
   set _f;
   subj = compress('p_'||students);
run;
proc sql noprint;
   select subj into :pnames separated by ' ' from _f;
quit;
%put &pnames;

proc mcmc data=a monitor=(p) diag=none stats=none outpost=a1
   plots=none seed=1;
   array p[&nuniq] &pnames;
   random u ~ n(0, sd=1) subject=students;
   p[index] = logistic(u);
   model general(0);
run;

proc print data=a1(obs=3);
run;
```

[4]The programming code that creates and adds the Index variable to the data set a keeps all variables from the original data set and does not discard them.

The first part of the DATA step and the PROC SQL call create array names for the p array that match the subject names in the students variable. It is not necessary to include these steps for the PROC MCMC program to work, but it makes the output more readable. The first DATA step steps through the _f data set and creates a subj variable that concatenates the prefix characters p_ with the names of the students. The PROC SQL calls put all the subj values to a macro called &pnames, which looks like the following:

```
p_smith p_john p_mary p_kay p_lizzy p_ben p_dylan p_toby p_abby
```

In the PROC MCMC program, the ARRAY statement defines an p array with size &nuniq (9), and use the macro variable &pnames to name the array elements. The P[INDEX] assignment statement uses the Index variable to find the correct array element to store the transformation. Figure 80.32 displays the first few observations of the OUTPOST=a1 data set.

Figure 80.32 First Few Observations of the Outpost Data Set

Obs	Iteration	p_smith	p_john	p_mary	p_kay	p_lizzy	p_ben	p_dylan	p_toby	p_abby	u_smith	u_john
1	1	0.5050	0.7334	0.5375	0.5871	0.6862	0.4944	0.5740	0.5991	0.6812	0.0198	1.0120
2	2	0.8678	0.5563	0.4254	0.7260	0.5280	0.4593	0.5797	0.5813	0.7360	1.8819	0.2261
3	3	0.9079	0.3351	0.6132	0.8031	0.4735	0.3135	0.7047	0.5273	0.6761	2.2887	-0.6854

Obs	u_mary	u_kay	u_lizzy	u_ben	u_dylan	u_toby	u_abby	LogReff	LogLike	LogPost
1	0.1504	0.3521	0.7826	-0.0225	0.2983	0.4019	0.7595	-9.5761	0	-9.5761
2	-0.3006	0.9742	0.1120	-0.1630	0.3213	0.3281	1.0253	-11.2370	0	-11.2370
3	0.4607	1.4058	-0.1060	-0.7837	0.8698	0.1092	0.7361	-13.1867	0	-13.1867

There are nine random-effects parameters (u_smith, u_john, and so on). There are nine elements of the p array (p_smith, p_john, and so on); each is the logit transformation of corresponding u elements.

You can use the same statements for subject variables that are numeric non-indexing variables. The following statements create a students variable that take large numbers that cannot be used as indices to an array. The rest of the program monitors functions of the effect u. The output is not displayed here.

```
data a;
   call streaminit(1);
   do i = 1 to 20;
      students = rand("poisson", 20);
      output;
   end;
   drop i;
run;

proc sql noprint;
   select count(distinct(students)) into :nuniq from a;
quit;
%put &nuniq;

proc freq data=a order=data noprint;
   tables students / out=_f(keep=students);
run;

data a(drop=n);
```

```
   set a;
   do i = 1 to nobs until(students=n);
      set _f(keep=students rename=(students=n)) point=i nobs=nobs;
      Index = i;
   end;
run;

data _f;
   set _f;
   subj = compress('p_'||students);
proc sql noprint;
   select subj into :pnames separated by ' ' from _f;
   quit;
%put &pnames;

proc mcmc data=a monitor=(p) diag=none stats=none outpost=a1
   plots=none seed=1;
   array p[&nuniq] &pnames;
   random u ~ n(0, sd=1) subject=students;
   p[index] = logistic(u);
   model general(0);
run;

proc print data=a1(obs=3);
run;
```

Spatial Prior

(View the complete code for this example at https://github.com/sassoftware/doc-supplement-statug/tree/main/Examples/m-n/mcmcspa.sas.)

You can use the conditional autoregressive Gaussian (NORMALCAR) prior in the RANDOM statement to model random effects that are spatially correlated. Suppose that the areas are indexed by the variable ID in a SAS data set. To model spatial dependence among areas, you can use the following statement:

```
random a ~ normalcar(neighbors=, num=, sd|var|prec=) subject=ID;
```

The NUM= option specifies the data set variable that indicates the number of neighbors for each ID, and the NEIGHBORS= option specifies the prefix (without quotation marks) of data variables that contain the adjacent IDs (neighbors) for each subject. The names of all neighboring variables must have this same prefix and must be followed by numbers 1, 2, . . ., N, where N is the maximum number of neighbors that a subject has in the data set. Missing values are allowed in the neighboring ID variables if a subject has fewer than N neighbors.

The default specification of the NORMALCAR prior corresponds to the full conditional distribution of each subject is the following CAR model,

$$\theta_i|\theta_{-i} \sim \text{normal}\left(\sum_{j\in N(i)} \theta_j/m_i, s^2\right)$$

where $N(i)$ is the set of neighbors of area i, m_i is the total number of neighbors of area i, and s is the standard deviation of the normal distribution.

You can also specify a CAR prior in which the variance is weighed by the number of neighbors:

$$\theta_i|\theta_{-i} \sim \text{normal}\left(\sum_{j\in N(i)} \theta_j/m_i, s^2/m_i\right)$$

To specify this prior, you use the following statement, where NumNei is the variable of number of neighbors and s2 is the variance parameter:

```
random a ~ normalcar(neighbor=, num=NumNei, var=s2/NumNei) subject=ID;
```

In most cases, the CAR prior is an improper prior because the corresponding covariance matrix of $\theta = \{\theta_i, \ldots, \theta_p\}$ is less than full rank. If the posterior distribution is proper, then using an improper prior is not an issue. A common practice is to re-center all the θ draws after each iteration (Banerjee, Carlin, and Gelfand 2004, p. 164). In effect, this re-centering sets the prior mean of the random effects to be zero. The centering step is performed by using the CENTER option in the RANDOM statement.

Here is an example that illustrates the use of the NORMALCAR prior distribution.

The following statements create a SAS data set that contains lung cancer data from a London Health Authority annual report (Thomas et al. 2004):

```
title 'Spatial Analysis';

data London;
   input ID N1-N9 NumNei O E Depriv;
   datalines;
  1   4   8   9  12  17   .   .   .   .  5    4   7.2090    1.233
  2   7  10  13  14   .   .   .   .   .  4    8   7.8144    8.162

   ... more lines ...

 44  22  24  25  26  31  32  41   .   .  7    7   6.3737    3.961
run;
```

The 44 wards are indicated by the ID variable. N1–N9 are variables of neighboring indices for each ward. For example, ward 1 is adjacent to five wards: 4, 8, 9, 12, and 17. The NumNei variable is the total number of neighbor wards that each ward has. The maximum number of neighbors wards is nine in the data set, so every observation must have nine neighboring ID variables. Missing data (".") are used in neighboring ID variables that have fewer than nine neighbors. The O and E variables are simulated observed and expected counts of lung cancer incidence. The Depriv variable is the socioeconomic variable for each ward.

The following random-effects model is considered,

$$\begin{aligned} \mathsf{O}_i &\sim \text{Poisson}(\mu_i) \\ \log(\mu_i) &= \log(\mathsf{E}_i) + \alpha + \beta\mathsf{Deprive}_i + b_i + h_i \end{aligned}$$

where the random effects b_i are given a spatial prior and h_i is a normal prior for each area. You can fit this model using the following PROC MCMC statements:

```
ods select none;
proc mcmc data=london seed=615926 nbi=10000 nmc=50000 thin=10
   plots=none outpost=londonpost;
   parms tau_b 0.5 tau_h 0.2;
   parms alpha 0 beta 0;
   prior tau: ~ gamma(0.5, is=0.0005);
   prior alpha ~ general(0);
   prior beta ~ n(0, prec=1e-5);
   random h ~ n(0, prec=tau_h) s=id;
   random b ~ normalcar(neighbors=n,num=numnei,prec=tau_b*numnei) s=id;
   mu=e*exp(alpha + beta*depriv + b + h);
   model o ~ poisson(mu);
run;
```

The two RANDOM statements specify the h_i and the b_i random effects; the NORMALCAR prior is assigned to b_i. In the NORMALCAR prior, NEIGHBORS=N specifies the prefix "n" for all the neighboring ID variables, NUM=NUMNEI specifies the number of neighbors that each ID has, and PREC=TAU_B*NUMNEI weights the precision parameter by the number of neighbors in the model.

The example assumes that the adjacency matrices have been created and stored in a SAS data set. More often, you might have the map polygon data in an .shp file. In that case, you can use the %NEIGHBOR autocall macro to import simple polygons and to compute the adjacent units and number of neighbors for each spatial unit:

```
%neighbor("map.shp", IDVAR=ID, OUTNBRS=neighbor, OUTADJ=adjacent);
```

The first argument is the path to the shape file, which must be specified in quotation marks. The IDVAR= option specifies the ID variable that identifies the sites or the units. The OUTNBRS= and OUTADJ= options save the neighborhood and adjacency information, respectively, to two SAS data sets. The default values for the IDVAR=, OUTNBRS=, and OUTADJ= options are ID, Neighbor, and Adjacent, respectively. You can combine these SAS data sets with covariates and response information before you use PROC MCMC to fit the spatial model.

Floating Point Errors and Overflows

When performing a Markov chain Monte Carlo simulation, you must calculate a proposed jump and an objective function (usually a posterior density). These calculations might lead to arithmetic exceptions and overflows. A typical cause of these problems is parameters with widely varying scales. If the posterior variances of your parameters vary by more than a few orders of magnitude, the numerical stability of the optimization problem can be severely reduced and can result in computational difficulties. A simple remedy is to rescale all the parameters so that their posterior variances are all approximately equal. Changing the SCALE= option might help if the scale of your parameters is much different than one. Another source of numerical instability is highly correlated parameters. Often a model can be reparameterized to reduce the posterior correlations between parameters.

If parameter rescaling does not help, consider the following actions:

- provide different initial values or try a different seed value
- use boundary constraints to avoid the region where overflows might happen
- change the algorithm (specified in programming statements) that computes the objective function

Problems Evaluating Code for Objective Function

The initial values must define a point for which the programming statements can be evaluated. However, during simulation, the algorithm might iterate to a point where the objective function cannot be evaluated. If you program your own likelihood, priors, and hyperpriors by using SAS statements and the GENERAL function in the MODEL, PRIOR, AND HYPERPRIOR statements, you can specify that an expression cannot be evaluated by setting the value you pass back through the GENERAL function to missing. This tells the PROC MCMC that the proposed set of parameters is invalid, and the proposal will not be accepted. If you use the shorthand notation that the MODEL, PRIOR, AND HYPERPRIOR statements provide, this error checking is done for you automatically.

Long Run Times

PROC MCMC can take a long time to run for problems with complex models, many parameters, or large input data sets. Although the techniques used by PROC MCMC are some of the best available, they are not guaranteed to converge or proceed quickly for all problems. Ill-posed or misspecified models can cause the algorithms to use more extensive calculations designed to achieve convergence, and this can result in longer run times. You should make sure that your model is specified correctly, that your parameters are scaled to the same order of magnitude, and that your data reasonably match the model that you are specifying.

To speed general computations, you should check over your programming statements to minimize the number of unnecessary operations. For example, you can use the proportional kernel in the priors or the likelihood and not add constants in the densities. You can also use the BEGINCNST and ENDCNST to reduce unnecessary computations on constants, and the BEGINNODATA and ENDNODATA statements to reduce observation-level calculations.

Reducing the number of blocks (the number of the PARMS statements) can speed up the sampling process. A single-block program is approximately three times faster than a three-block program for the same number of iterations. On the other hand, you do not want to put too many parameters in a single block, because blocks with large size tend not to produce well-mixed Markov chains.

If some parameters satisfy the conditional independence assumption, such as in the random-effects models or latent variable models, consider using the RANDOM statement to model these parameters. This statement takes advantage of the conditional independence assumption and can sample a larger number of parameters at a more efficient pace.

Slow or No Convergence

If the simulator is slow or fails to converge, you can try changing the model as follows:

- Change the number of Monte Carlo iterations (NMC=), or the number of burn-in iterations (NBI=), or both. Perhaps the chain just needs to run a little longer. Note that after the simulation, you can always use the DATA step or the FIRSTOBS data set option to throw away initial observations where the algorithm has not yet burned in, so it is not always necessary to set NBI= to a large value.

- Increase the number of tuning. The proposal tuning can often work better in large models (models that have more parameters) with larger values of NTU=. The idea of tuning is to find a proposal distribution that is a good approximation to the posterior distribution. Sometimes 500 iterations per tuning phase (the default) is not sufficient to find a good approximating covariance.

- Change the initial values to more feasible starting values. Sometimes the proposal tuning starts badly if the initial values are too far away from the main mass of the posterior density, and it might not be able to recover.

- Use the PROPCOV= option to start the Markov chain at better starting values. With the PROPCOV=QUANEW option, PROC MCMC optimizes the object function and uses the posterior mode as the starting value of the Markov chain. In addition, a quadrature approximation to the posterior mode is used as the proposal covariance matrix. This option works well in many cases and can improve the mixing of the chain and shorten the tuning and burn-in time.

- Parameterize your model to include conjugacy, such as using the gamma prior on the precision parameter in a normal distribution or using an inverse gamma on the variance parameter. For a list of conjugate sampling methods that PROC MCMC supports, see the section "Conjugate Sampling" on page 6272.

- Change the blocking by using the PARMS statements. Sometimes poor mixing and slow convergence can be attributed to highly correlated parameters being in different parameter blocks.

- Modify the target acceptance rate. A target acceptance rate of about 25% works well for many multi-parameter problems, but if the mixing is slow, a lower target acceptance rate might be better.

- Change the initial scaling or the TUNEWT= option to possibly help the proposal tuning.

- Consider using a different proposal distribution. If from a trace plot you see that a chain traverses to the tail area and sometimes takes quite a few simulations before it comes back, you can consider using a t proposal distribution. You can do this by either using the PROC option PROPDIST=T or using a PARMS statement option T.

- Transform parameters and sample on a different scale. For example, if a parameter has a gamma distribution, sample on the logarithm scale instead. A parameter a that has a gamma distribution is equivalent to $\log(a)$ that has an egamma distribution, with the same distribution specification. For example, the following two formulations are equivalent:

```
parm a;
prior a ~ gamma(shape = 0.001, scale = 0.001);
```

and

```
parm la;
prior la ~ egamma(shape = 0.001, scale = 0.001);
a = exp(la);
```

See "Example 80.6: Nonlinear Poisson Regression Models" on page 6392 and "Example 80.20: Using a Transformation to Improve Mixing" on page 6473. You can also use the logit transformation on parameters that have uniform(0, 1) priors. This prior is often used on probability parameters. The logit

transformation is as follows: $q = \log(\frac{p}{1-p})$. The distribution on q is the Jacobian of the transformation: $\exp(-q)(1+\exp(-q))^{-2}$.

Again, the following two formulations are equivalent:

```
parm p;
prior p ~ uniform(0, 1);
```

and

```
parm q;
lp = -q - 2 * log(1 + exp(-q));
prior q ~ general(lp);
p = 1/(1+exp(-q));
```

Precision of Solution

In some applications, PROC MCMC might produce parameter values that are not precise enough. Usually, this means that there were not enough iterations in the simulation. At best, the precision of MCMC estimates increases with the square of the simulation sample size. Autocorrelation in the parameter values deflate the precision of the estimates. For more information about autocorrelations in Markov chains, see the section "Autocorrelations" on page 174 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

Handling Error Messages

PROC MCMC does not have a debugger. This section covers a few ways to debug and resolve error messages.

Using the PUT Statement

Adding the PUT statement often helps to find errors in a program. The following statements produce an error:

```
data a;
run;

proc mcmc data=a seed=1;
   parms sigma lt w;

   beginnodata;
   prior sigma ~ unif(0.001,100);
   s2 = sigma*sigma;
   prior lt ~ gamma(shape=1, iscale=0.001);
   t = exp(lt);
   c = t/s2;
   d = 1/(s2);
   prior w ~ gamma(shape=c, iscale=d);
   endnodata;
   model general(0);
run;
```

```
ERROR: PROC MCMC is unable to generate an initial value for the
       parameter w.  The first parameter in the prior distribution is
       missing.
```

To find out why the shape parameter c is missing, you can add the put statement and examine all the calculations that lead up to the assignment of c:

```
proc mcmc data=a seed=1;
   parms sigma lt w;

   beginnodata;
   prior sigma ~ unif(0.001,100);
   s2 = sigma*sigma;
   prior lt ~ gamma(shape=1, iscale=0.001);
   t = exp(lt);
   c = t/s2;
   d = 1/(s2);
   put c= t= s2= lt=; /* display the values of these symbols. */
   prior w ~ gamma(shape=c, iscale=d);
   endnodata;

   model general(0);
run;
```

In the log file, you see the following:

```
c=. t=. s2=. lt=.
c=. t=. s2=2500.0500003 lt=1000
c=. t=. s2=2500.0500003 lt=1000
ERROR: PROC MCMC is unable to generate an initial value for the parameter w.
       The first parameter in the prior distribution is missing.
```

You can ignore the first few lines. They are the results of initial set up by PROC MCMC. The last line is important. The variable c is missing because t is the exponential of a very large number, 1000, in lt. The value 1000 is assigned to lt by PROC MCMC because none was given. The gamma prior with shape of 1 and inverse scale of 0.001 has mode 0 (see "Standard Distributions" on page 6275 for more details). PROC MCMC avoids starting the Markov chain at the boundary of the support of the distribution, and it uses the mean value here instead. The mean of the gamma prior is 1000, hence the problem. You can change how the initial value is generated by using the PROC statement INIT=RANDOM. Remember to take out the put statement once you identify the problem. Otherwise, you will see a voluminous output in the log file.

Using the HYPER Statement

You can use the HYPER statement to narrow down possible errors in the prior distribution specification. With multiple PRIOR statements in a program, you might see the following error message if one of the prior distributions is not specified correctly:

```
ERROR: The initial prior parameter specifications must yield log
       of positive prior density values.
```

This message is displayed when PROC MCMC detects an error in the prior distribution calculation but cannot pinpoint the specific parameter at fault. It is frequently, although not necessarily, associated with parameters that have GENERAL or DGENERAL distributions. If you have a complicated model with many PRIOR statements, finding the parameter at fault can be time consuming. One way is to change a subset of the PRIOR statements to HYPER statements. The two statements are treated the same in PROC MCMC and the simulation is not affected, but you get a different message if the hyperprior distributions are calculated incorrectly:

```
ERROR: The initial hyperprior parameter specifications must yield
       log of positive hyperprior density values.
```

This message can help you identify more easily which distributions are producing the error, and you can then use the PUT statement to further investigate.

Computational Resources

It is impossible to estimate how long it will take for a general Markov chain to converge to its stationary distribution. It takes a skilled and thoughtful analysis of the chain to decide whether it has converged to the target distribution and whether the chain is mixing rapidly enough. In some cases, you might be able to estimate how long a particular simulation might take. The running time of a program that does not have RANDOM statements is approximately linear to the following factors: the number of samples in the input data set, the number of simulations, the number of blocks in the program, and the speed of your computer. For an analysis that uses a data set of size *nsamples*, a simulation length of *nsim*, and a block design of *nblocks*, PROC MCMC evaluates the log-likelihood function the following number of times, excluding the tuning phase:

$$nsamples \times nsim \times nblocks$$

The faster your computer evaluates a single log-likelihood function, the faster this program runs. Suppose you have *nsamples* equal to 200, *nsim* equal to 55,000, and *nblocks* equal to 3. PROC MCMC evaluates the log-likelihood function approximately 3.3×10^7 times. If your computer can evaluate the log likelihood for one observation 10^6 times per second, this program takes approximately a half a minute to run. If you want to increase the number of simulations five-fold, the run time increases approximately five-fold.

Each RANDOM statement adds one pass through the input data at each iteration. If the Metropolis algorithm is used to sample the random-effects parameter, the conditional density (objective function) is calculated twice per pass through the data, which requires a computational resource that is approximately equivalent to adding two blocks of parameters.

Of course, larger problems take longer than shorter ones, and if your model is amenable to frequentist treatment, then one of the other SAS procedures might be more suitable. With “regular” likelihoods and a lot of data, the results of standard frequentist analysis are often asymptotically equivalent to a Bayesian approach. If PROC MCMC requires too much CPU time, then perhaps another SAS/STAT tool would be suitable.

Displayed Output

This section describes the output that PROC MCMC displays. For a quick reference of all ODS table names, see the section "ODS Table Names" on page 6359. ODS tables are arranged under four groups, which are listed in the following sections: "Model and Data Related ODS Tables" on page 6355, "Sampling Related ODS Tables" on page 6355, "Posterior Statistics Related ODS Tables" on page 6356, "Convergence Diagnostics Related ODS Tables" on page 6357, and "Optimization Related ODS Tables" on page 6359.

Model and Data Related ODS Tables

Missing Data Information Table

The "Missing Data Information" table (ODS table name MISSDATAINFO) displays the name of the response variable that contains missing values, the number of missing observations, the corresponding observation indices in the input data set, and the sampling method used in the simulation for the missing values.

Number of Observation Table

The "NObs" table (ODS table name NOBS) shows the number of observations that is in the data set and the number of observations that is used in the analysis. By default, observations with missing values are not used (see the section "Handling of Missing Data" on page 6338 for more details). This table is displayed by default.

Parameters

The "Parameters" table (ODS table name Parameters) shows the name of each parameter, the block number of each parameter, the sampling method used for the block, the initial values, and the prior or hyperprior distributions. This table is displayed by default.

REObsInfo

The "Random Effect Observation Information" table (ODS table name REObsInfo) lists the name of the random effect, each subject value, the number of observations in each subject, and their corresponding observation indices in the input data set. You can request this table by specifying the REOBSINFO option.

REParameters

The "REParameters" table (ODS table name REParameters) lists the name of the random effect, sampling algorithm, the subject variable, the number of subjects, unique values of the subject variable, and the prior distribution. This table is displayed by default if a RANDOM statement is used in the program.

Sampling Related ODS Tables

Burn-In History

The "Burn-In History" table (ODS table name BurnInHistory) shows the scales and acceptance rates for each parameter block in the burn-in phase. The table is not displayed by default and can be requested by specifying the option MCHISTORY=BRIEF | DETAILED.

Parameters Initial Value Table

The "Parameters Initial" table (ODS table name ParametersInit) shows the value of each parameter after the tuning phase. This table is not displayed by default and can be requested by specifying the option INIT=PINIT.

Posterior Samples

The "Posterior Samples" table (ODS table name PosteriorSample) stores posterior draws of all parameters. It is not printed by PROC MCMC. You can create an ODS output data set of the chain by specifying the following:

```
ODS OUTPUT PosteriorSample = SAS-data-set;
```

Sampling History

The "Sampling History" table (ODS table name SamplingHistory) shows the scales and acceptance rates for each parameter block in the main sampling phase. The table is not displayed by default and can be requested by specifying the option MCHISTORY=BRIEF | DETAILED.

Tuning Covariance

The "Tuning Covariance" table (ODS table name TuneCov) shows the proposal covariance matrices for each parameter block after the tuning phase. The table is not displayed by default and can be requested by specifying the option INIT=PINIT. For more details about proposal tuning, see the section "Tuning the Proposal Distribution" on page 6269.

Tuning History

The "Tuning History" table (ODS table name TuningHistory) shows the number of tuning phases used in establishing the proposal distribution. The table also displays the scales and acceptance rates for each parameter block at each of the tuning phases. For more information about the self-adapting proposal tuning algorithm used by PROC MCMC, see the section "Tuning the Proposal Distribution" on page 6269. The table is not displayed by default and can be requested by specifying the option MCHISTORY=BRIEF | DETAILED.

Tuning Probability Vector

The "Tuning Probability" table (ODS table name TuneP) shows the proposal probability vector for each discrete parameter block (when the option DISCRETE=GEO is specified and the geometric proposal distribution is used for discrete parameters) after the tuning phase. The table is not displayed by default and can be requested by specifying the option INIT=PINIT. For more information about proposal tuning, see the section "Tuning the Proposal Distribution" on page 6269.

Posterior Statistics Related ODS Tables

PROC MCMC calculates some essential posterior statistics and outputs them to a number of ODS tables that you can request and save individually. For details of the calculations, see the section "Summary Statistics" on page 175 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

Summary and Interval Statistics

The "Posterior Summaries and Intervals" table (ODS table name PostSumInt) contains a summary of basic point and interval statistics for each parameter. The table lists the number of posterior samples, the posterior mean and standard deviation estimates, and the 95% HPD interval estimates. This table is displayed by default.

Summary Statistics

The "Posterior Summaries" table (ODS table name PostSummaries) contains basic statistics for each parameter. The table lists the number of posterior samples, the posterior mean and standard deviation estimates, and the percentile estimates. The table is not displayed by default and can be requested by specifying the option STATISTICS=SUMMARY.

Correlation Matrix

The "Posterior Correlation Matrix" table (ODS table name Corr) contains the posterior correlation of model parameters. The table is not displayed by default and can be requested by specifying the option STATISTICS=CORR.

Covariance Matrix

The "Posterior Covariance Matrix" table (ODS table name Cov) contains the posterior covariance of model parameters. The table is not displayed by default and can be requested by specifying the option STATISTICS=COV.

Deviance Information Criterion

The "Deviance Information Criterion" table (ODS table name DIC) contains the DIC of the model. The table is not displayed by default and can be requested by specifying the option DIC. For details of the calculations, see the section "Deviance Information Criterion (DIC)" on page 177 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

Interval Statistics

The "Posterior Intervals" table (ODS table name PostIntervals) contains the equal-tail and highest posterior density (HPD) interval estimates for each parameter. The default α value is 0.05, and you can change it to other levels by using the STATISTICS= option. The table is not displayed by default and can be requested by specifying the option STATISTICS=INTERVAL.

Convergence Diagnostics Related ODS Tables

PROC MCMC has convergence diagnostic tests that check for Markov chain convergence. PROC MCMC produces a number of ODS tables that you can request and save individually. For details in calculation, see the section "Statistical Diagnostic Tests" on page 166 in Chapter 8, "Introduction to Bayesian Analysis Procedures."

Autocorrelation

The "Autocorrelations" table (ODS table name AUTOCORR) contains the first-order autocorrelations of the posterior samples for each parameter. The "Parameter" column states the name of the parameter. By default, PROC MCMC displays lag 1, 5, 10, and 50 estimates of the autocorrelations. You can request different autocorrelations by using the DIAGNOSTICS=AUTOCORR(LAGS=) option. The table is not displayed by default and can be requested by specifying the option DIAGNOSTICS=AUTOCORR.

Effective Sample Size

The "Effective Sample Sizes" table (ODS table name ESS) calculates the effective sample size of each parameter. See the section "Effective Sample Size" on page 175 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. The table is displayed by default.

Monte Carlo Standard Errors

The "Monte Carlo Standard Errors" table (ODS table name MCSE) calculates the standard errors of the posterior mean estimate. See the section "Standard Error of the Mean Estimate" on page 176 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. The table is not displayed by default and can be requested by specifying the option DIAGNOSTICS=MCSE.

Geweke Diagnostics

The "Geweke Diagnostics" table (ODS table name Geweke) lists the result of the Geweke diagnostic test. See the section "Geweke Diagnostics" on page 169 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. The table is not displayed by default and can be requested by specifying the option DIAGNOSTICS=GEWEKE.

Heidelberger-Welch Diagnostics

The "Heidelberger-Welch Diagnostics" table (ODS table name Heidelberger) lists the result of the Heidelberger-Welch diagnostic test. The test is consisted of two parts: a stationary test and a half-width test. See the section "Heidelberger and Welch Diagnostics" on page 170 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. The table is not displayed by default and can be requested by specifying DIAGNOSTICS=HEIDEL.

Raftery-Lewis Diagnostics

The "Raftery-Lewis Diagnostics" table (ODS table name Raftery) lists the result of the Raftery-Lewis diagnostic test. See the section "Raftery and Lewis Diagnostics" on page 172 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for more details. The table is not displayed by default and can be requested by specifying DIAGNOSTICS=RAFTERY.

Summary and Interval Statistics for Prediction

The "Posterior Summaries and Intervals for Prediction" table (ODS table name PredSumInt) contains a summary of basic point and interval statistics for each prediction. The table lists the number of posterior samples, the posterior mean and standard deviation estimates, and the 95% HPD interval estimates. This table is displayed by default if any PREDDIST statement is used in the program.

Summary Statistics for Prediction

The "Posterior Summaries for Prediction" table (ODS table name PredSummaries) contains basic statistics for each prediction. The table lists the number of posterior samples, the posterior mean and standard deviation estimates, and the percentile estimates. This table is not displayed by default and can be requested by specifying the option STATISTIC=SUMMARY.

Interval Statistics for Prediction

The "Posterior Intervals for Prediction" table (ODS table name PredIntervals) contains the equal-tail and highest posterior density (HPD) interval estimates for each prediction. The default α value is 0.05, and you can change it to other levels by using the STATISTICS option in a PREDDIST statement, or the STATISTICS= option in the PROC MCMC statement if the option is not specified in a statement. This table is not displayed by default and can be requested by specifying the option STATISTIC=INTERVAL.

Optimization Related ODS Tables

PROC MCMC can perform optimization on the joint posterior distribution. This is requested by the PROPCOV= option. The most commonly used optimization method is the quasi-Newton method: PROPCOV=QUANEW(ITPRINT). The ITPRINT option displays the ODS tables, listed as follows:

Input Options

The "Input Options" table (ODS table name InputOptions) lists optimization options used in the procedure.

Optimization Start

The "Optimization Start" table (ODS table name ProblemDescription) shows the initial state of the optimization.

Iteration History

The "Iteration History" table (ODS table name IterHist) shows iteration history of the optimization.

Optimization Results

The "Optimization Results" table (ODS table name IterStop) shows the results of the optimization, includes information about the number of function calls, and the optimized objective function, which is the joint log posterior density.

Convergence Status

The "Convergence Status" table (ODS table name ConvergenceStatus) shows whether the convergence criterion is satisfied.

Parameters Value After Optimization Table

The "Parameter Values After Optimization" table (ODS table name OptiEstimates) lists the parameter values that maximize the joint log posterior. These are the maximum a posteriori point estimates, and they are used to start the Markov chain.

Covariance Matrix After Optimization Table

The "Proposal Covariance" table (ODS table name OptiCov) lists covariance matrices for each block parameter by using quadrature approximation at the posterior mode. These covariance matrices are used in the proposal distribution.

ODS Table Names

PROC MCMC assigns a name to each table it creates. You can use these names to refer to the table when you use the Output Delivery System (ODS) to select tables and create output data sets. These names are listed in Table 80.53. For more information about ODS, see Chapter 24, "Statistical Graphics Using ODS."

Table 80.53 ODS Tables Produced in PROC MCMC

ODS Table Name	Description	Statement or Option
AutoCorr	Autocorrelation statistics for each parameter	DIAGNOSTICS=AUTOCORR
BurnInHistory	History of burn-in phase sampling	MCHISTORY=BRIEF | DETAILED
ConvergenceStatus	Optimization convergence status	PROPCOV=method(ITPRINT)
Corr	Correlation matrix of the posterior samples	STATS=CORR
Cov	Covariance matrix of the posterior samples	STATS=COV
DIC	Deviance information criterion	DIC
ESS	Effective sample size for each parameter	Default
MCSE	Monte Carlo standard error for each parameter	DIAGNOSTICS=MCSE
Geweke	Geweke diagnostics for each parameter	DIAGNOSTICS=GEWEKE
Heidelberger	Heidelberger-Welch diagnostics for each parameter	DIAGNOSTICS=HEIDEL
InputOptions	Optimization input table	PROPCOV=method(ITPRINT)
IterHist	Optimization iteration history	PROPCOV=method(ITPRINT)
IterStop	Optimization results table	PROPCOV=method(ITPRINT)
MissDataInfo	Response variable, number of missing observations, missing observation indices, and sampling algorithm	Default with sampling of missing values
NObs	Number of observations	Default
OptiEstimates	Parameter values after either optimization	PROPCOV=method(ITPRINT)
OptiCov	Covariance used in proposal distribution after optimization	PROPCOV=method(ITPRINT)
Parameters	Summary of the PARMS, BLOCKING, PRIOR, sampling method, and initial value specification	Default
ParametersInit	Parameter values after the tuning phase	INIT=PINIT
PosteriorSample	Posterior samples for each parameter	(For ODS output data set only)
PostIntervals	Equal-tail and HPD intervals for each parameter	STATISTICS=INTERVAL
PostSumInt	Basic posterior statistics for each parameter, including sample size, mean, standard deviation, and HPD intervals	Default
PostSummaries	Basic posterior statistics for each parameter, including sample size, mean, standard deviation, and percentiles	STATISTICS=SUMMARY

Table 80.53 *continued*

ODS Table Name	Description	Statement or Option
PredIntervals	Equal-tail and HPD intervals for each prediction	STATISTICS=INTERVAL in PREDDIST statement
PredSumInt	Basic posterior statistics for each prediction, including sample size, mean, standard deviation, and HPD intervals	Default with any PREDDIST statement
PredSummaries	Basic posterior statistics for each prediction	STATISTICS=SUMMARY in PREDDIST statement
ProblemDescription	Optimization table	PROPCOV=method(ITPRINT)
REObsInfo	Random effect, subject values, number of observations in each unique subject value, and corresponding observation indices	REOBSINFO
REParameters	Random effect, sampling method, subject variable, number of subjects, unique values of the subject variable, and prior distribution of the random effect	Default with any RANDOM statement
Raftery	Raftery-Lewis diagnostics for each parameter	DIAGNOSTICS=RAFTERY
SamplingHistory	History of main phase sampling	MCHISTORY=BRIEF | DETAILED
TuneCov	Proposal covariance matrix (for continuous parameters) after the tuning phase	INIT=PINIT
TuneP	Proposal probability vector (for discrete parameters) after the tuning phase	INIT=PINIT and DISCRETE=GEO
TuningHistory	History of proposal distribution tuning	MCHISTORY=BRIEF | DETAILED

ODS Graphics

Statistical procedures use ODS Graphics to create graphs as part of their output. ODS Graphics is described in detail in Chapter 24, "Statistical Graphics Using ODS."

Before you create graphs, ODS Graphics must be enabled (for example, by specifying the ODS GRAPHICS ON statement). For more information about enabling and disabling ODS Graphics, see the section "Enabling and Disabling ODS Graphics" on page 687 in Chapter 24, "Statistical Graphics Using ODS."

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" on page 686 in Chapter 24, "Statistical Graphics Using ODS."

You can reference every graph produced through ODS Graphics with a name. The names of the graphs that PROC MCMC generates are listed in Table 80.54.

Table 80.54 Graphs Produced by PROC MCMC

ODS Graph Name	Plot Description	Statement and Option
ADPanel	Autocorrelation function and density panel	PLOTS=(AUTOCORR DENSITY)
AutocorrPanel	Autocorrelation function panel	PLOTS=AUTOCORR
AutocorrPlot	Autocorrelation function plot	PLOTS(UNPACK)=AUTOCORR
DensityPanel	Density panel	PLOTS=DENSITY
DensityPlot	Density plot	PLOTS(UNPACK)=DENSITY
TAPanel	Trace and autocorrelation function panel	PLOTS=(TRACE AUTOCORR)
TADPanel	Trace, density, and autocorrelation function panel	PLOTS=(TRACE AUTOCORR DENSITY)
TDPanel	Trace and density panel	PLOTS=(TRACE DENSITY)
TracePanel	Trace panel	PLOTS=TRACE
TracePlot	Trace plot	PLOTS(UNPACK)=TRACE

Examples: MCMC Procedure

The examples in this chapter are available in the GitHub repository located at `https://github.com/sassoftware/doc-supplement-statug`.

Example 80.1: Simulating Samples From a Known Density

(View the complete code for this example (mcmcex1.sas) in the example repository.)

This example illustrates how you can obtain random samples from a known function. The target distributions are the normal distribution (a standard distribution) and a mixture of the normal distributions (a nonstandard distribution). For more information, see the sections "Standard Distributions" on page 6275 and "Specifying a New Distribution" on page 6290). This example also shows how you can use PROC MCMC to estimate an integral (area under a curve). Monte Carlo simulation is data-independent; hence, you do not need an input data set from which to draw random samples from the desired distribution.

Sampling from a Normal Density

When you run a simulation without an input data set, the posterior distribution is the same as the prior distribution. Hence, if you want to generate samples from a distribution, you declare the distribution in the PRIOR statement and set the likelihood function to a constant. Although there is no contribution from any data set variable to the likelihood calculation, you still must specify a data set and the MODEL statement needs a distribution. You can input an empty data set and use the GENERAL function to provide a flat likelihood. The following statements generate 10,000 samples from a standard normal distribution:

```
title 'Simulating Samples from a Normal Density';
data x;
run;

ods graphics on;
proc mcmc data=x outpost=simout seed=23 nmc=10000 diagnostics=none;
   ods exclude nobs;
   parm alpha 0;
   prior alpha ~ normal(0, sd=1);
   model general(0);
run;
```

The ODS GRAPHICS ON statement enables ODS Graphics. The PROC MCMC statement specifies the input and output data sets, a random number seed, and the size of the simulation sample. The STATISTICS= option displays only the summary and interval statistics. The ODS EXCLUDE statement excludes the display of the NObs table. PROC MCMC draws independent samples from the normal distribution directly (see Output 80.1.1). Therefore, the simulation does not require any tuning, and PROC MCMC omits the default burn-in phrase.

Output 80.1.1 Parameters Information

Simulating Samples from a Normal Density

The MCMC Procedure

Parameters

Block	Parameter	Sampling Method	Initial Value	Prior Distribution
1	alpha	Direct	0	normal(0, sd=1)

The summary statistics (Output 80.1.2) are what you would expect from a standard normal distribution.

Output 80.1.2 MCMC Summary and Interval Statistics from a Normal Target Distribution

Simulating Samples from a Normal Density

The MCMC Procedure

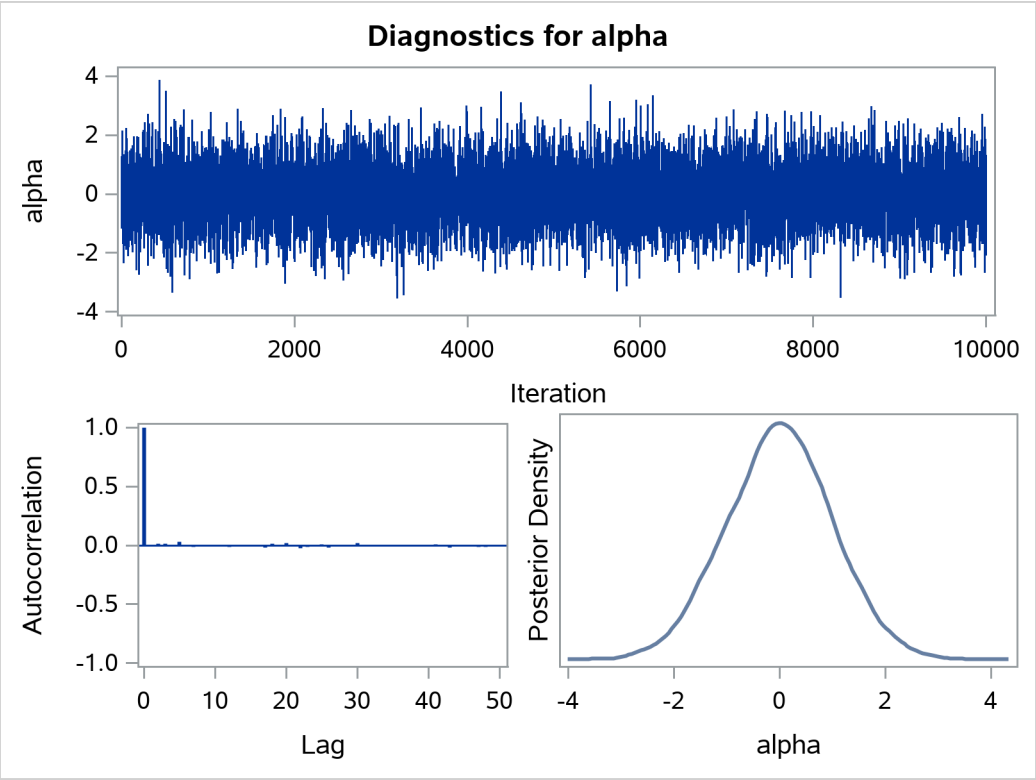
Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
alpha	10000	0.00195	0.9949	-1.9664	1.9302

The trace plot (Output 80.1.3) shows perfect mixing with no autocorrelation in the lag plot. This is expected

because these are independent draws.

Output 80.1.3 Diagnostics Plots for α



You can overlay the estimated kernel density with the true density to visually compare the densities, as displayed in Output 80.1.4. To create the kernel comparison plot, you first call PROC KDE (see Chapter 73, "The KDE Procedure") to obtain a kernel density estimate of the posterior density on alpha. Then you evaluate a grid of alpha values on a normal density. The following statements evaluate kernel density and compute the corresponding normal density:

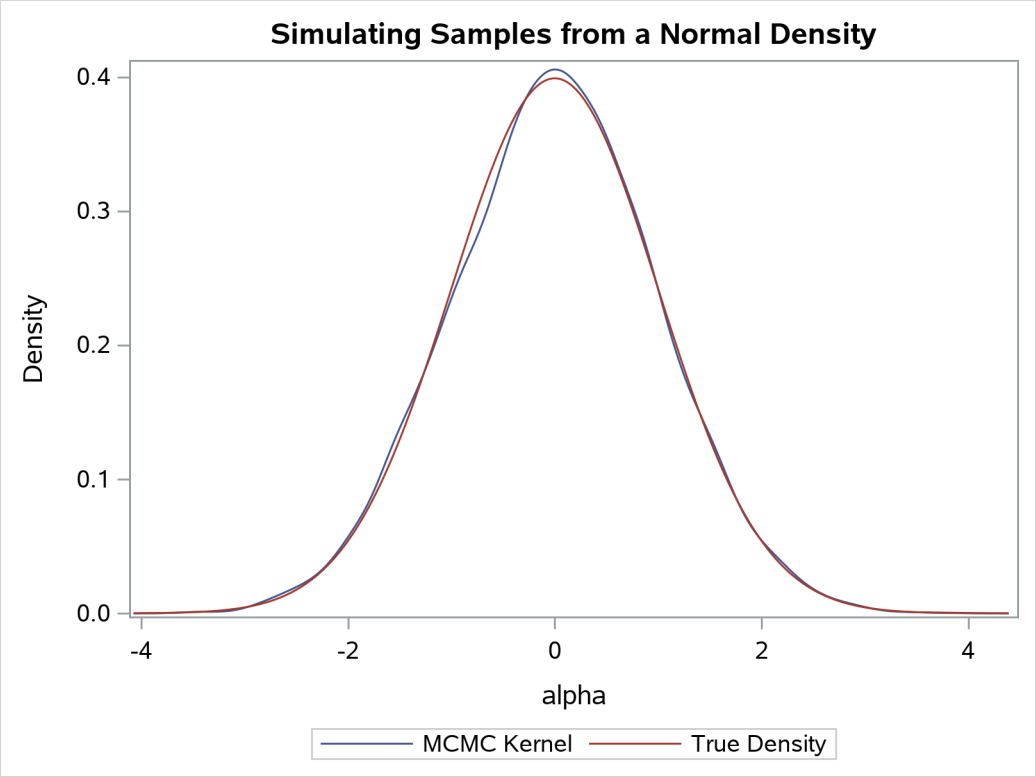
```
proc kde data=simout;
   ods exclude inputs controls;
   univar alpha /out=sample;
run;

data den;
   set sample;
   alpha = value;
   true  = pdf('normal', alpha, 0, 1);
   keep alpha density true;
run;
```

Next you plot the two curves on top of each other by using PROC SGPLOT (see Chapter 24, "Statistical Graphics Using ODS") as follows:

```
proc sgplot data=den;
   yaxis label="Density";
   series y=density x=alpha / legendlabel = "MCMC Kernel";
   series y=true x=alpha / legendlabel = "True Density";
   discretelegend;
run;
```

Output 80.1.4 shows the result. You can see that the kernel estimate and the true density are very similar to each other.

Output 80.1.4 Estimated Density versus the True Density



Density Visualization Macro

In programs that do not involve any data set variables, PROC MCMC samples directly from the (joint) prior distributions of the parameters. The modification makes the sampling from a known distribution more efficient and more precise. For example, you can write simple programs, such as the following macro, to understand different aspects of a prior distribution of interest, such as its moments, intervals, shape, spread, and so on:

```
%macro density(dist=, seed=0);
   %let savenote = %sysfunc(getoption(notes));
   options nonotes;
   title "&dist distribution.";
   data _a;
   run;
```

```
   ods select densitypanel postsumint;
   proc mcmc data=_a nmc=10000 diag=none nologdist
      plots=density seed=&seed;
      parms alpha;
      prior alpha ~ &dist;
      model general(0);
   run;

   proc datasets nolist;
      delete _a;
   run;
   options &savenote;
%mend;

%density(dist=beta(4, 12), seed=1);
```

The macro %density creates an empty data set, invokes PROC MCMC, draws 10,000 samples from a beta(4, 12) distribution, displays summary and interval statistics, and generates a kernel density plot. Summary and interval statistics from the beta distribution are displayed in Output 80.1.5.

Output 80.1.5 Beta Distribution Statistics

beta(4, 12) distribution.

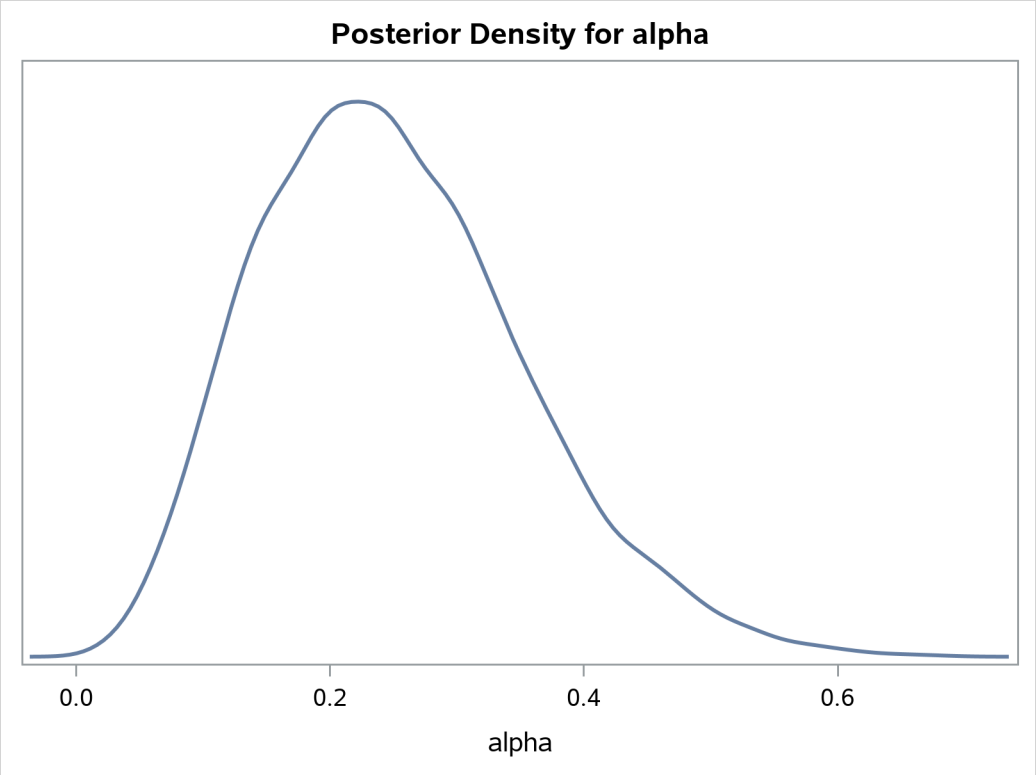
The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
alpha	10000	0.2494	0.1039	0.0657	0.4590

The distribution is displayed in Output 80.1.6.

Output 80.1.6 Density Plot



Calculation of Integrals

One advantage of MCMC methods is to estimate any integral under the curve of a target distribution. This can be done fairly easily using the MCMC procedure. Suppose you are interested in estimating the following cumulative probability:

$$\int_{\alpha=0}^{1.3} \phi(\alpha|0,1)d\alpha$$

To estimate this integral, PROC MCMC draws samples from the distribution and counts the portion of the simulated values that fall within the desired range of [0, 1.3]. This becomes a Monte Carlo estimate of the integral. The following statements simulate samples from a standard normal distribution and estimate the integral:

```
proc mcmc data=x outpost=simout seed=23 nmc=10000 nologdist
   monitor=(int) diagnostics=none;
   ods select postsumint;
   parm alpha 0;
   prior alpha ~ normal(0, sd=1);
   int = (0 <= alpha <= 1.3);
   model general(0);
run;
```

The ODS SELECT statement displays the posterior summary statistics table. The MONITOR= option outputs analysis on the variable int (the integral estimate). The STATISTICS= option computes the summary statistics.

The NOLOGDIST option omits the calculation of the log of the prior distribution at each iteration, shortening the simulation time[5]. The INT assignment statement sets int to be 1 if the simulated alpha value falls between 0 and 1.3, and 0 otherwise. PROC MCMC supports the usage of the IF-ELSE logical control if you need to account for more complex conditions. Output 80.1.7 displays the estimated integral value:

Output 80.1.7 Monte Carlo Integral from a Normal Distribution

beta(4, 12) distribution.

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval
int	10000	0.4079	0.4915	0 1.0000

In this simulation, 4079 samples fall between 0 and 1.3, making the expected probability 0.4079. In this example, you can verify the actual cumulative probability by calling the CDF function in the DATA step:

```
data _null_;
   int = cdf("normal", 1.3, 0, 1) - cdf("normal", 0, 0, 1);
   put int=;
run;
```

The value is 0.4032.

Sampling from a Mixture of Normal Densities

Suppose you are interested in generating samples from a three-component mixture of normal distributions, with the density specified as follows:

$$p(\alpha) = 0.3 \cdot \phi(-3, \sigma = 2) + 0.4 \cdot \phi(2, \sigma = 1) + 0.3 \cdot \phi(10, \sigma = 4)$$

You can either specify the distribution directly or use a latent variable approach to generate samples from the normal mixture.

To specify the normal mixture density directly in PROC MCMC, you need to construct the function because the normal mixture distribution is not one of the standard distributions that PROC MCMC supports. The following statements generate random samples from the normal mixture density:

```
title 'Simulating Samples from a Mixture of Normal Densities';
data x;
run;

proc mcmc data=x outpost=simout seed=1234 nmc=30000;
   ods select TADpanel;
   parm alpha 0.3;
   lp = logpdf('normalmix', alpha, 3, 0.3, 0.4, 0.3, -3, 2, 10, 2, 1, 4);
   prior alpha  ~ general(lp);
```

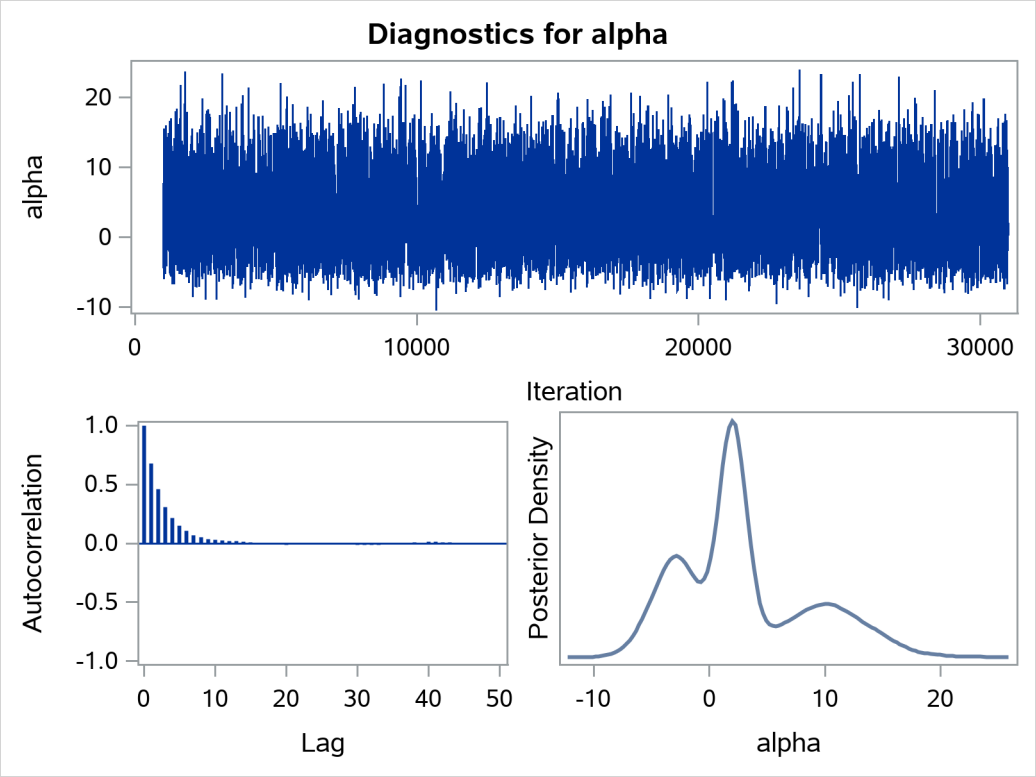
[5] In this example, the NOLOGDIST option saves only a fraction of the time. But in more complex simulation schemes that involve a larger number of distributions and parameters, the time reduction could be significant.

```
   model general(0);
run;
```

The ODS SELECT statement displays the diagnostic plots. All other tables, such as the NObs tables, are excluded. The PROC MCMC statement uses the input data set X, saves output to the Simout data set, sets a random number seed, and draws 30,000 samples.

The LP assignment statement evaluates the log density of alpha at the mixture density, using the SAS function LOGPDF. The number 3 after alpha in the LOGPDF function indicates that the density is a three-component normal mixture. The following three numbers, 0.3, 0.4, and 0.3, are the weights in the mixture; –3, 2, and 10 are the means; 2, 1, and 4 are the standard deviations. The PRIOR statement assigns this log density function to alpha as its prior. Note that the GENERAL function interprets the density on the log scale, and not the original scale–you must use the LOGPDF function, not the PDF function. Output 80.1.8 displays the results. The kernel density clearly shows three modes.

Output 80.1.8 Plots of Posterior Samples from a Mixture Normal Distribution



Alternatively, the normal mixture distribution can also decomposed into a marginal distribution for the component (call it Z) and a conditional model of the response variable Y given Z, as

$$\begin{aligned} z &\sim \text{categorical}(p_1, p_2, \ldots, p_K) \\ y|z &\sim \text{normal}(\mu_z, \sigma_z^2) \end{aligned}$$

where K is the total number of mixture components, z is the component indicator, and μ_z and σ_z^2 are the model parameters for the zth component.

PROC MCMC supports a categorical distribution that can be used to model the discrete random variable for components. You can use PRIOR statements to specify a normal mixture distribution and generate samples accordingly:

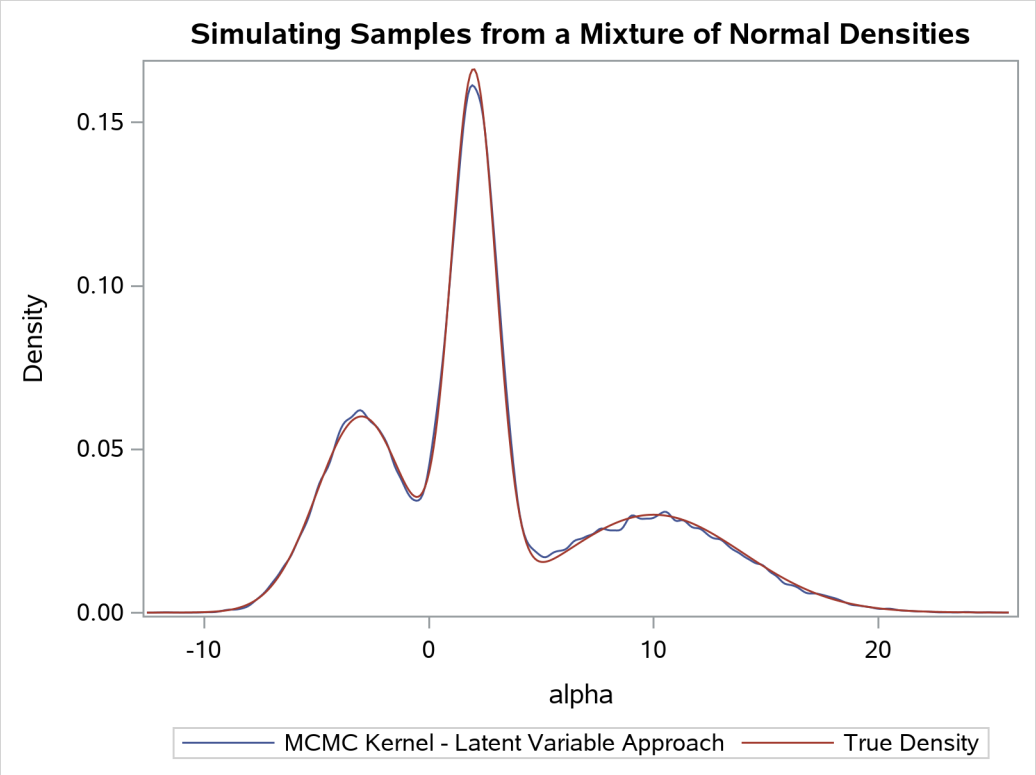
```
proc mcmc data=x outpost=simout_m seed=1234 nmc=30000;
   array p[3] (0.3 0.4 0.3);
   array mu[3] (-3 2 10);
   array sd[3] (2 1 4);
   parm z alpha;
   prior z ~ table(p);
   prior alpha ~ normal(mu[z], sd=sd[z]);
   model general(0);
run;
```

The ARRAY statements define one array p for the mixture weights, one array mu for the means of the normal distributions, and one array sd for the corresponding standard deviations. The PRIOR statements specify a categorical prior on the parameter z and a conditional normal prior on alpha. The mean and standard deviation of the alpha parameter depend on the component indicator z. No output is created.

You can use the following set of statements, which are similar to the previous example, to overlay the estimated kernel density with the true density. The comparison is shown in Output 80.1.9.

```
proc kde data=simout_m;
   ods exclude inputs controls;
   univar alpha /out=sample;
run;

data den;
   set sample;
   alpha = value;
   true  = pdf('normalmix', alpha, 3, 0.3, 0.4, 0.3, -3, 2, 10, 2, 1, 4);
   keep alpha density true;
run;

proc sgplot data=den;
   yaxis label="Density";
   series y=density x=alpha /
      legendlabel = "MCMC Kernel - Latent Variable Approach";
   series y=true x=alpha / legendlabel = "True Density";
   discretelegend;
run;
```

Output 80.1.9 Estimated Density (Latent Variable Approach) versus the True Density



Example 80.2: Box-Cox Transformation

(View the complete code for this example (mcmcex2.sas) in the example repository.)

Box-Cox transformations (Box and Cox 1964) are often used to find a power transformation of a dependent variable to ensure the normality assumption in a linear regression model. This example illustrates how you can use PROC MCMC to estimate a Box-Cox transformation for a linear regression model. Two different priors on the transformation parameter λ are considered: a continuous prior and a discrete prior. You can estimate the probability of λ being 0 with a discrete prior but not with a continuous prior. The IF-ELSE statements are demonstrated in the example.

Using a Continuous Prior on λ

The following statements create a SAS data set with measurements of y (the response variable) and x (a single dependent variable):

```
title 'Box-Cox Transformation, with a Continuous Prior on Lambda';
data boxcox;
   input y x @@;
   datalines;
10.0  3.0  72.6  8.3  59.7  8.1  20.1  4.8  90.1  9.8   1.1  0.9
78.2  8.5  87.4  9.0   9.5  3.4   0.1  1.4   0.1  1.1  42.5  5.1
57.0  7.5   9.9  1.9   0.5  1.0 121.1  9.9  37.5  5.9  49.5  6.7
```

```
   ... more lines ...

 2.6  1.8  58.6  7.9  81.2  8.1  37.2  6.9
;
```

The Box-Cox transformation of y takes on the form of:

$$y(\lambda) = \begin{cases} \frac{y^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0; \\ \log(y) & \text{if } \lambda = 0. \end{cases}$$

The transformed response $y(\lambda)$ is assumed to be normally distributed:

$$y_i(\lambda) \sim \text{normal}(\beta_0 + \beta_1 x_i, \sigma^2)$$

The likelihood with respect to the original response y_i is as follows:

$$p(y_i|\lambda, \beta, \sigma^2, x_i) \propto \phi(y_i|\beta_0 + \beta_1 x_i, \sigma^2) \cdot J(\lambda, y_i)$$

where $J(\lambda, y_i)$ is the Jacobian:

$$J(\lambda, y) = \begin{cases} y_i^{\lambda-1} & \text{if } \lambda \neq 0; \\ 1/y_i & \text{if } \lambda = 0. \end{cases}$$

And on the log-scale, the Jacobian becomes:

$$\log(J(\lambda, y)) = \begin{cases} (\lambda - 1) \cdot \log(y_i) & \text{if } \lambda \neq 0; \\ -\log(y_i) & \text{if } \lambda = 0. \end{cases}$$

There are four model parameters: $\lambda, \boldsymbol{\beta} = \{\beta_0, \beta_1\}$, and σ^2. You can considering using a flat prior on $\boldsymbol{\beta}$ and a gamma prior on σ^2.

To consider only power transformations ($\lambda \neq 0$), you can use a continuous prior (for example, a uniform prior from –2 to 2) on λ. One issue with using a continuous prior is that you cannot estimate the probability of $\lambda = 0$. To do so, you need to consider a discrete prior that places positive probability mass on the point 0. See "Modeling $\lambda = 0$" on page 6376.

The following statements fit a Box-Cox transformation model:

```
ods graphics on;
proc mcmc data=boxcox nmc=50000 propcov=quanew seed=12567
          monitor=(lda);
   ods select PostSumInt TADpanel;
   parms beta0 0  beta1 0  lda 1 s2 1;

   beginnodata;
   prior beta: ~ general(0);
   prior s2 ~ gamma(shape=3, scale=2);
   prior lda ~ unif(-2,2);
   sd = sqrt(s2);
   endnodata;

   ys = (y**lda-1)/lda;
   mu = beta0+beta1*x;
   ll = (lda-1)*log(y)+lpdfnorm(ys, mu, sd);
   model general(ll);
run;
```

The PROPCOV= option initializes the Markov chain at the posterior mode and uses the estimated inverse Hessian matrix as the initial proposal covariance matrix. The MONITOR= option selects λ as the variable to report. The ODS SELECT statement displays the summary statistics table, the interval statistics table, and the diagnostic plots.

The PARMS statement puts all four parameters, β_0, β_1, λ, and σ^2, in a single block and assigns initial values to each of them. Three PRIOR statements specify previously stated prior distributions for these parameters. The assignment to sd transforms a variance to a standard deviation. It is better to place the transformation inside the BEGINNODATA and ENDNODATA statements to save computational time.

The assignment to the symbol ys evaluates the Box-Cox transformation of y, where mu is the regression mean and ll is the log likelihood of the transformed variable ys. Note that the log of the Jacobian term is included in the calculation of ll.

Summary statistics and interval statistics for lda are listed in Output 80.2.1.

Output 80.2.1 Box-Cox Transformation

Box-Cox Transformation, with a Continuous Prior on Lambda

The MCMC Procedure

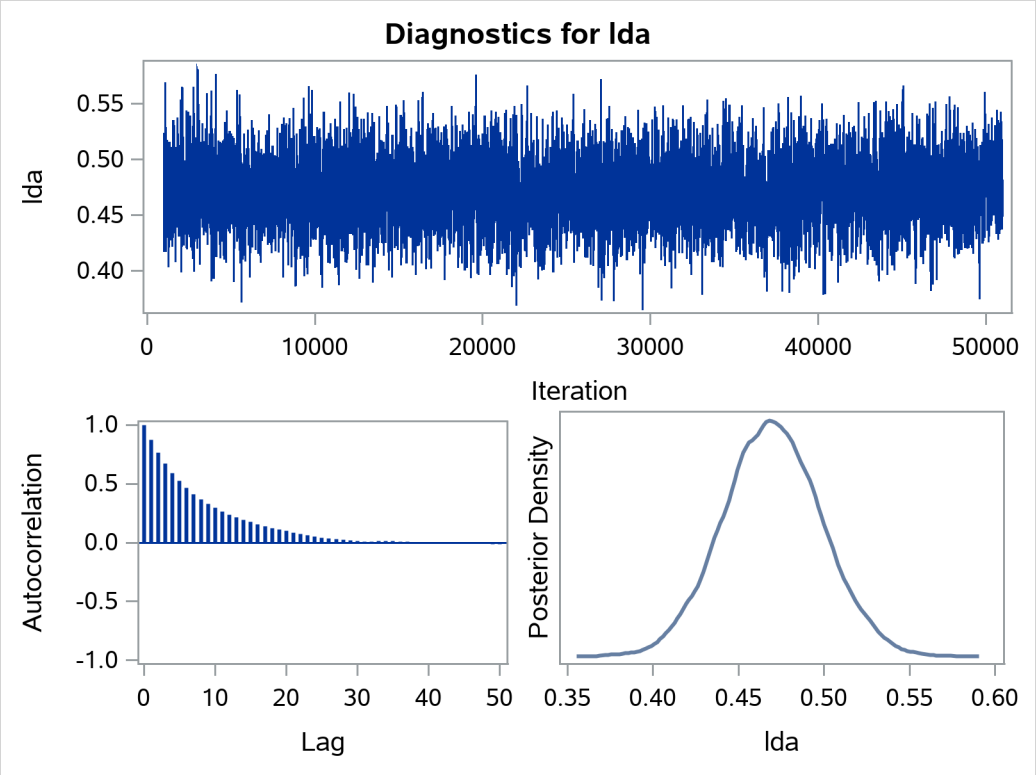
Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
lda	50000	0.4698	0.0288	0.4119	0.5249

The posterior mean of λ is 0.47, with a 95% equal-tail interval of $[0.42, 0.53]$ and a similar HPD interval. The preferred power transformation would be 0.5 (rounding λ up to the square root transformation).

Output 80.2.2 shows diagnostics plots for lda. The chain appears to converge, and you can proceed to make inferences. The density plot shows that the posterior density is relatively symmetric around its mean estimate.

Output 80.2.2 Diagnostic Plots for λ

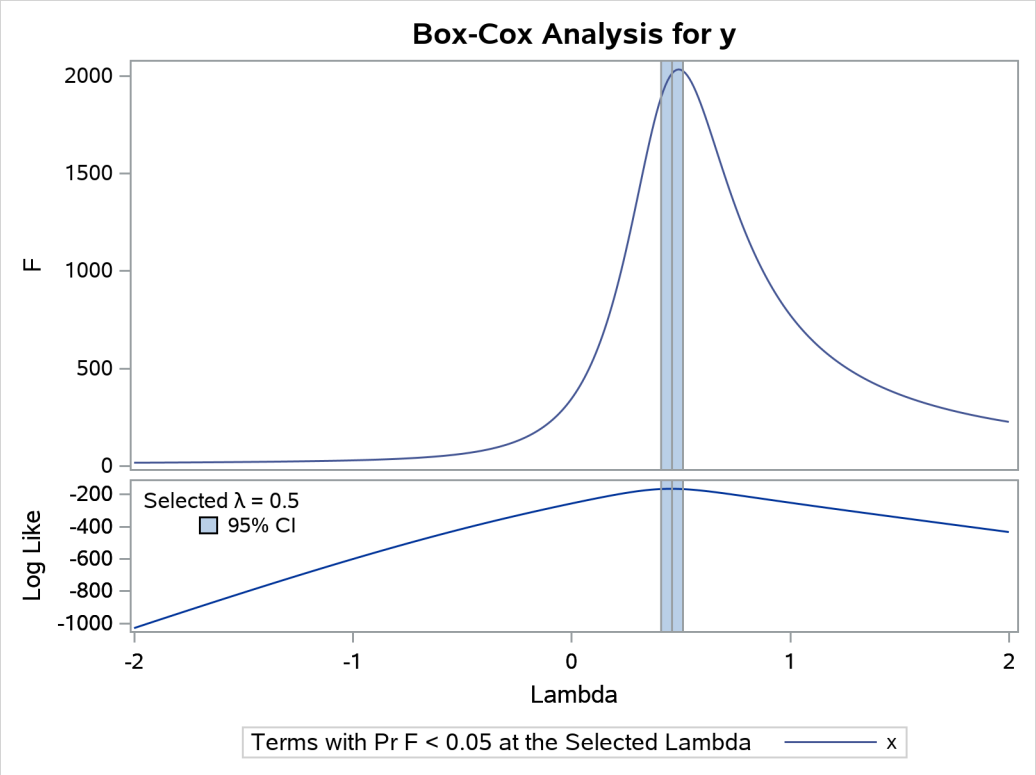


To verify the results, you can use PROC TRANSREG (see Chapter 126, "The TRANSREG Procedure") to find the estimate of λ.

```
proc transreg data=boxcox details pbo;
   ods output boxcox = bc;
   model boxcox(y / convenient lambda=-2 to 2 by 0.01) = identity(x);
   output out=trans;
run;
```

Output from PROC TRANSREG is shown in Output 80.2.5 and Output 80.2.4. PROC TRANSREG produces a similar point estimate of $\lambda = 0.46$, and the 95% confidence interval is shown in Output 80.2.5.

Output 80.2.3 Box-Cox Transformation Using PROC TRANSREG



Output 80.2.4 Estimates Reported by PROC TRANSREG

Box-Cox Transformation, with a Continuous Prior on Lambda

The TRANSREG Procedure

Model Statement Specification Details

Type	DF	Variable	Description	Value
Dep	1	BoxCox(y)	Lambda Used	0.5
			Lambda	0.46
			Log Likelihood	-167.0
			Conv. Lambda	0.5
			Conv. Lambda LL	-168.3
			CI Limit	-169.0
			Alpha	0.05
			Options	Convenient Lambda Used
Ind	1	Identity(x)	DF	1

The ODS data set Bc contains the 95% confidence interval estimates produced by PROC TRANSREG. This ODS table is rather large, and you want to see only the relevant portion. The following statements generate the part of the table that is important and display Output 80.2.5:

```
proc print noobs label data=bc(drop=rmse);
   title2 'Confidence Interval';
   where ci ne ' ' or abs(lambda - round(lambda, 0.5)) < 1e-6;
   label convenient = '00'x ci = '00'x;
run;
```

The estimated 90% confidence interval is $[0.41, 0.51]$, which is very close to the reported Bayesian credible intervals. The resemblance of the intervals is probably due to the noninformative prior that you used in this analysis.

Output 80.2.5 Estimated Confidence Interval on λ

Box-Cox Transformation, with a Continuous Prior on Lambda
Confidence Interval

Dependent	Lambda		R-Square	Log Likelihood	
BoxCox(y)	-2.00		0.14	-1030.56	
BoxCox(y)	-1.50		0.17	-810.50	
BoxCox(y)	-1.00		0.22	-602.53	
BoxCox(y)	-0.50		0.39	-415.56	
BoxCox(y)	0.00		0.78	-257.92	
BoxCox(y)	0.41		0.95	-168.40	*
BoxCox(y)	0.42		0.95	-167.86	*
BoxCox(y)	0.43		0.95	-167.46	*
BoxCox(y)	0.44		0.95	-167.19	*
BoxCox(y)	0.45		0.95	-167.05	*
BoxCox(y)	0.46		0.95	-167.04	<
BoxCox(y)	0.47		0.95	-167.16	*
BoxCox(y)	0.48		0.95	-167.41	*
BoxCox(y)	0.49		0.95	-167.79	*
BoxCox(y)	0.50	+	0.95	-168.28	*
BoxCox(y)	0.51		0.95	-168.89	*
BoxCox(y)	1.00		0.89	-253.09	
BoxCox(y)	1.50		0.79	-345.35	
BoxCox(y)	2.00		0.70	-435.01	

Modeling $\lambda = 0$

With a continuous prior on λ, you can get only a continuous posterior distribution, and this makes the probability of $\Pr(\lambda = 0|\text{data})$ equal to 0 by definition. To consider $\lambda = 0$ as a viable solution to the Box-Cox transformation, you need to use a discrete prior that places some probability mass on the point 0 and allows for a meaningful posterior estimate of $\Pr(\lambda = 0|\text{data})$.

This example uses a simulation study where the data are generated from an exponential likelihood. The simulation implies that the correct transformation should be the logarithm and λ should be 0. Consider the following exponential model:

$$y = \exp(x + \epsilon),$$

where $\epsilon \sim \text{normal}(0, 1)$. The transformed data can be fitted with a linear model:

$$\log(y) = x + \epsilon$$

The following statements generate a SAS data set with a gridded x and corresponding y:

```
title 'Box-Cox Transformation, Modeling Lambda = 0';
data boxcox;
   do x = 1 to 8 by 0.025;
      ly = x + normal(7);
      y = exp(ly);
      output;
   end;
run;
```

The log-likelihood function, after taking the Jacobian into consideration, is as follows:

$$\log p(y_i|\lambda, x_i) = \begin{cases} (\lambda-1)\log(y_i) - \frac{1}{2}\left(\log\sigma^2 + \frac{\left((y_i^\lambda - 1)/\lambda - x_i\right)^2}{\sigma^2}\right) + C_1 & \text{if } \lambda \neq 0; \\ -\log(y_i) - \frac{1}{2}\left(\log\sigma^2 + \frac{(\log(y_i)-x_i)^2}{\sigma^2}\right) + C_2 & \text{if } \lambda = 0. \end{cases}$$

where C_1 and C_2 are two constants.

You can use the function DGENERAL to place a discrete prior on λ. The function is similar to the function GENERAL, except that it indicates a discrete distribution. For example, you can specify a discrete uniform prior from –2 to 2 using

```
prior lda ~ dgeneral(1, lower=-2, upper=2);
```

This places equal probability mass on five points, –2, –1, 0, 1, and 2. This prior might not work well here because the grid is too coarse. To consider smaller values of λ, you can sample a parameter that takes a wider range of integer values and transform it back to the λ space. For example, set alpha as your model parameter and give it a discrete uniform prior from –200 to 200. Then define λ as alpha/100 so λ can take values between –2 and 2 but on a finer grid.

The following statements fit a Box-Cox transformation by using a discrete prior on λ:

```
proc mcmc data=boxcox outpost=simout nmc=50000 seed=12567
          monitor=(lda);
   ods select PostSumInt;
   parms s2 1 alpha 10;
   beginnodata;
   prior s2 ~ gamma(shape=3, scale=2);
   if alpha=0 then lp = log(2);
      else lp = log(1);
   prior alpha ~ dgeneral(lp, lower=-200, upper=200);
   lda = alpha * 0.01;
   sd = sqrt(s2);
   endnodata;
   if alpha=0 then
      ll = -ly+lpdfnorm(ly, x, sd);
   else do;
      ys = (y**lda - 1)/lda;
      ll = (lda-1)*ly+lpdfnorm(ys, x, sd);
```

```
   end;
   model general(ll);
run;
```

There are two parameters, s2 and alpha, in the model. They are placed in a single PARMS statement so that they are sampled in the same block.

The parameter s2 takes a gamma distribution, and alpha takes a discrete prior. The IF-ELSE statements state that alpha takes twice as much prior density when it is 0 than otherwise. Note that on the original scale, $\Pr(\text{alpha} = 0) = 2 \cdot \Pr(\text{alpha} \neq 0)$. Translating that to the log scale, the densities become $\log(2)$ and $\log(1)$, respectively. The LDA assignment statement transforms alpha to the parameter of interest: lda takes values between –2 and 2. You can model lda on a even smaller scale by dividing alpha by a larger constant. However, an increment of 0.01 in the Box-Cox transformation is usually sufficient. The SD assignment statement calculates the square root of the variance term.

The log-likelihood function uses another set of IF-ELSE statements, separating the case of $\lambda = 0$ from the others. The formulas are stated previously. The output of the program is shown in Output 80.2.6.

Output 80.2.6 Box-Cox Transformation

Box-Cox Transformation, Modeling Lambda = 0

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
lda	50000	-0.00001	0.00199	0	0

From the summary statistics table, you see that the point estimate for λ is 0 and both of the 95% equal-tail and HPD credible intervals are 0. This strongly suggests that $\lambda = 0$ is the best estimate for this problem. In addition, you can also count the frequency of λ among posterior samples to get a more precise estimate on the posterior probability of λ being 0.

The following statements use PROC FREQ to produce Output 80.2.7 and Output 80.2.8:

```
proc freq data=simout;
   ods select onewayfreqs freqplot;
   tables lda /nocum plot=freqplot(scale=percent);
run;
ods graphics off;
```

Output 80.2.7 shows the frequency count table. An estimate of $\Pr(\lambda = 0|\text{data})$ is 96%. The conclusion is that the log transformation should be the appropriate transformation used here, which agrees with the simulation setup. Output 80.2.8 shows the histogram of λ.

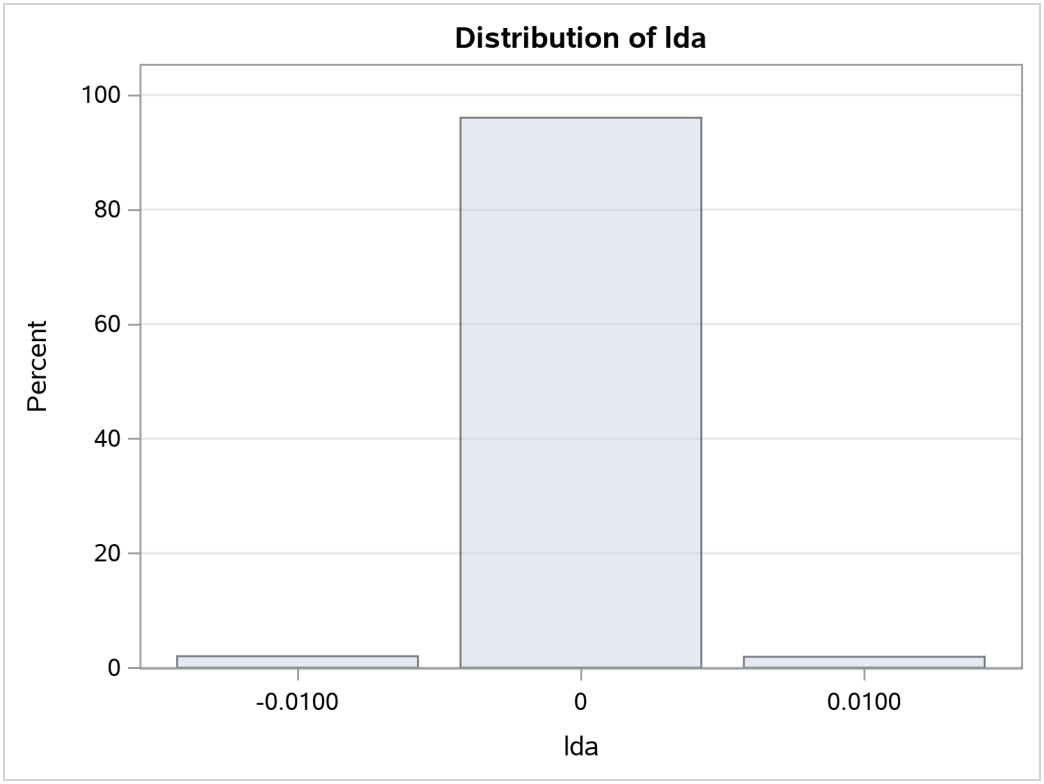
Output 80.2.7 Frequency Counts of λ

Box-Cox Transformation, Modeling Lambda = 0

The FREQ Procedure

lda	Frequency	Percent
-0.0100	1011	2.02
0	48029	96.06
0.0100	960	1.92

Output 80.2.8 Histogram of λ



Example 80.3: Logistic Regression Model with a Diffuse Prior

(View the complete code for this example (mcmcex3.sas) in the example repository.)

This example illustrates how to fit a logistic regression model with a diffuse prior in PROC MCMC. You can also use the BAYES statement in PROC GENMOD. See Chapter 51, "The GENMOD Procedure."

The following statements create a SAS data set with measurements of the number of deaths, y, among n beetles that have been exposed to an environmental contaminant x:

```
title 'Logistic Regression Model with a Diffuse Prior';
data beetles;
   input n y x @@;
   datalines;
6  0  25.7   8  2  35.9   5  2  32.9   7  7  50.4   6  0  28.3
7  2  32.3   5  1  33.2   8  3  40.9   6  0  36.5   6  1  36.5
6  6  49.6   6  3  39.8   6  4  43.6   6  1  34.1   7  1  37.4
8  2  35.2   6  6  51.3   5  3  42.5   7  0  31.3   3  2  40.6
;
```

You can model the data points y_i with a binomial distribution,

$$y_i | p_i \sim \text{binomial}(n_i, p_i)$$

where p_i is the success probability and links to the regression covariate x_i through a logit transformation:

$$logit(p_i) = \log\left(\frac{p_i}{1-p_i}\right) = \alpha + \beta x_i$$

The priors on α and β are both diffuse normal:

$$\begin{aligned} \alpha &\sim \text{normal}(0, \text{var} = 10000) \\ \beta &\sim \text{normal}(0, \text{var} = 10000) \end{aligned}$$

These statements fit a logistic regression with PROC MCMC:

```
ods graphics on;
proc mcmc data=beetles ntu=1000 nmc=20000 propcov=quanew
          diag=(mcse ess) outpost=beetleout seed=246810;
   ods select PostSumInt mcse ess TADpanel;
   parms (alpha  beta) 0;
   prior alpha beta ~ normal(0, var = 10000);
   p = logistic(alpha + beta*x);
   model y ~ binomial(n,p);
run;
```

The key statement in the program is the assignment to p that calculates the probability of death. The SAS function LOGISTIC does the proper transformation. The MODEL statement specifies that the response variable, y, is binomially distributed with parameters n (from the input data set) and p. The summary statistics table, interval statistics table, the Monte Carlo standard error table, and the effective sample sizes table are shown in Output 80.3.1.

Output 80.3.1 MCMC Results

Logistic Regression Model with a Diffuse Prior

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
alpha	20000	-11.7688	2.0941	-15.9411	-7.7492
beta	20000	0.2919	0.0541	0.1901	0.4029

Logistic Regression Model with a Diffuse Prior

The MCMC Procedure

Monte Carlo Standard Errors

Parameter	MCSE	Standard Deviation	MCSE/SD
alpha	0.0418	2.0941	0.0200
beta	0.00109	0.0541	0.0201

Effective Sample Sizes

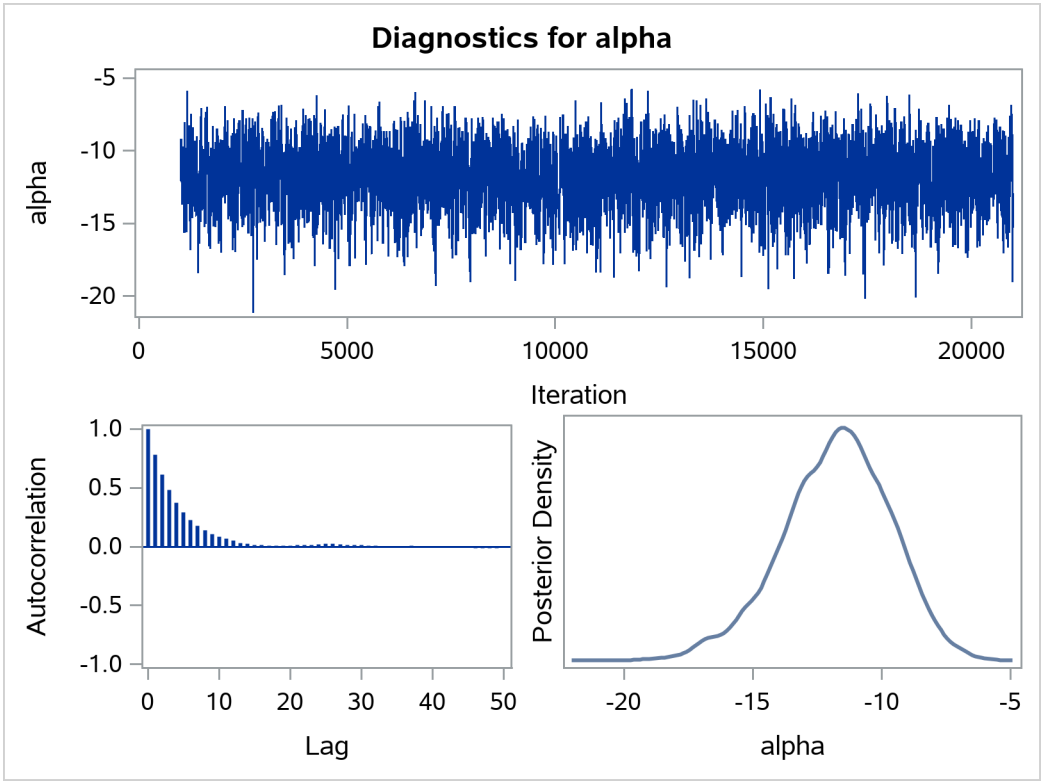
Parameter	ESS	Autocorrelation Time	Efficiency
alpha	2507.0	7.9778	0.1253
beta	2478.5	8.0694	0.1239

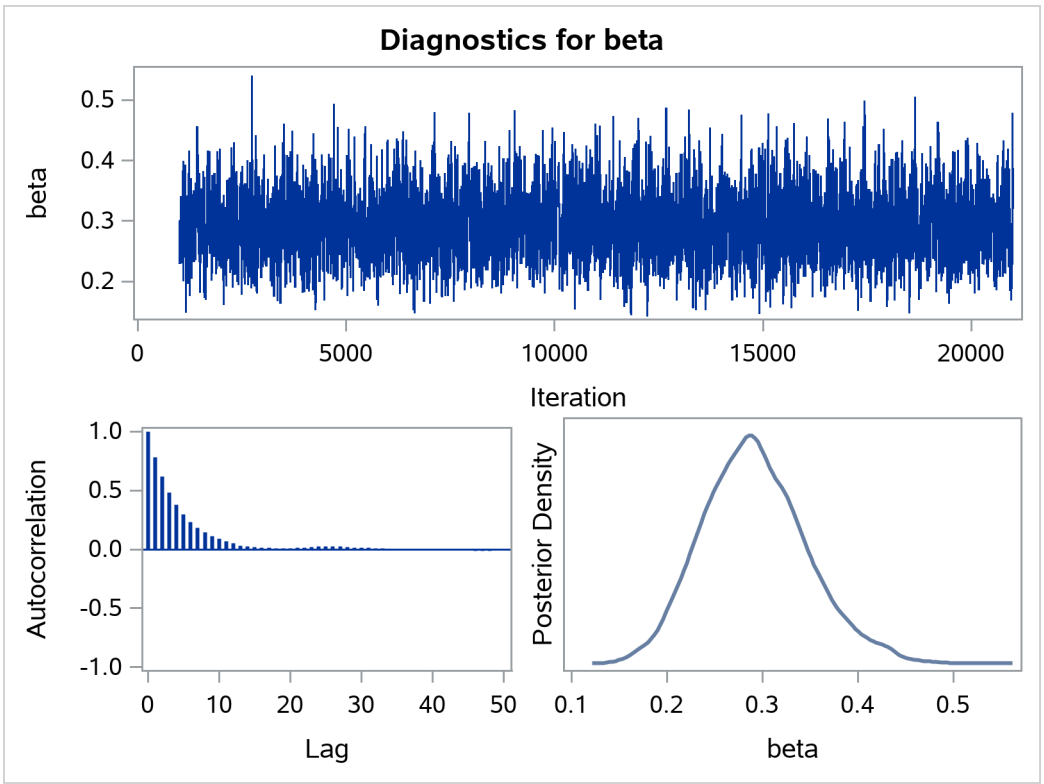
The summary statistics table shows that the sample mean of the output chain for the parameter alpha is –11.77. This is an estimate of the mean of the marginal posterior distribution for the intercept parameter alpha. The estimated posterior standard deviation for alpha is 2.09. The two 95% credible intervals for alpha are both negative, which indicates with very high probability that the intercept term is negative. On the other hand, you observe a positive effect on the regression coefficient beta. Exposure to the environment contaminant increases the probability of death.

The Monte Carlo standard errors of each parameter are significantly small relative to the posterior standard deviations. A small MCSE/SD ratio indicates that the Markov chain has stabilized and the mean estimates do not vary much over time. Note that the precision in the parameter estimates increases with the square of the MCMC sample size, so if you want to double the precision, you must quadruple the MCMC sample size.

MCMC chains do not produce independent samples. Each sample point depends on the point before it. In this case, the correlation time estimate, read from the effective sample sizes table, is roughly 8. This means that it takes four observations from the MCMC output to make inferences about alpha with the same precision that you would get from using an independent sample. The effective sample size of around 2500 reflects this loss of efficiency. The coefficient beta has similar efficiency. You can often observe that some parameters have significantly better mixing (better efficiency) than others, even in a single Markov chain run.

Output 80.3.2 Plots for Parameters in the Logistic Regression Example





Trace plots and autocorrelation plots of the posterior samples are shown in Output 80.3.2. Convergence looks

good in both parameters; there is good mixing in the trace plot and quick drop-off in the ACF plot.

One advantage of Bayesian methods is the ability to directly answer scientific questions. In this example, you might want to find out the posterior probability that the environmental contaminant increases the probability of death—that is, $\Pr(\beta > 0|\mathbf{y})$. This can be estimated using the following steps:

```
proc format;
   value betafmt low-0 = 'beta <= 0' 0<-high = 'beta > 0';
run;

proc freq data=beetleout;
   tables beta /nocum;
   format beta betafmt.;
run;
```

Output 80.3.3 Frequency Counts

Logistic Regression Model with a Diffuse Prior

The FREQ Procedure

beta	Frequency	Percent
beta > 0	20000	100.00

All of the simulated values for β are greater than zero, so the sample estimate of the posterior probability that $\beta > 0$ is 100%. The evidence overwhelmingly supports the hypothesis that increased levels of the environmental contaminant increase the probability of death.

If you are interested in making inference based on any quantities that are transformations of the random variables, you can either do it directly in PROC MCMC or by using the DATA step after you run the simulation. Transformations sometimes can make parameter inference quite formidable using direct analytical methods, but with simulated chains, it is easy to compute chains for any set of parameters. Suppose you are interested in the lethal dose and want to estimate the level of the covariate x that corresponds to a probability of death, p. Abbreviate this quantity as ldp. In other words, you want to solve the logit transformation with a fixed value p. The lethal dose is as follows:

$$\mathsf{ldp} = \frac{\log\left(\frac{p}{1-p}\right) - \alpha}{\beta}$$

You can obtain an estimate of any ldp by using the posterior mean estimates for α and β. For example, lp95, which corresponds to $p = 0.95$, is calculated as follows:

$$\mathsf{lp95} = \frac{\log\left(\frac{0.95}{1-0.95}\right) + 11.77}{0.29} = 50.79$$

where –11.77 and 0.29 are the posterior mean estimates of α and β, respectively, and 50.79 is the estimated lethal dose that leads to a 95% death rate.

While it is easy to obtain the point estimates, it is harder to estimate other posterior quantities, such as the standard deviation directly. However, with PROC MCMC, you can trivially get estimates of any posterior quantities of lp95. Consider the following program in PROC MCMC:

```
proc mcmc data=beetles ntu=1000 nmc=20000 propcov=quanew
          outpost=beetleout seed=246810 plot=density
          monitor=(pi30 ld05 ld50 ld95);
   ods select PostSumInt densitypanel;
   parms (alpha  beta) 0;
   begincnst;
      c1 = log(0.05 / 0.95);
      c2 = -c1;
   endcnst;

   beginnodata;
   prior alpha beta ~ normal(0, var = 10000);
   pi30 = logistic(alpha + beta*30);
   ld05 = (c1 - alpha) / beta;
   ld50 = - alpha / beta;
   ld95 = (c2 - alpha) / beta;
   endnodata;
   pi = logistic(alpha + beta*x);
   model y ~ binomial(n,pi);
run;
ods graphics off;
```

The program estimates four additional posterior quantities. The three lpd quantities, ld05, ld50, and ld95, are the three levels of the covariate that kills 5%, 50%, and 95% of the population, respectively. The predicted probability when the covariate x takes the value of 30 is pi30. The MONITOR= option selects the quantities of interest. The PLOTS= option selects kernel density plots as the only ODS graphical output, excluding the trace plot and autocorrelation plot.

Programming statements between the BEGINCNST and ENDCNST statements define two constants. These statements are executed once at the beginning of the simulation. The programming statements between the BEGINNODATA and ENDNODATA statements evaluate the quantities of interest. The symbols, pi30, ld05, ld50, and ld95, are functions of the parameters alpha and beta only. Hence, they should not be processed at the observation level and should be included in the BEGINNODATA and ENDNODATA statements. Output 80.3.4 lists the posterior summary and Output 80.3.5 shows the density plots of these posterior quantities.

Output 80.3.4 PROC MCMC Results

Logistic Regression Model with a Diffuse Prior

The MCMC Procedure

Posterior Summaries and Intervals

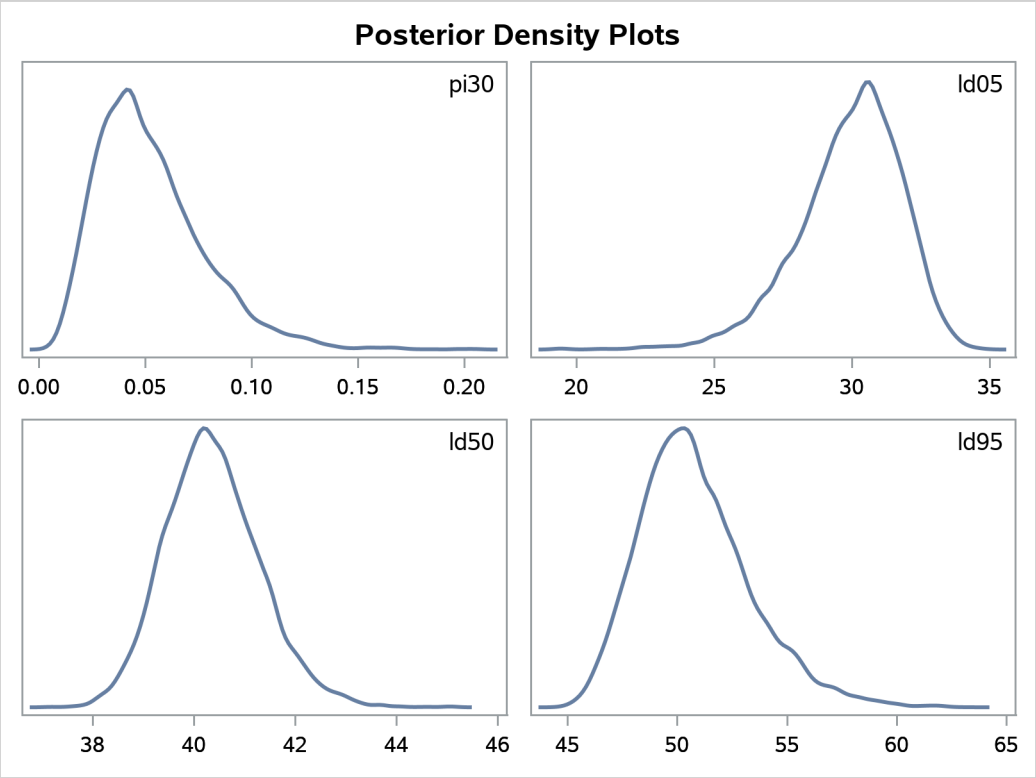
Parameter	N	Mean	Standard Deviation	95% HPD Interval	
pi30	20000	0.0524	0.0252	0.0126	0.1022
ld05	20000	29.9310	1.8731	26.2171	33.2648
ld50	20000	40.3780	0.9371	38.5334	42.1808
ld95	20000	50.8251	2.5327	46.2350	55.7957

The posterior mean estimate of lp95 is 50.82, which is close to the estimate of 50.79 by using the posterior mean estimates of the parameters. With PROC MCMC, in addition to the mean estimate, you can get the

standard deviation, quantiles, and interval estimates at any level of significance.

From the density plots, you can see, for example, that the sample distribution for π_{30} is skewed to the right, and almost all of your posterior belief concerning π_{30} is concentrated in the region between zero and 0.15.

Output 80.3.5 Density Plots of Quantities of Interest in the Logistic Regression Example



It is easy to use the DATA step to calculate these quantities of interest. The following DATA step uses the simulated values of α and β to create simulated values from the posterior distributions of ld05, ld50, ld95, and π_{30}:

```
data transout;
   set beetleout;
   pi30 = logistic(alpha + beta*30);
   ld05 = (log(0.05 / 0.95) - alpha) / beta;
   ld50 = (log(0.50 / 0.50) - alpha) / beta;
   ld95 = (log(0.95 / 0.05) - alpha) / beta;
run;
```

Subsequently, you can use SAS/INSIGHT, or the UNIVARIATE, CAPABILITY, or KDE procedures to analyze the posterior sample. If you want to regenerate the default ODS graphs from PROC MCMC, see "Regenerating Diagnostics Plots" on page 6325.

Example 80.4: Logistic Regression Model with Jeffreys' Prior

(View the complete code for this example (mcmcex4.sas) in the example repository.)

A controlled experiment was run to study the effect of the rate and volume of air inspired on a transient reflex vasoconstriction in the skin of the fingers. Thirty-nine tests under various combinations of rate and volume of air inspired were obtained (Finney 1947). The result of each test is whether or not vasoconstriction occurred. Pregibon (1981) uses this set of data to illustrate the diagnostic measures he proposes for detecting influential observations and to quantify their effects on various aspects of the maximum likelihood fit. The following statements create the data set Vaso:

```
title 'Logistic Regression Model with Jeffreys Prior';
data vaso;
   input vol rate resp @@;
   lvol = log(vol);
   lrate = log(rate);
   ind = _n_;
   cnst = 1;
   datalines;
3.7  0.825  1  3.5  1.09  1  1.25  2.5   1  0.75  1.5  1
0.8  3.2    1  0.7  3.5   1  0.6   0.75  0  1.1   1.7  0
0.9  0.75   0  0.9  0.45  0  0.8   0.57  0  0.55  2.75 0
0.6  3.0    0  1.4  2.33  1  0.75  3.75  1  2.3   1.64 1
3.2  1.6    1  0.85 1.415 1  1.7   1.06  0  1.8   1.8  1
0.4  2.0    0  0.95 1.36  0  1.35  1.35  0  1.5   1.36 0
1.6  1.78   1  0.6  1.5   0  1.8   1.5   1  0.95  1.9  0
1.9  0.95   1  1.6  0.4   0  2.7   0.75  1  2.35  0.03 0
1.1  1.83   0  1.1  2.2   1  1.2   2.0   1  0.8   3.33 1
0.95 1.9    0  0.75 1.9   0  1.3   1.625 1
;
```

The variable resp represents the outcome of a test. The variable lvol represents the log of the volume of air intake, and the variable lrate represents the log of the rate of air intake. You can model the data by using logistic regression. You can model the response with a binary likelihood:

$$\text{resp}_i \sim \text{binary}(p_i)$$

with

$$p_i = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 \text{lvol}_i + \beta_2 \text{lrate}_i))}$$

Let $\mathbf{X}$ be the design matrix in the regression. Jeffreys' prior for this model is

$$p(\boldsymbol{\beta}) \propto |\mathbf{X}'\mathbf{M}\mathbf{X}|^{1/2}$$

where $\mathbf{M}$ is a 39 by 39 matrix with off-diagonal elements being 0 and diagonal elements being $p_i(1 - p_i)$. For details on Jeffreys' prior, see "Jeffreys' Prior" on page 151 in Chapter 8, "Introduction to Bayesian Analysis Procedures." You can use a number of matrix functions, such as the determinant function, in PROC MCMC to construct Jeffreys' prior. The following statements illustrate how to fit a logistic regression with Jeffreys' prior:

```
%let n = 39;
proc mcmc data=vaso nmc=10000 outpost=mcmcout seed=17;
   ods select PostSumInt;

   array beta[3] beta0 beta1 beta2;
   array m[&n, &n];
   array x[1] / nosymbols;
   array xt[3, &n];
   array xtm[3, &n];
   array xmx[3, 3];
   array p[&n];

   parms beta0 1 beta1 1 beta2 1;

   begincnst;
      if (ind eq 1) then do;
         rc = read_array("vaso", x, "cnst", "lvol", "lrate");
         call transpose(x, xt);
         call zeromatrix(m);
      end;
   endcnst;

   beginnodata;
   call mult(x, beta, p);              /* p = x * beta */
   do i = 1 to &n;
      p[i] = 1 / (1 + exp(-p[i]));     /* p[i] = 1/(1+exp(-x*beta)) */
      m[i,i] = p[i] * (1-p[i]);
   end;
   call mult (xt, m, xtm);             /* xtm = xt * m        */
   call mult (xtm, x, xmx);            /* xmx = xtm * x       */
   call det (xmx, lp);                 /* lp = det(xmx)       */
   lp = 0.5 * log(lp);                 /* lp = -0.5 * log(lp) */
   prior beta: ~ general(lp);
   endnodata;

   model resp ~ bern(p[ind]);
run;
```

The first ARRAY statement defines an array beta with three elements: beta0, beta1, and beta2. The subsequent statements define arrays that are used in the construction of Jeffreys' prior. These include m (the M matrix), x (the design matrix), xt (the transpose of x), and some additional work spaces.

The explanatory variables lvol and lrate are saved in the array x in the BEGINCNST and ENDCNST statements. See "BEGINCNST/ENDCNST Statement" on page 6234 for details. After all the variables are read into x, you transpose the x matrix and store it to xt. The ZEROMATRIX function call assigns all elements in matrix m the value zero. To avoid redundant calculation, it is best to perform these calculations as the last observation of the data set is processed—that is, when ind is 39.

You calculate Jeffreys' prior in the BEGINNODATA and ENDNODATA statements. The probability vector p is the product of the design matrix x and parameter vector beta. The diagonal elements in the matrix m are $p_i(1-p_i)$. The expression lp is the logarithm of Jeffreys' prior. The PRIOR statement assigns lp as the prior for the $\boldsymbol{\beta}$ regression coefficients. The MODEL statement assigns a binary likelihood to resp, with probability p[ind]. The p array is calculated earlier using the matrix function MULT. You use the ind variable to pick out

the right probability value for each resp.

Posterior summary statistics are displayed in Output 80.4.1.

Output 80.4.1 PROC MCMC Results, Jeffreys' prior

Logistic Regression Model with Jeffreys Prior

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta0	10000	-2.9587	1.3258	-5.5936	-0.6027
beta1	10000	5.2905	1.8193	1.8590	8.7222
beta2	10000	4.6889	1.8189	1.3611	8.2490

You can also use PROC GENMOD to fit the same model by using the following statements:

```
proc genmod data=vaso descending;
   ods select PostSummaries PostIntervals;
   model resp = lvol lrate / d=bin link=logit;
   bayes seed=17 coeffprior=jeffreys nmc=20000 thin=2;
run;
```

The MODEL statement indicates that resp is the response variable and lvol and lrate are the covariates. The options in the MODEL statement specify a binary likelihood and a logit link function. The BAYES statement requests Bayesian capability. The SEED=, NMC=, and THIN= arguments work in the same way as in PROC MCMC. The COEFFPRIOR=JEFFREYS option requests Jeffreys' prior in this analysis.

The PROC GENMOD statements produce Output 80.4.2, with estimates very similar to those reported in Output 80.4.1. Note that you should not expect to see identical output from PROC GENMOD and PROC MCMC, even with the simulation setup and identical random number seed. The two procedures use different sampling algorithms. PROC GENMOD uses the adaptive rejection metropolis algorithm (ARMS) (Gilks and Wild 1992; Gilks 2003) while PROC MCMC uses a random walk Metropolis algorithm. The asymptotic answers, which means that you let both procedures run an very long time, would be the same as they both generate samples from the same posterior distribution.

Output 80.4.2 PROC GENMOD Results

Logistic Regression Model with Jeffreys Prior

The GENMOD Procedure

Bayesian Analysis

Posterior Summaries

Parameter	N	Mean	Standard Deviation	Percentiles 25%	50%	75%
Intercept	10000	-2.8773	1.3213	-3.6821	-2.7326	-1.9097
lvol	10000	5.2059	1.8707	3.8535	4.9574	6.3337
lrate	10000	4.5525	1.8140	3.2281	4.3722	5.6643

Output 80.4.2 *continued*

Posterior Intervals					
		Equal-Tail			
Parameter	Alpha	Interval		HPD Interval	
Intercept	0.050	-5.7447	-0.6877	-5.4593	-0.5488
lvol	0.050	2.2066	9.4415	2.0729	9.2343
lrate	0.050	1.5906	8.5272	1.3351	8.1152

Example 80.5: Poisson Regression

(View the complete code for this example (mcmcex5.sas) in the example repository.)

You can use the Poisson distribution to model the distribution of cell counts in a multiway contingency table. Aitkin et al. (1989) have used this method to model insurance claims data. Suppose the following hypothetical insurance claims data are classified by two factors: age group (with two levels) and car type (with three levels). The following statements create the data set:

```
title 'Poisson Regression';
data insure;
   input n c car $ age;
   ln = log(n);
   datalines;
   500   42  small  0
   1200  37  medium 0
   100    1  large  0
   400  101  small  1
   500   73  medium 1
   300   14  large  1
;

proc transreg data=insure design;
   model class(car / zero=last);
   id n c age ln;
   output out=input_insure(drop=_: Int:);
   run;
```

The variable n represents the number of insurance policy holders and the variable c represents the number of insurance claims. The variable car is the type of car involved (classified into three groups), and it is coded into two levels. The variable age is the age group of a policy holder (classified into two groups).

Assume that the number of claims c has a Poisson probability distribution and that its mean, μ_i, is related to the factors car and age for observation i by

$$\begin{aligned}\log(\mu_i) &= \log(\mathrm{n}_i) + \mathbf{x}'\boldsymbol{\beta} \\ &= \log(\mathrm{n}_i) + \beta_0 + \\ &\quad \mathrm{car}_i(1)\beta_1 + \mathrm{car}_i(2)\beta_2 + \mathrm{car}_i(3)\beta_3 + \\ &\quad \mathrm{age}_i(1)\beta_4 + \mathrm{age}_i(2)\beta_5\end{aligned}$$

The indicator variables $\text{car}_i(j)$ is associated with the jth level of the variable car for observation i in the following way:

$$\text{car}_i(j) = \begin{cases} 1 & \text{if car} = j \\ 0 & \text{if car} \neq j \end{cases}$$

A similar coding applies to age. The $\boldsymbol{\beta}$'s are parameters. The logarithm of the variable n is used as an offset—that is, a regression variable with a constant coefficient of 1 for each observation. Having the offset constant in the model is equivalent to fitting an expanded data set with 3000 observations, each with response variable y observed on an individual level. The log link relates the mean and the factors car and age.

The following statements run PROC MCMC:

```
proc mcmc data=input_insure outpost=insureout nmc=5000 propcov=quanew
          maxtune=0 seed=7;
   ods select PostSumInt;
   array data[4] 1 &_trgind age;
   array beta[4] alpha beta_car1 beta_car2 beta_age;
   parms alpha beta:;
   prior alpha beta: ~ normal(0, prec = 1e-6);
   call mult(data, beta, mu);
   model c ~ poisson(exp(mu+ln));
run;
```

The analysis uses a relatively flat prior on all the regression coefficients, with mean at 0 and precision at 10^{-6}. The option MAXTUNE=0 skips the tuning phase because the optimization routine (PROPCOV=QUANEW) provides good initial values and proposal covariance matrix.

There are four parameters in the model: alpha is the intercept; beta_car1 and beta_car2 are coefficients for the CLASS variable car, which has three levels; and beta_age is the coefficient for age. The symbol mu connects the regression model and the Poisson mean by using the log link. The MODEL statement specifies a Poisson likelihood for the response variable c.

Posterior summary and interval statistics are shown in Output 80.5.1.

Output 80.5.1 MCMC Results

Poisson Regression

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
alpha	5000	-2.6403	0.1344	-2.9133	-2.3831
beta_car1	5000	-1.8335	0.2917	-2.4692	-1.3336
beta_car2	5000	-0.6931	0.1255	-0.9485	-0.4589
beta_age	5000	1.3151	0.1386	1.0387	1.5812

To fit the same model by using PROC GENMOD, you can do the following. Note that the default normal prior on the coefficients $\boldsymbol{\beta}$ is $N(0, \text{prec} = 1e-6)$, the same as used in the PROC MCMC. The following statements run PROC GENMOD and create Output 80.5.2:

```
proc genmod data=insure;
   ods select PostSummaries PostIntervals;
   class car age(descending);
   model c = car age / dist=poisson link=log offset=ln;
   bayes seed=17 nmc=5000 coeffprior=normal;
run;
```

To compare, posterior summary and interval statistics from PROC GENMOD are reported in Output 80.5.2, and they are very similar to PROC MCMC results in Output 80.5.1.

Output 80.5.2 PROC GENMOD Results

Poisson Regression

The GENMOD Procedure

Bayesian Analysis

Posterior Summaries

Parameter	N	Mean	Standard Deviation	Percentiles 25%	50%	75%
Intercept	5000	-2.6424	0.1336	-2.7334	-2.6391	-2.5547
carlarge	5000	-1.8040	0.2764	-1.9859	-1.7929	-1.6101
carmedium	5000	-0.6908	0.1311	-0.7797	-0.6898	-0.6044
age1	5000	1.3207	0.1384	1.2264	1.3209	1.4140

Posterior Intervals

Parameter	Alpha	Equal-Tail Interval		HPD Interval	
Intercept	0.050	-2.9154	-2.3893	-2.8997	-2.3850
carlarge	0.050	-2.3668	-1.2891	-2.2992	-1.2378
carmedium	0.050	-0.9437	-0.4231	-0.9434	-0.4230
age1	0.050	1.0455	1.5871	1.0266	1.5629

Note that the descending option in the CLASS statement reverses the sort order of the CLASS variable age so that the results agree with PROC MCMC. If this option is not used, the estimate for age has a reversed sign as compared to Output 80.5.2.

Example 80.6: Nonlinear Poisson Regression Models

(View the complete code for this example (mcmcex6.sas) in the example repository.)

This example illustrates how to fit a nonlinear Poisson regression with PROC MCMC. In addition, it shows how you can improve the mixing of the Markov chain by selecting a different proposal distribution or by sampling on the transformed scale of a parameter. This example shows how to analyze count data for calls to a technical support help line in the weeks immediately following a product release. This information could be used to decide upon the allocation of technical support resources for new products. You can model the number of daily calls as a Poisson random variable, with the average number of calls modeled as a nonlinear function of the number of weeks that have elapsed since the product's release. The data are input into a SAS data set as follows:

```
title 'Nonlinear Poisson Regression';
data calls;
   input weeks calls @@;
   datalines;
1   0   1   2   2   2   2   1   3   1   3   3
4   5   4   8   5   5   5   9   6  17   6   9
7  24   7  16   8  23   8  27
;
```

During the first several weeks after a new product is released, the number of questions that technical support receives concerning the product increases in a sigmoidal fashion. The expression for the mean value in the classic Poisson regression involves the log link. There is some theoretical justification for this link, but with MCMC methodologies, you are not constrained to exploring only models that are computationally convenient. The number of calls to technical support tapers off after the initial release, so in this example you can use a logistic-type function to model the mean number of calls received weekly for the time period immediately following the initial release. The mean function $\lambda(t)$ is modeled as follows:

$$\lambda_i = \frac{\gamma}{1 + \exp\left[-(\alpha + \beta t_i)\right]}$$

The likelihood for every observation calls_i is

$$\text{calls}_i \sim \text{Poisson}\left(\lambda_i\right)$$

Past experience with technical support data for similar products suggests the following prior distributions:

$$\begin{aligned}
\gamma &\sim \text{gamma}(\text{shape} = 3.5, \text{scale} = 12) \\
\alpha &\sim \text{normal}(-5, \text{sd} = 0.5) \\
\beta &\sim \text{normal}(0.75, \text{sd} = 0.5)
\end{aligned}$$

The following PROC MCMC statements fit this model:

```
ods graphics on;
proc mcmc data=calls outpost=callout seed=53197 ntu=1000 nmc=20000
       propcov=quanew stats=none diag=ess;
   ods select TADpanel ess;
   parms alpha -4 beta 1 gamma 2;
```

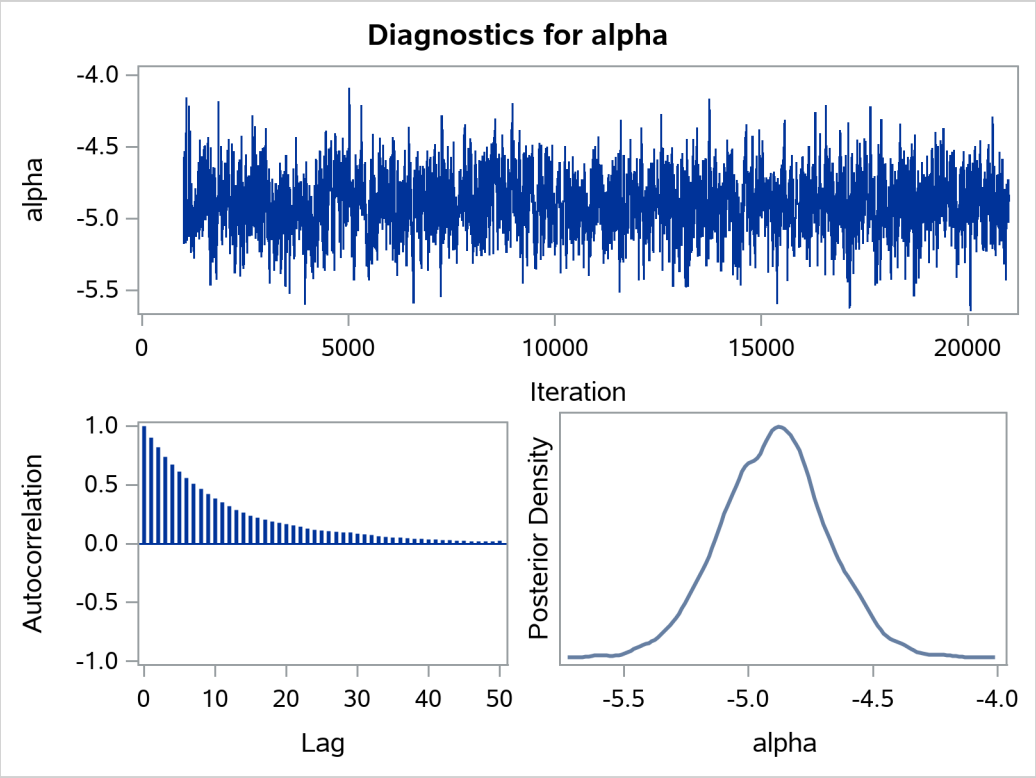
```
   prior gamma ~ gamma(3.5, scale=12);
   prior alpha ~ normal(-5, sd=0.25);
   prior beta  ~ normal(0.75, sd=0.5);
   lambda = gamma*logistic(alpha+beta*weeks);
   model calls ~ poisson(lambda);
run;
```

The one PARMS statement defines a block of all parameters and sets their initial values individually. The PRIOR statements specify the informative prior distributions for the three parameters. The assignment statement defines λ, the mean number of calls. Instead of using the SAS function LOGISTIC, you can use the following statement to calculate λ and get the same result:

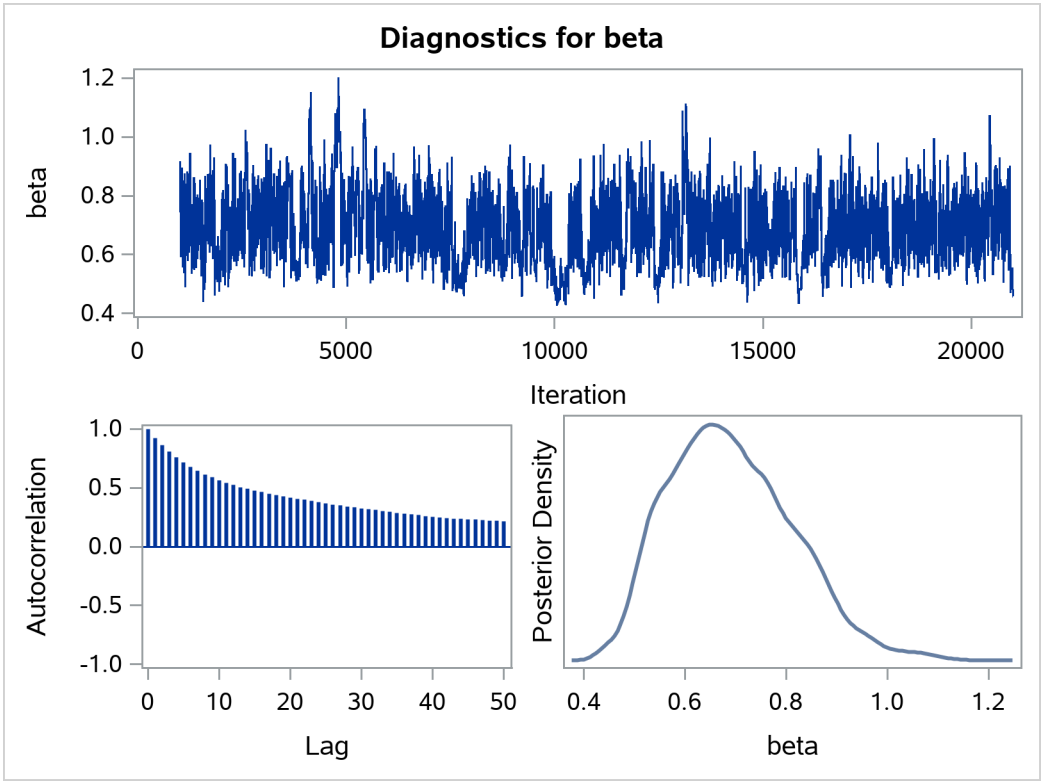
```
lambda = gamma / (1 + exp(-(alpha+beta*weeks)));
```

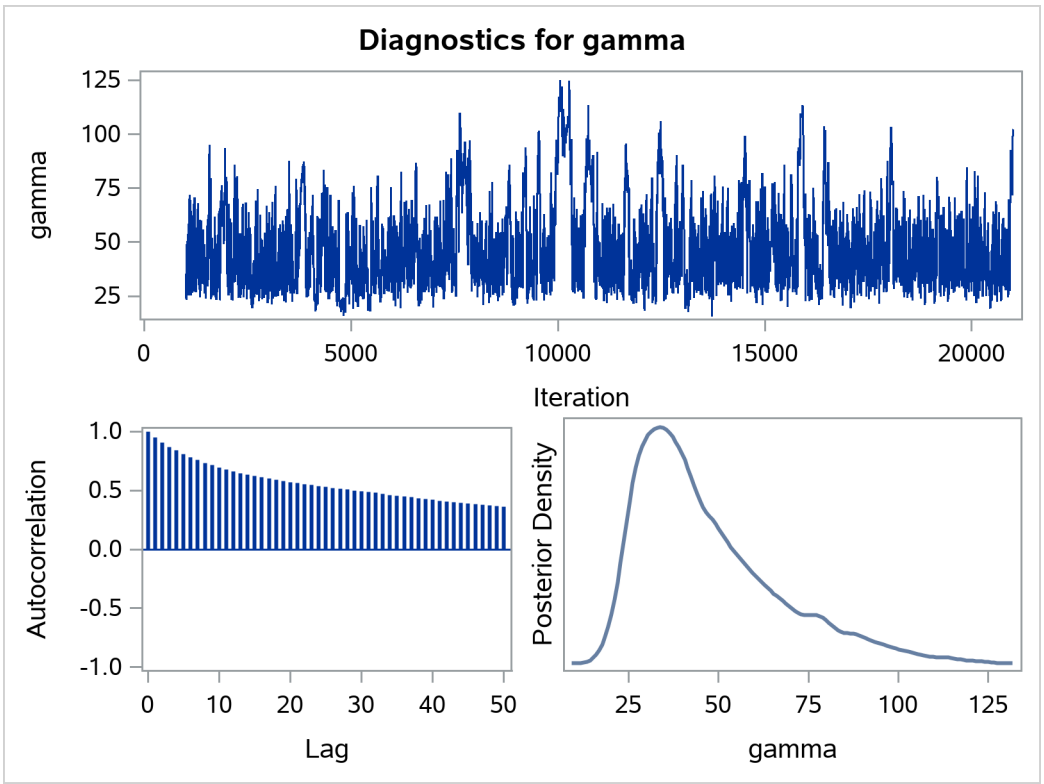
Mixing is not particularly good with this run of PROC MCMC. The ODS SELECT statement displays the diagnostic graphs and effective sample sizes (ESS) calculation while excluding all other output. The graphical output is shown in Output 80.6.1, and the ESS of each parameters are all relatively low (Output 80.6.2).

Output 80.6.1 Plots for Parameters



Output 80.6.1 *continued*





Output 80.6.2 Effective Sample Sizes

Nonlinear Poisson Regression

The MCMC Procedure

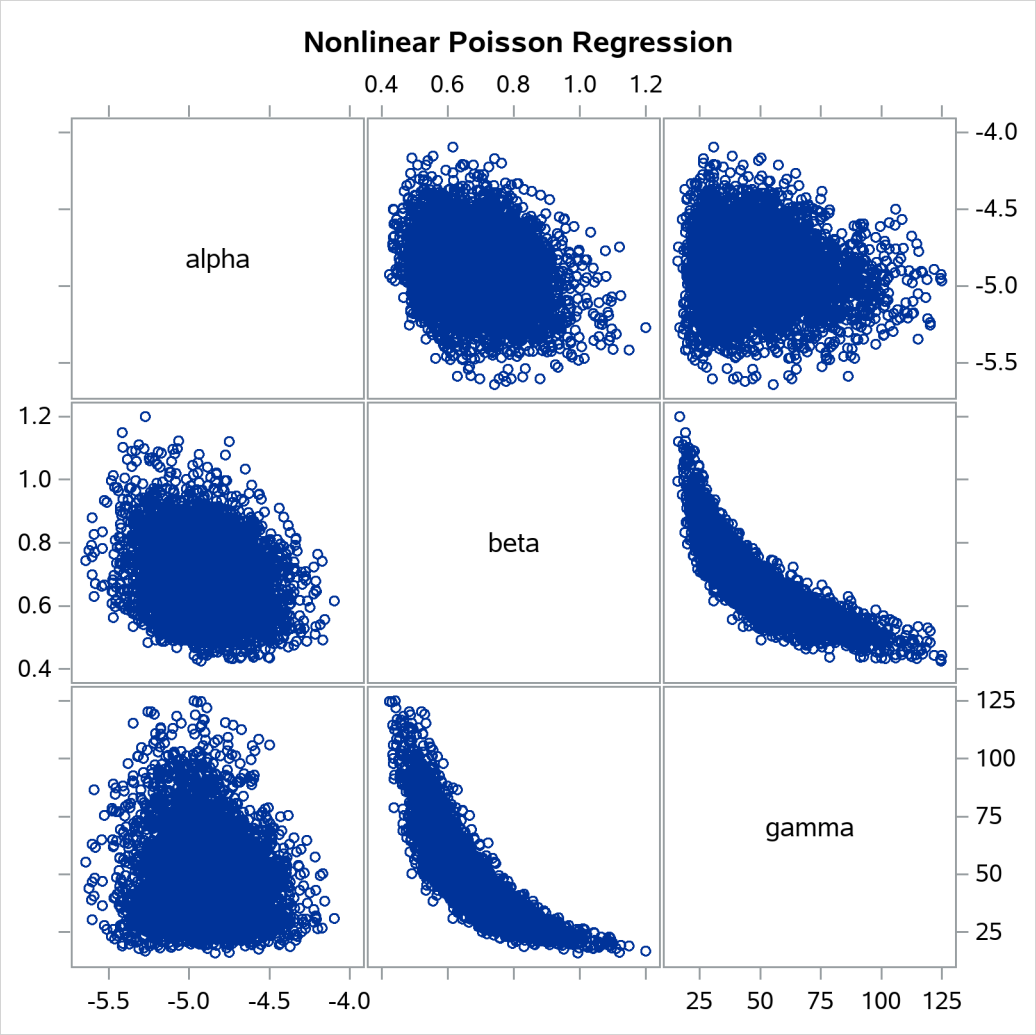
Effective Sample Sizes

Parameter	ESS	Autocorrelation Time	Efficiency
alpha	897.4	22.2870	0.0449
beta	231.6	86.3553	0.0116
gamma	162.9	122.8	0.0081

Often a simple scatter plot of the posterior samples can reveal a potential cause of the bad mixing. You can use PROC SGSCATTER to generate pairwise scatter plots of the three model parameters. The following statements generate Output 80.6.3:

```
proc sgscatter data=callout;
   matrix alpha beta gamma;
run;
```

Output 80.6.3 Pairwise Scatter Plots of the Parameters



The nonlinearity in parameters beta and gamma stands out immediately. This explains why a random walk Metropolis with normal proposal has a difficult time exploring the joint distribution efficiently—the algorithm works best when the target distribution is unimodal and symmetric (normal-like). When there is nonlinearity in the parameters, it is impossible to find a single proposal scale parameter that optimally adapts to different regions of the joint parameter space. As a result, the Markov chain can be inefficient in traversing some parts of the distribution. This is evident in examining the trace plot of the gamma parameter. You see that the Markov chain sometimes gets stuck in the far-right tail and does not travel back to the high-density area quickly. This effect can be seen around the simulations 8,000 and 18,000 in Output 80.6.1.

Reparameterization can often improve the mixing of the Markov chain. Note that the parameter gamma has a positive support and that the posterior distribution is right-skewed. This suggests that the chain might mix more rapidly if you sample on the logarithm of the parameter gamma.

Let $\delta = \log(\gamma)$, and reparameterize the mean function as follows:

$$\lambda_i = \frac{\exp(\delta)}{1 + \exp\left[-(\alpha + \beta t_i)\right]}$$

To obtain the same inference, you use an induced prior on delta based on the gamma prior on the gamma parameter. This involves a transformation of variables, and you can obtain the following equivalency, where $|\exp(\delta)|$ is the Jacobian:

$$\begin{aligned} & \pi(\gamma) = \text{gamma}(\gamma; a, \text{scale} = b) = \frac{1}{b^a \Gamma(a)} \gamma^{a-1} \exp(-\gamma/b) \\ \Leftrightarrow \quad & \pi(\delta) = \text{gamma}(\gamma = \exp(\delta); a, \text{scale} = b) \cdot |\exp(\delta)| \end{aligned}$$

The distribution on δ simplifies to the following:

$$\pi(\delta) = \frac{1}{b^a \Gamma(a)} \exp(a\delta) \exp(-\exp(\delta)/b)$$

PROC MCMC supports such a distribution on the logarithm transformation of a gamma random variable. It is called the ExpGamma distribution.

In the original model, you specify a prior on gamma:

```
prior gamma ~ gamma(3.5, scale=12);
```

You can obtain the same inference by specifying an ExpGamma prior on delta and take an exponential transformation to get back to gamma:

```
prior delta ~ egamma(3.5, scale=12);
gamma = exp(delta);
```

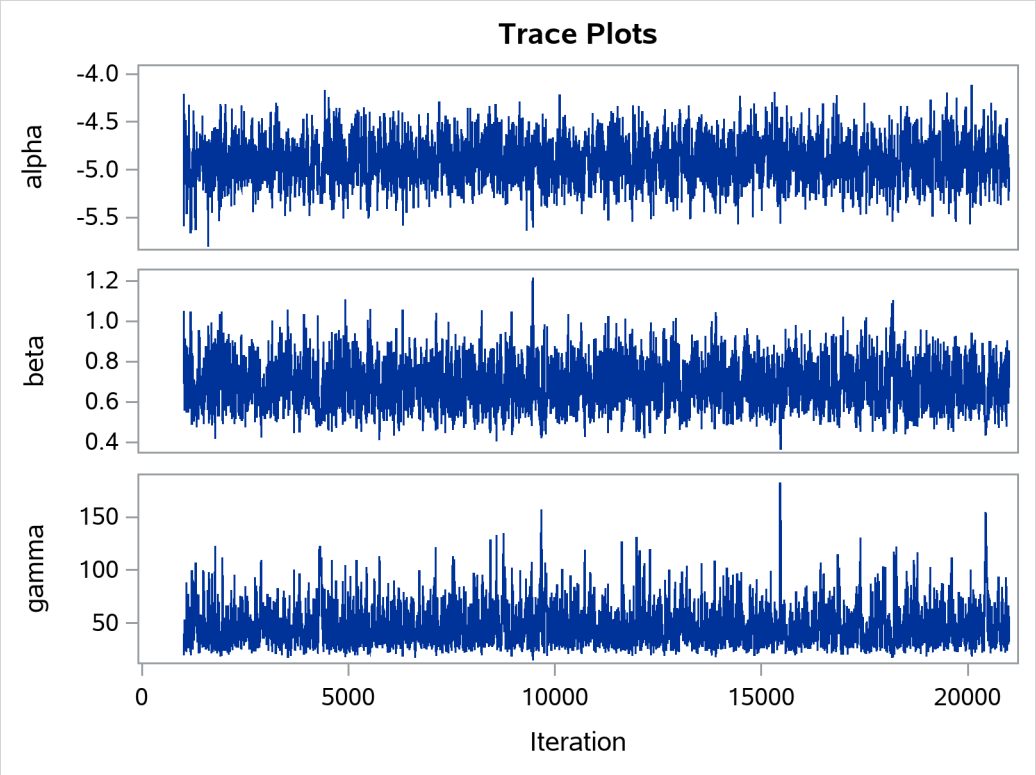
The following statements produce Output 80.6.6 and Output 80.6.4:

```
proc mcmc data=calls outpost=tcallout seed=53197 ntu=1000 nmc=20000
       propcov=quanew diag=ess plots=(trace) monitor=(alpha beta gamma);
   ods select PostSumInt ESS TRACEpanel;
   parms alpha -4 beta 1 delta 2;
   prior alpha ~ normal(-5, sd=0.25);
   prior beta  ~ normal(0.75, sd=0.5);
   prior delta ~ egamma(3.5, scale=12);
   gamma = exp(delta);
   lambda = gamma*logistic(alpha+beta*weeks);
   model calls ~ poisson(lambda);
run;
```

The PARMS statement declares delta, instead of gamma, as a model parameter. The prior distribution of delta is **egamma**, as opposed to the **gamma** distribution. The GAMMA assignment statement transforms delta to gamma. The LAMBDA assignment statement calculates the mean for the Poisson by using the gamma parameter. The MODEL statement specifies a Poisson likelihood for the calls response.

The trace plots in Output 80.6.4 show better mixing of the parameters, and the effective sample sizes in Output 80.6.5 show substantial improvements over the original formulation of the model. The improvements are especially obvious in beta and gamma, where the increase is fivefold to tenfold.

Output 80.6.4 Plots for Parameters, Sampling on the Log Scale of Gamma



Output 80.6.5 Effective Sample Sizes, Sampling on the Log Scale of Gamma

Nonlinear Poisson Regression

The MCMC Procedure

Effective Sample Sizes

Parameter	ESS	Autocorrelation Time	Efficiency
alpha	1457.3	13.7242	0.0729
beta	1071.9	18.6589	0.0536
gamma	951.8	21.0123	0.0476

Output 80.6.6 shows the posterior summary and interval statistics of the nonlinear Poisson regression.

Output 80.6.6 MCMC Results, Sampling on the Log Scale of Gamma

Nonlinear Poisson Regression

The MCMC Procedure

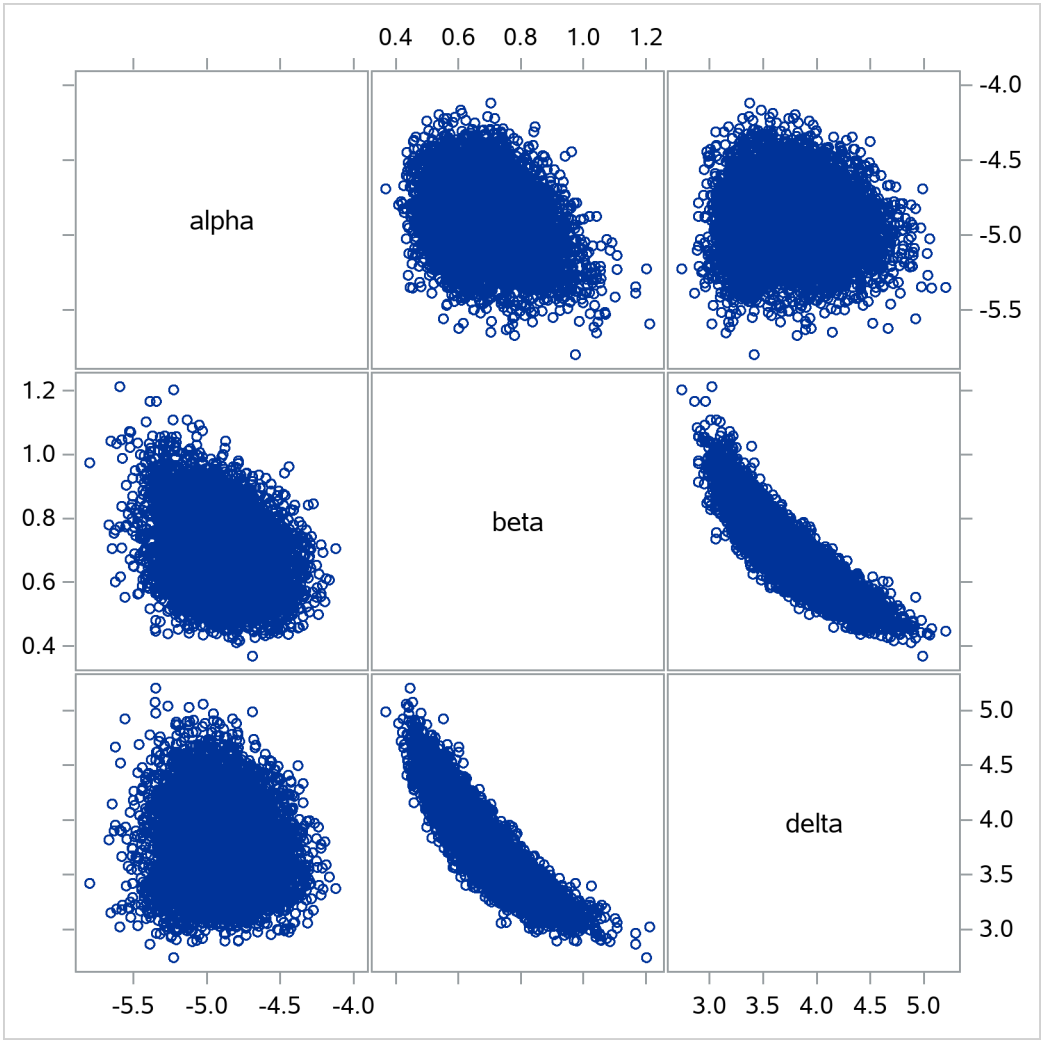
Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
alpha	20000	-4.9054	0.2268	-5.3551	-4.4658
beta	20000	0.6904	0.1160	0.4732	0.9059
gamma	20000	46.7576	19.7467	20.5841	87.6046

Note that the delta parameter has a more symmetric density than the skewed gamma parameter. A pairwise scatter plot (Output 80.6.7) shows a more linear relationship between beta and delta. The Metropolis algorithm always works better if the target distribution is approximately normal.

```
proc sgscatter data=tcallout;
   matrix alpha beta delta;
run;
```

Output 80.6.7 Pairwise Scatter Plots of the Transformed Parameters



If you are still unsatisfied with the slight nonlinearity in the parameters beta and delta, you can try another transformation on beta. Normally you would not want to do a logarithm transformation on a parameter that has support on the real axis, because you would risk taking the logarithm of negative values. However, because all the beta samples are positive and the marginal posterior distribution is away from 0, you can try a such a transformation.

Let $\kappa = \log(\beta)$. The prior distribution on κ is the following:

$$\pi(\kappa) = \text{normal}(\beta = \exp(\kappa); \mu, \sigma^2) \cdot |\exp(\kappa)|$$

You can specify the prior distribution in PROC MCMC by using a GENERAL function:

```
parms kappa;
lprior = logpdf("normal", exp(kappa), 0.75, 0.5) + kappa;
prior kappa ~ general(lp);
beta = exp(kappa);
```

The PARMS statement declares the transformed parameter kappa, which will be sampled. The LPRIOR assignment statement defines the logarithm of the prior distribution on kappa. The LOGPDF function is used here to simplify the specification of the distribution. The PRIOR statement specifies the nonstandard distribution as the prior on kappa. Finally, the BETA assignment statement transforms kappa back to the beta parameter.

Applying logarithm transformations on both beta and gamma yields the best mixing. (The results are not shown here, but you can find the code in the file *mcmcex6.sas* in the SAS Sample Library.) The transformed parameters kappa and delta have much clearer linear correlation. However, the improvement over the case where gamma alone is transformed is only marginally significant (50% increase in ESS).

This example illustrates that PROC MCMC can fit Bayesian nonlinear models just as easily as Bayesian linear models. More importantly, transformations can sometimes improve the efficiency of the Markov chain, and that is something to always keep in mind. Also see "Example 80.20: Using a Transformation to Improve Mixing" on page 6473 for another example of how transformations can improve mixing of the Markov chains.

Example 80.7: Logistic Regression Random-Effects Model

(View the complete code for this example (mcmcex7.sas) in the example repository.)

This example illustrates how you can use PROC MCMC to fit random-effects models. In the example "Random-Effects Model" on page 6210 in "Getting Started: MCMC Procedure" on page 6200, you already saw PROC MCMC fit a linear random-effects model. This example shows how to fit a logistic random-effects model in PROC MCMC. Although you can use PROC MCMC to analyze random-effects models, you might want to first consider some other SAS procedures. For example, you can use PROC MIXED (see Chapter 84, "The MIXED Procedure") to analyze linear mixed effects models, PROC NLMIXED (see Chapter 89, "The NLMIXED Procedure") for nonlinear mixed effects models, and PROC GLIMMIX (see Chapter 52, "The GLIMMIX Procedure") for generalized linear mixed effects models. In addition, a sampling-based Bayesian analysis is available in the MIXED procedure through the PRIOR statement (see "PRIOR Statement" on page 6791 in Chapter 84, "The MIXED Procedure").

The data are taken from Crowder (1978). The Seeds data set is a 2×2 factorial layout, with two types of seeds, *O. aegyptiaca 75* and *O. aegyptiaca 73*, and two root extracts, *bean* and *cucumber*. You observe r, which is the number of germinated seeds, and n, which is the total number of seeds. The independent variables are seed and extract.

The following statements create the data set:

```
title 'Logistic Regression Random-Effects Model';
data seeds;
   input r n seed extract @@;
   ind = _N_;
   datalines;
10  39  0  0    23  62  0  0    23  81  0  0    26  51  0  0
17  39  0  0     5   6  0  1    53  74  0  1    55  72  0  1
32  51  0  1    46  79  0  1    10  13  0  1     8  16  1  0
10  30  1  0     8  28  1  0    23  45  1  0     0   4  1  0
 3  12  1  1    22  41  1  1    15  30  1  1    32  51  1  1
 3   7  1  1
;
```

You can model each observation r_i as having its own probability of success p_i, and the likelihood is as follows:

$$r_i \sim \text{binomial}(n_i, p_i)$$

You can use the logit link function to link the covariates of each observation, seed and extract, to the probability of success,

$$\begin{aligned} \mu_i &= \beta_0 + \beta_1 \cdot \text{seed}_i + \beta_2 \cdot \text{extract}_i + \beta_3 \cdot \text{seed}_i \cdot \text{extract}_i \\ p_i &= \text{logistic}(\mu_i + \delta_i) \end{aligned}$$

where δ_i is assumed to be an iid random effect with a normal prior:

$$\delta_i \sim \text{normal}(0, \text{var} = \sigma^2)$$

The four $\boldsymbol{\beta}$ regression coefficients and the standard deviation σ^2 in the random effects are model parameters; they are given noninformative priors as follows:

$$\begin{aligned} \pi(\beta_0, \beta_1, \beta_2, \beta_3) &\propto 1 \\ \sigma^2 &\sim \text{igamma}(\text{shape} = 0.01, \text{scale} = 0.01) \end{aligned}$$

Another way of expressing the same model is as

$$p_i = \text{logistic}(\delta_i)$$

where

$$\delta_i \sim \text{normal}(\beta_0 + \beta_1 \cdot \text{seed}_i + \beta_2 \cdot \text{extract}_i + \beta_3 \cdot \text{seed}_i \cdot \text{extract}_i, \sigma^2)$$

The two models are equivalent. In the first model, the random effects δ_i centers at 0 in the normal distribution, and in the second model, δ_i centers at the regression mean. This hierarchical centering can sometimes improve mixing.

The following statements fit the second model and generate Output 80.7.1:

```
proc mcmc data=seeds outpost=postout seed=332786 nmc=20000;
   ods select PostSumInt;
   parms beta0 0 beta1 0 beta2 0 beta3 0 s2 1;
   prior s2 ~ igamma(0.01, s=0.01);
   prior beta: ~ general(0);
   w = beta0 + beta1*seed + beta2*extract + beta3*seed*extract;
   random delta ~ normal(w, var=s2) subject=ind;
   pi = logistic(delta);
   model r ~ binomial(n = n, p = pi);
run;
```

The PROC MCMC statement specifies the input and output data sets, sets a seed for the random number generator, and requests a large simulation size. The ODS SELECT statement displays the summary statistics and interval statistics tables. The PARMS statement declares the model parameters, and the PRIOR statements specify the prior distributions for $\boldsymbol{\beta}$ and σ^2.

The symbol w calculates the regression mean, and the RANDOM statement specifies the random effect, with a normal prior distribution, centered at w with variance σ^2. Note that the variable w is a function of the input data set variables. You can use data set variable in constructing the hyperparameters of the random-effects parameters, as long as the hyperparameters remain constant within each subject group. The SUBJECT= option indicates the group index for the random-effects parameters.

The symbol pi is the logit transformation. The MODEL specifies the response variable r as a binomial distribution with parameters n and pi.

Output 80.7.1 lists the posterior mean and interval estimates of the regression parameters.

Output 80.7.1 Logistic Regression Random-Effects Model

Logistic Regression Random-Effects Model

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta0	20000	-0.5570	0.1929	-0.9422	-0.1816
beta1	20000	0.0776	0.3276	-0.5690	0.7499
beta2	20000	1.3667	0.2923	0.8463	1.9724
beta3	20000	-0.8469	0.4718	-1.7741	0.0742
s2	20000	0.1171	0.0993	0.00163	0.3045

Example 80.8: Nonlinear Poisson Regression Multilevel Random-Effects Model

(View the complete code for this example (mcmcex8.sas) in the example repository.)

This example uses the pump failure data of Gaver and O'Muircheartaigh (1987) to illustrate how to fit a multilevel random-effects model with PROC MCMC. The number of failures and the time of operation are recorded for 10 pumps. Each of the pumps is classified into one of two groups that correspond to either continuous or intermittent operation. The following statements generate the data set:

```
title 'Nonlinear Poisson Regression Random-Effects Model';
data pump;
   input y t group @@;
   pump = _n_;
   logtstd = log(t) - 2.4564900;
   datalines;
 5  94.320 1   1  15.720 2    5  62.880 1
14 125.760 1   3   5.240 2   19  31.440 1
 1   1.048 2   1   1.048 2    4   2.096 2
22  10.480 2
;
```

Each row denotes data for a single pump, and the variable logtstd contains the centered operation times. Letting y_{ij} denote the number of failures for the jth pump in the ith group, Draper (1996) considers the

following hierarchical model for these data, where the data set variable logtstd is $\log t_{ij} - \overline{\log t}$:

$$\begin{aligned} y_{ij}|\lambda_{ij} &\sim \text{Poisson}(\lambda_{ij}) \\ \log \lambda_{ij} &= \alpha_i + \beta_i(\log t_{ij} - \overline{\log t}) + e_{ij} \end{aligned}$$

This model specifies different intercepts and slopes for each group ($i = 1, 2$), and the random effect e_{ij} is a mechanism for accounting for overdispersion. You can use noninformative priors on the parameters α_i, β_i, and σ^2, and a normal prior on e_{ij},

$$\begin{aligned} u_i &= \begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} \sim \text{mvn}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1e6 & 0 \\ 0 & 1e6 \end{pmatrix}\right) \quad \text{for } i = 1, 2 \\ \sigma^2 &\sim \text{igamma}\,(\text{shape} = 0.01, \text{scale} = 0.01) \\ e_{ij}|\sigma^2 &\sim \text{normal}(0, \sigma^2) \end{aligned}$$

where u_i is a multidimensional random effect. The following statements fit such a random-effects model:

```
ods graphics on;
proc mcmc data=pump outpost=postout seed=248601 nmc=10000
   plots=trace stats=none diag=none;
   ods select tracepanel;
   array u[2] alpha beta;
   array mu[2] (0 0);
   parms s2;
   prior s2 ~ igamma(0.01, scale=0.01);
   random u ~ MVNAR(mu, sd=1e6, rho=0) subject=group monitor=(u);
   random e ~ normal(0, var=s2) subject=pump monitor=(random(1));
   w = alpha + beta * logtstd;
   lambda = exp(w+e);
   model y ~ poisson(lambda);
run;
```

The PROC MCMC statement specifies the input data set (Pump), the output data set (Postout), a seed for the random number generator, and a simulation sample size of 10,000. The program requests that only trace plots be produced, disabling all posterior calculations and convergence diagnostics tests. The ODS SELECT statement displays the trace plots, which are the primary focus.

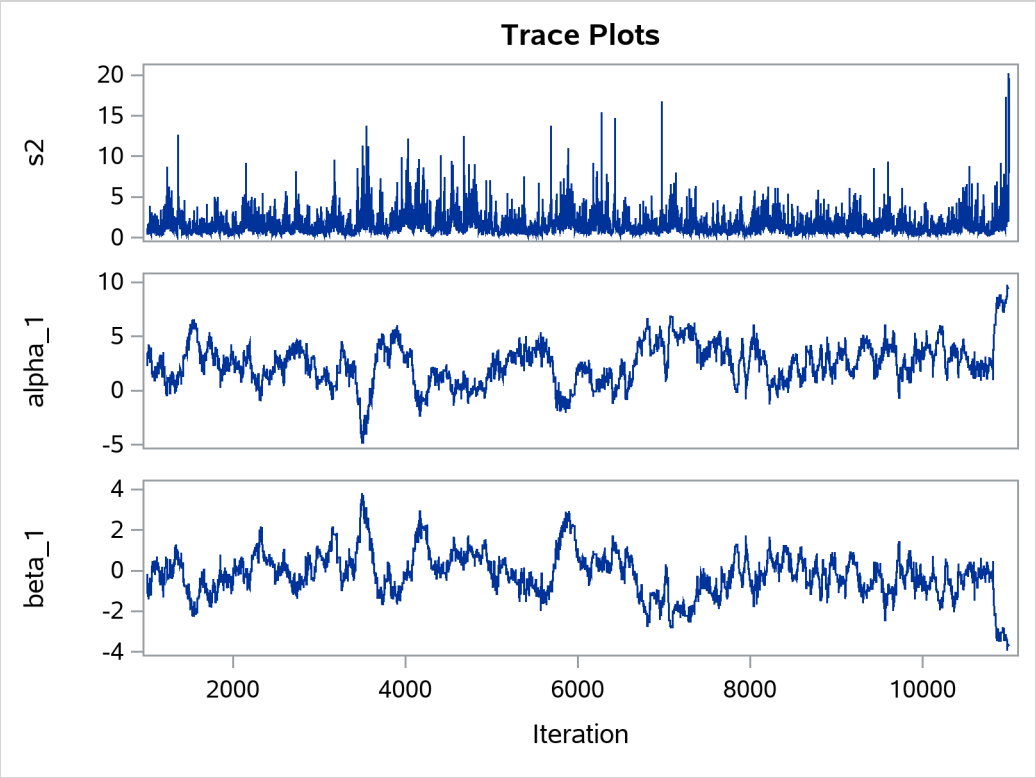
The first ARRAY statement declares an array u of size 2 and names the elements alpha and beta. The array u stores the random-effects parameters alpha and beta. The next ARRAY statement defines the mean of the multivariate normal prior on u.

The PARMS statement declares the only model parameter here, the variance s2 in the prior distribution for the random effect e_{ij}. The PRIOR statement specifies an inverse gamma prior on the variance. The first RANDOM statement specifies a multivariate normal prior on u. The MVNAR distribution is a multivariate normal distribution with a first-order autoregressive covariance. When the argument *rho* is set to 0, this distribution simplifies to a multivariate normal distribution with a shared variance. The RANDOM statement also indicates the group variable as its subject index and monitors all elements u. The second RANDOM statement specifies a normal prior on the effect e, where the subject index variable is pump. The MONITOR= option requests that PROC MCMC randomly choose one of the 10 e random-effects parameters to monitor.

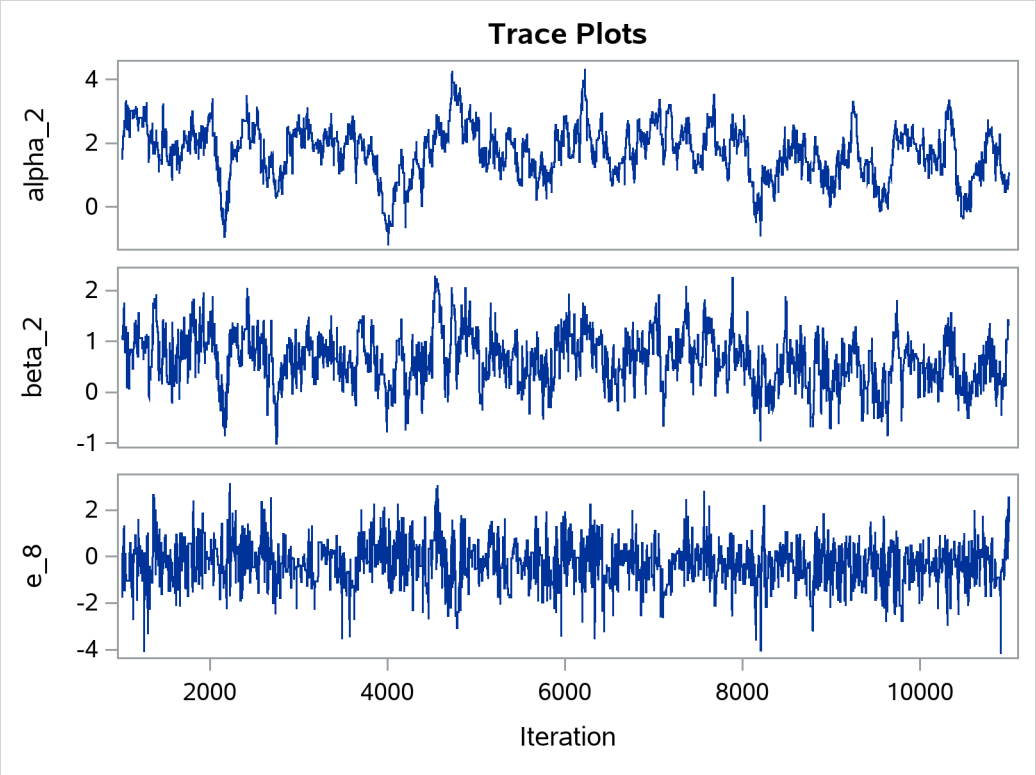
Next, programming statements construct the mean of the Poisson likelihood, and the MODEL statement specifies the likelihood function for each observation.

Output 80.8.1 shows trace plots for $\sigma^2, \alpha_1, \alpha_2, \beta_1, \beta_2,$ and e_8. You can see that the chains are mixing poorly.

Output 80.8.1 Trace Plots of σ^2, α, $\boldsymbol{\beta}$, and e_8 without Hierarchical Centering



Output 80.8.1 *continued*



To improve mixing, you can repeat the same analysis by using a hierarchical centering technique, where instead of using a normal prior centered at 0 on e_{ij}, you center the random effects on the group means:

$$\begin{aligned} y_{ij}|\lambda_{ij} &\sim \text{Poisson}(\lambda_{ij}) \\ \log \lambda_{ij} &\sim \text{normal}\left(\alpha_i + \beta_i(\log t_{ij} - \overline{\log t}), \text{var} = \sigma^2\right) \end{aligned}$$

The following statements illustrate how to fit a multilevel hierarchical centering random-effects model:

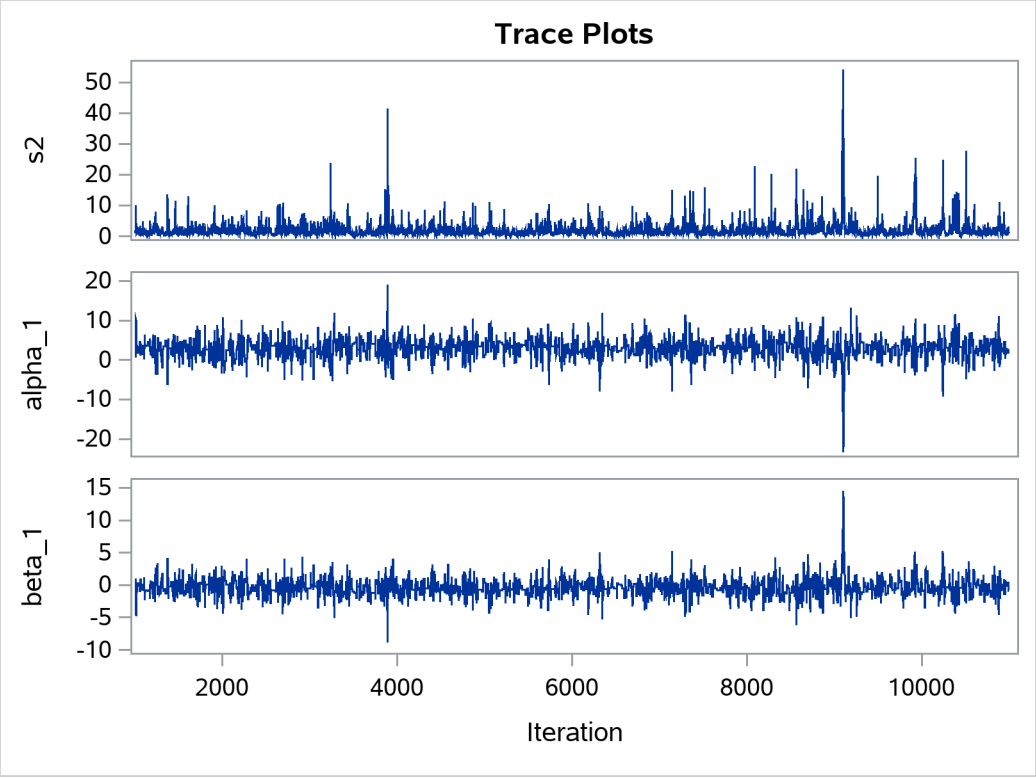
```
proc mcmc data=pump outpost=postout_c seed=248601 nmc=10000
   plots=trace diag=none;
   ods select tracepanel postsumint;
   array u[2] alpha beta;
   array mu[2] (0 0);
   parms s2 1;
   prior s2 ~ igamma(0.01, scale=0.01);
   random u ~ MVNAR(mu, sd=1e6, rho=0) subject=group monitor=(u);
   w = alpha + beta * logtstd;
   random llambda ~ normal(w, var = s2) subject=pump monitor=(random(1));
   lambda = exp(llambda);
   model y ~ poisson(lambda);
run;
```

The difference between these statements and the previous statements on page 6404 is that these statements have the variable w as the prior mean of the random effect llambda. The symbol lambda is the exponential of

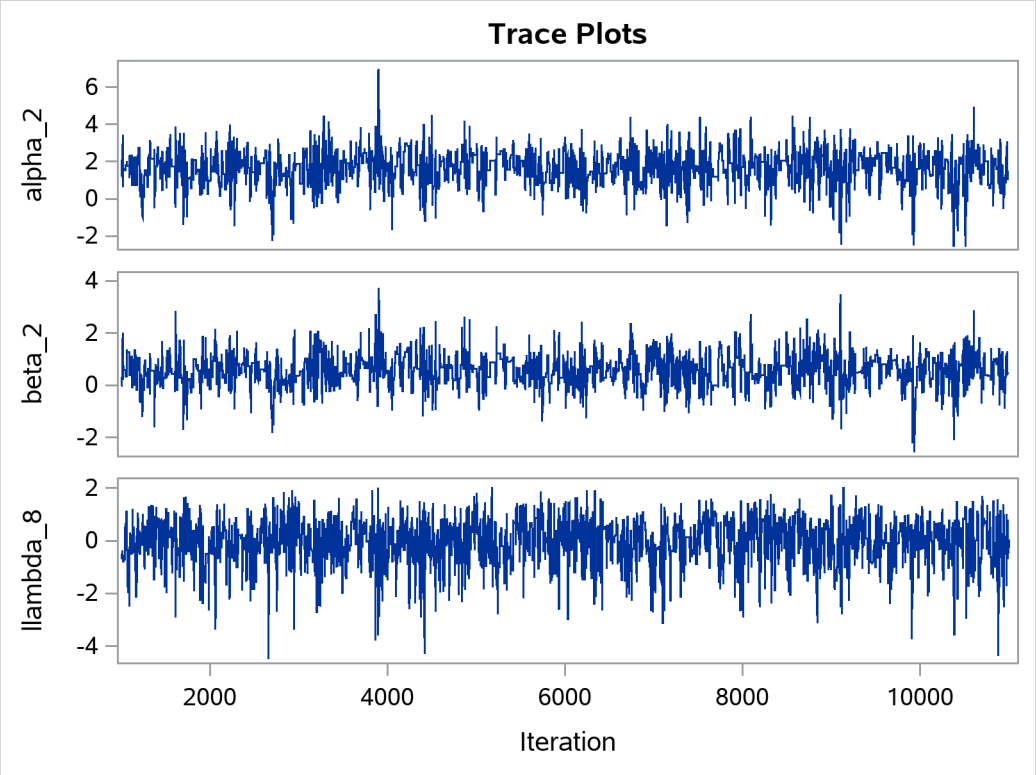
the corresponding $\log \lambda_{ij}$ (llambda), and the MODEL statement assigns the response variable y a Poisson likelihood with a mean parameter lambda, the same way it did in the previous statements.

The trace plots of the monitored parameters are shown in Output 80.8.2. The mixing is significantly improved over the previous model. The posterior summary and interval statistics tables are shown in Output 80.8.3.

Output 80.8.2 Trace Plots of σ^2, α, and $\boldsymbol{\beta}$ with Hierarchical Centering



Output 80.8.2 *continued*



Output 80.8.3 Posterior Summary Statistics

Nonlinear Poisson Regression Random-Effects Model

The MCMC Procedure

Posterior Summaries and Intervals

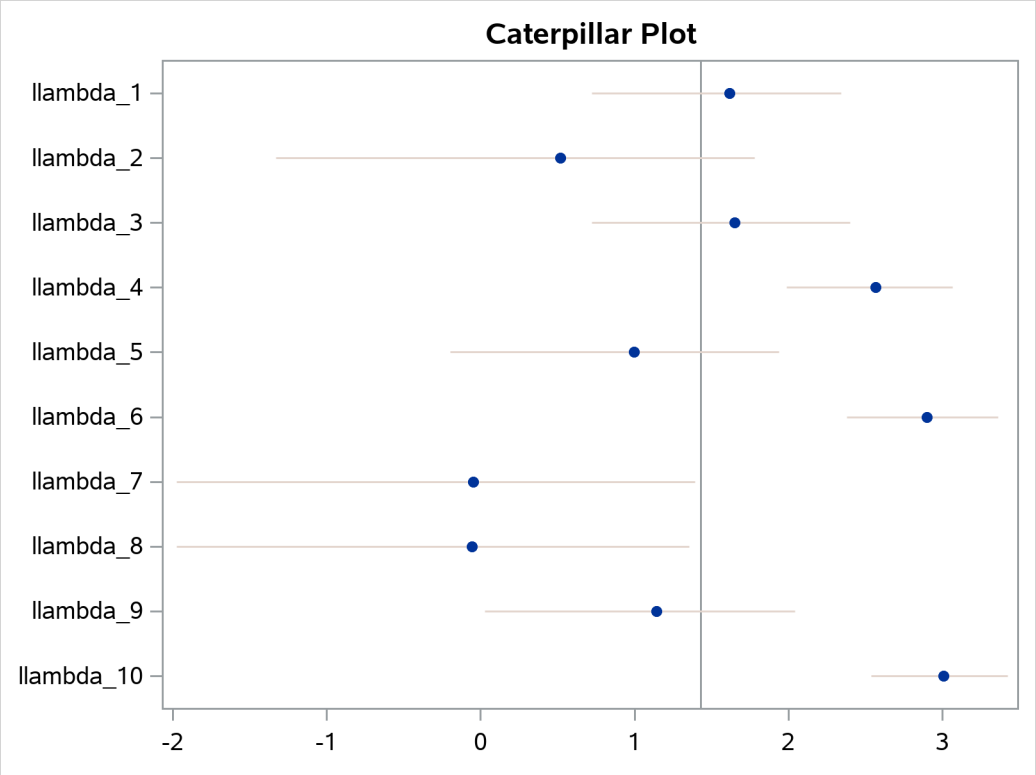
Parameter	N	Mean	Standard Deviation	95% HPD Interval	
s2	10000	1.7606	2.2022	0.1039	4.7631
alpha_1	10000	2.9286	2.4247	-1.9416	7.4115
beta_1	10000	-0.4018	1.3110	-2.9323	2.0623
alpha_2	10000	1.6105	0.8801	-0.0436	3.2985
beta_2	10000	0.5652	0.5804	-0.5469	1.7381
llambda_8	10000	-0.0560	0.8395	-1.6933	1.4612

You can generate a caterpillar plot (Output 80.8.4) of the random-effects parameters by calling the %CATER macro:

```
%CATER(data=postout_c, var=llambda_:);
ods graphics off;
```

Varying llambda indicates nonconstant dispersion in the Poisson model.

Output 80.8.4 Caterpillar Plots of the Random-Effects Parameters



Example 80.9: Multivariate Normal Random-Effects Model

(View the complete code for this example (mcmcex9.sas) in the example repository.)

Gelfand et al. (1990) use a multivariate normal hierarchical model to estimate growth regression coefficients for the growth of 30 young rats in a control group over a period of 5 weeks. The following statements create a SAS data set with measurements of Weight, Age (in days), and Subject.

```
title 'Multivariate Normal Random-Effects Model';
data rats;
   array days[5] (8 15 22 29 36);
   input weight @@;
   subject = ceil(_n_/5);
   index = mod(_n_-1, 5) + 1;
   age = days[index];
   drop index days:;
   datalines;
151 199 246 283 320 145 199 249 293 354
147 214 263 312 328 155 200 237 272 297
135 188 230 280 323 159 210 252 298 331
141 189 231 275 305 159 201 248 297 338
177 236 285 350 376 134 182 220 260 296
160 208 261 313 352 143 188 220 273 314
154 200 244 289 325 171 221 270 326 358
```

```
163 216 242 281 312 160 207 248 288 324
142 187 234 280 316 156 203 243 283 317
157 212 259 307 336 152 203 246 286 321
154 205 253 298 334 139 190 225 267 302
146 191 229 272 302 157 211 250 285 323
132 185 237 286 331 160 207 257 303 345
169 216 261 295 333 157 205 248 289 316
137 180 219 258 291 153 200 244 286 324
;
```

The model assumes normal measurement errors,

$$\text{Weight}_{ij} \sim \text{normal}\left(\alpha_i + \beta_i \text{Age}_{ij}, \sigma^2\right), \quad i = 1, \ldots, 30; j = 1, \ldots, 5$$

where i indexes rat (Subject variable), j indexes the time period, Weight_{ij} and Age_{ij} denote the weight and age of the ith rat in week j, and σ^2 is the variance in the normal likelihood. The individual intercept and slope coefficients are modeled as the following:

$$\boldsymbol{\theta}_i = \begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} \sim \text{MVN}\left(\boldsymbol{\theta}_c = \begin{pmatrix} \alpha_c \\ \beta_c \end{pmatrix}, \boldsymbol{\Sigma}_c\right), \quad i = 1, \ldots, 30$$

You can use the following independent prior distributions on $\boldsymbol{\theta}_c$, $\boldsymbol{\Sigma}_c$, and σ^2:

$$\begin{aligned}
\boldsymbol{\theta}_c &\sim \text{MVN}\left(\boldsymbol{\mu_0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \boldsymbol{\Sigma_0} = \begin{pmatrix} 1000 & 0 \\ 0 & 1000 \end{pmatrix}\right) \\
\boldsymbol{\Sigma}_c &\sim \text{iwishart}\left(\rho = 2, \mathbf{S} = \rho \cdot \begin{pmatrix} 0.01 & 0 \\ 0 & 10 \end{pmatrix}\right) \\
\sigma^2 &\sim \text{igamma}\left(\text{shape} = 0.01, \text{scale} = 0.01\right)
\end{aligned}$$

The following statements fit this multivariate normal random-effects model:

```
proc mcmc data=rats nmc=10000 outpost=postout
   seed=17 init=random;
   ods select Parameters REParameters PostSumInt;
   array theta[2] alpha beta;
   array theta_c[2];
   array Sig_c[2,2];
   array mu0[2] (0 0);
   array Sig0[2,2] (1000 0 0 1000);
   array S[2,2] (0.02 0 0 20);

   parms theta_c Sig_c {121 0 0 0.26} var_y;
   prior theta_c ~ mvn(mu0, Sig0);
   prior Sig_c ~ iwish(2, S);
   prior var_y ~ igamma(0.01, scale=0.01);

   random theta ~ mvn(theta_c, Sig_c) subject=subject
      monitor=(alpha_9 beta_9 alpha_25 beta_25);
   mu = alpha + beta * age;
   model weight ~ normal(mu, var=var_y);
run;
```

The ODS SELECT statement displays information about model parameters, random-effects parameters, and the posterior summary statistics. The ARRAY statements allocate memory space for the multivariate parameters and hyperparameters in the model. The parameters are $\boldsymbol{\theta}$ (theta where the variable name of each element is alpha or beta), $\boldsymbol{\theta}_c$ (theta_c), and $\boldsymbol{\Sigma}_c$ (Sig_c). The hyperparameters are $\boldsymbol{\mu_0}$ (mu0), $\boldsymbol{\Sigma_0}$ (Sig0), and $\mathbf{S}$ (S). The multivariate hyperparameters are assigned with constant values using parentheses ( ).

The PARMS statement declares model parameters and assigns initial values to Sig_c using braces { }. The PRIOR statements specify the prior distributions. The RANDOM statement defines an array random effect theta and specifies a multivariate normal prior distribution. The SUBJECT= option indicates cluster membership for each of the random-effects parameter. The MONITOR= option monitors the individual intercept and slope coefficients of subjects 9 and 25.

You can use the following syntax in the RANDOM statement to monitor all parameters in an array random effect:

```
monitor=(theta)
```

This would produce posterior summary statistics on $\alpha_1 \cdots \alpha_{30}$ and $\beta_1 \cdots \beta_{30}$.

The following syntax monitors all α_i parameters:

```
monitor=(alpha)
```

If you did not name elements of theta to be alpha and beta, the SAS System creates variable names automatically in a consecutive fashion, as in theta1 and theta2.

Output 80.9.1 Parameter and Random-Effects Parameter Information Table

Multivariate Normal Random-Effects Model

The MCMC Procedure

Parameters

Block	Parameter	Array Index	Sampling Method	Initial Value	Prior Distribution
1	theta_c1		Conjugate	-4.5834	MVNormal(mu0, Sig0)
	theta_c2			5.7930	
2	Sig_c1	[1,1]	Conjugate	121.0	iWishart(2, S)
	Sig_c2	[1,2]		0	
	Sig_c3	[2,1]		0	
	Sig_c4	[2,2]		0.2600	
3	var_y		Conjugate	2806714	igamma(0.01, scale=0.01)

Random Effect Parameters

Parameter	Sampling Method	Subject	Number of Subjects	Subject Values	Prior Distribution
theta	N-Metropolis	subject	30	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ...	MVNormal(theta_c, Sig_c)

Output 80.9.1 displays the parameter and random-effects parameter information tables. The Array Index column in "Parameters" table shows the index reference of the elements in the array parameter Sig_c. The total number of subjects in the study is 30.

Output 80.9.2 Multivariate Normal Random-Effects Model

Multivariate Normal Random-Effects Model

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
theta_c1	10000	106.1	2.2486	101.7	110.6
theta_c2	10000	6.1975	0.1988	5.8058	6.5815
Sig_c1	10000	110.8	45.9169	37.9670	203.8
Sig_c2	10000	-1.4267	2.3320	-6.2878	2.7756
Sig_c3	10000	-1.4267	2.3320	-6.2878	2.7756
Sig_c4	10000	1.0591	0.2979	0.5549	1.6538
var_y	10000	37.6855	5.9591	27.0943	49.4449
alpha_9	10000	119.4	5.6756	108.1	130.5
beta_9	10000	7.4670	0.2382	7.0146	7.9278
alpha_25	10000	86.5673	6.3694	74.4247	99.9007
beta_25	10000	6.7804	0.2612	6.2529	7.2906

Output 80.9.2 displays posterior summary statistics for model parameters and the random-effects parameters for subjects 9 and 25. You can see that there is a substantial difference in the intercepts and growth rates between the two rats.

A seemingly confusing message might occur if a symbol name matches an internally generated variable name for elements of an array. For example, if, instead of using the symbol var_y in the SAS program for the model variance σ^2, you used s2, the SAS System produces the following error message:

```
ERROR: The initial value 0 for the parameter S2 is outside of the prior
       distribution support set.
```

This is confusing because the program does not assign an initial value for the parameter s2 in the PARMS statement, and you might expect that PROC MCMC would not generate an invalid initial value. The confusion is caused by the ARRAY statement that defines the array variable S:

```
array S[2,2] (0.02 0 0 20);
```

Elements of S are automatically given names s1–s4. PROC MCMC interprets s2 as an element in S that was given a value of 0, hence producing this error message.

Example 80.10: Missing at Random Analysis

(View the complete code for this example (mcmcex10.sas) in the example repository.)

This example illustrates how PROC MCMC treats missing at random (MAR) data. For a short overview of missing data problems, see the section "Handling of Missing Data" on page 6338.

Researchers studied the effects of air pollution on respiratory disease in children. The response variable (y) represented whether a child exhibited wheezing symptoms; it was recorded as 1 for symptoms exhibited and 0 for no symptoms exhibited. City of residency (x1) and maternal smoking status (x2) were the explanatory variables. The variable x1 was coded as 1 if the child lived in the more polluted city, Steel City, and 0 if the child lived in Green Hills. The variable x2 was the number of cigarettes the mother reported that she smoked per day. Both the covariates contain missing values: 17 for x1 and 30 for x2, respectively. The total number of observations in the data set is 390. The following statements generate the data set air:

```
title 'Missing at Random Analysis';
data air;
   input y x1 x2 @@;
   datalines;
 0  0  0   0  0  0   0  1  0   0  0  0   0  0 11   0  1  7
 0  0  8   0  1 10   0  1  9   0  0  0   1  1  6   0  1 10
 0  1 12   0  0  .   0  0  0   0  1  0   0  1  7   1  1 15
 0  0  8   0  0  0   1  1  0   1  0  6   0  0  0   1  1 11
 0  0  0   1  0  0   1  0  5   0  0  8   0  0  0   0  1  9

   ... more lines ...

 0  0 11   0  0  0   0  0  6   0  0 12   0  0 10   0  1 10
 0  1 11   0  0  9   1  0 11   0  1  7   0  0  7   0  0  0
 0  . 11   1  1  6   0  0  8   0  0  0   0  1 12   0  0  0
 0  1  0   1  1  8   0  0  0   0  1 11   0  1  0   0  1  8
 0  .  0   1  0  0   1  1 10   0  .  4   1  1 16   0  . 13
;
```

Suppose you want to fit a logistic regression model for whether the subject develops wheezing symptoms with density for the $i = 1, \ldots, 390$ subjects as follows:

$$\begin{aligned} \mathsf{y}_i &\sim \text{binary}(p_i) \\ \text{logit}(p_i) &= \beta_0 + \beta_1 \cdot \mathsf{x1}_i + \beta_2 \cdot \mathsf{x2}_i \\ \pi(\beta_0), \pi(\beta_1), \pi(\beta_2) &= \text{normal}(0, \sigma^2 = 10) \end{aligned}$$

Suppose you specify a joint distribution of x1 and x2 in terms of the product of a conditional and marginal distribution; that is,

$$p(\mathsf{x1}, \mathsf{x2}|\boldsymbol{\alpha}) = p(\mathsf{x1}|\mathsf{x2}, \alpha_{10}, \alpha_{11})\, p(\mathsf{x2}|\alpha_{20})$$

where $p(\mathsf{x1}_i|\mathsf{x2}_i, \alpha_{10}, \alpha_{11})$ could be a logistic model and $p(\mathsf{x2}_i|\alpha_{20})$ could be a Poisson distribution that models the following counts:

$$\begin{aligned} p(\mathsf{x1}_i|\mathsf{x2}_i, \alpha_{10}, \alpha_{11}) &= \text{binary}(p_{c,i}) \\ \text{logit}(p_{c,i}) &= \alpha_{10} + \alpha_{11} \cdot \mathsf{x2}_i \\ \pi(\alpha_{10}), \pi(\alpha_{11}) &= \text{normal}(0, \sigma^2 = 10) \\ p(\mathsf{x2}_i|\alpha_{20}) &= \text{Poisson}(e^{\alpha_{20}}) \\ \pi(\alpha_{20}) &= \text{normal}(0, \sigma^2 = 2) \end{aligned}$$

The researchers are interested in interpreting how the odds of developing a wheeze changes for a child living in the more polluted city. The odds ratio can be written as the follows:

$$\text{OR}_{\mathsf{x1}} = \exp(\beta_1)$$

Similarly, the odds ratio for the maternal smoking effect can be written as follows:

$$\text{OR}_{\mathsf{x2}} = \exp(\beta_2)$$

The following statements fit a Bayesian logistic regression with missing covariates:

```
proc mcmc data=air seed=1181 nmc=10000 monitor=(_parms_ orx1 orx2)
   diag=none plots=none;
   parms beta0 -1 beta1 0.1 beta2 .01;
   parms alpha10 0 alpha11 0 alpha20 0;

   prior beta: alpha1: ~ normal(0,var=10);
   prior alpha20 ~ normal(0,var=2);

   beginnodata;
   pm = exp(alpha20);
   orx1 = exp(beta1);
   orx2 = exp(beta2);
   endnodata;
   model x2 ~ poisson(pm) monitor=(1 3 10);
   p1 = logistic(alpha10 + alpha11 * x2);
   model x1 ~ binary(p1) monitor=(random(3));
   p = logistic(beta0 + beta1*x1 + beta2*x2);
   model y ~ binary(p);
run;
```

The PARMS statements specify the parameters in the model and assign initial values to each of them. The PRIOR statements specify priors for all the model parameters. The notations beta: and alpha: in the PRIOR statements are shorthand for all variables that start with "beta," and "alpha," respectively. The shorthand notation is not necessary, but it keeps your code succinct.

The BEGINNODATA and ENDNODATA statements enclose three programming statements that calculate the Poisson mean (pm) and the two odds ratios (ORX1 and ORX2). These enclosed statements are independent of any data set variables, and they are run only once per iteration to reduce unnecessary observation-level computations.

The first MODEL statement assigns a Poisson likelihood with mean pm to x2. The statement models missing values in x2 automatically, creating one variable for each of the missing values, and augments them accordingly. By default, PROC MCMC does not output analyses of the posterior samples of the missing values. You can use the MONITOR= option to choose the missing values that you want to monitor. In the example, the first, third, and tenth missing values are monitored.

The P1 assignment statement calculates $p_{c.i}$. The second MODEL statement assigns a binary likelihood with probability p1 and requests a random choice of three missing data variables of x1 to monitor.

The P assignment statement calculates p_i in the logistic model. The third MODEL statement specifies the complete data likelihood function for Y.

Output 80.10.1 displays the number of observations read from the DATA= data set, the number of observations used in the analysis, and the "Missing Data Information" table. No observations were omitted from the data set in the analysis.

The "Missing Data Information" table lists the variables that contain missing values, which are x1 and x2, the number of missing observations in each variable, the observation indices of these missing values, and the sampling algorithms used. By default, the first 20 observation indices of each variable are printed in the table.

Output 80.10.1 Observation Information and Missing Data Information

Missing at Random Analysis

The MCMC Procedure

Number of Observations Read	390
Number of Observations Used	390

Missing Data Information Table

Variable	Number of Missing Obs	Observation Indices	Sampling Method
x2	30	14 41 50 55 59 66 71 83 88 90 118 158 174 175 178 183 196 203 210 212 ...	N-Metropolis
x1	17	50 92 93 167 194 231 273 296 303 304 308 330 349 373 385 388 390	Inverse CDF

There are 30 missing values in the variable x2, and 17 in x1. Internally, PROC MCMC creates 30 and 17 variables for the missing values in x2 and x1, respectively. The default naming convention for these missing values is to concatenate the response variable and the observation number. For example, the first missing value in x2 is the fourteenth observation, and the corresponding variable is x2_14.

Output 80.10.2 displays the summary and interval statistics for each parameter, the odds ratios, and the monitored missing values.

Output 80.10.2 Posterior Summary and Interval Statistics

Missing at Random Analysis

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta0	10000	-1.3732	0.2078	-1.7909	-0.9676
beta1	10000	0.4797	0.2387	0.0268	0.9491
beta2	10000	0.0156	0.0227	-0.0265	0.0642
alpha10	10000	-0.2166	0.1422	-0.4662	0.0874
alpha11	10000	0.0126	0.0201	-0.0267	0.0521
alpha20	10000	1.5635	0.0235	1.5199	1.6105
orx1	10000	1.6627	0.4094	0.9205	2.4493
orx2	10000	1.0160	0.0231	0.9739	1.0663
x2_14	10000	4.9022	2.2083	1.0000	9.0000
x2_50	10000	4.8924	2.1626	1.0000	9.0000
x2_90	10000	4.8263	2.0816	1.0000	8.0000
x1_296	10000	0.4160	0.4929	0	1.0000
x1_304	10000	0.4460	0.4971	0	1.0000
x1_373	10000	0.4443	0.4969	0	1.0000

The odds ratio for x1 is the multiplicative change in the odds of a child wheezing in Steel City compared to the odds of the child wheezing in Green Hills. The estimated odds ratio (ORX1) value is 1.6736 with a corresponding 95% equal-tail credible interval of (1.0248, 2.5939). City of residency is a significant factor in a child's wheezing status. The estimated odds ratio for x2 is the multiplicative change in the odds of developing a wheeze for each additional reported cigarette smoked per day. The odds ratio of ORX2 indicates that the odds of a child developing a wheeze is 1.0150 times higher for each reported cigarette a mother smokes. The corresponding 95% equal-tail credible interval is (0.9695, 1.0619). Since this interval contains the value 1, maternal smoking is not considered to be an influential effect.

Example 80.11: Nonignorably Missing Data (MNAR) Analysis

(View the complete code for this example (mcmcex11.sas) in the example repository.)

This example illustrates how to fit a nonignorably missing data model (MNAR) with PROC MCMC. For a short overview of missing data problems, see the section "Handling of Missing Data" on page 6338.

This data set comes from an environmental study that involve workers in a cotton factory. A similar data set was analyzed from Ibrahim, Chen, and Lipsitz (2001). There are 912 workers in the data set, and the response variable of interest is whether they develop dyspnea (difficult or labored respiration). The data are collected over three time points, and there are six covariates. The following statements create the data set:

```
title 'Nonignorably Missing Data Analysis';
data dyspnea;
   input smoke1 smoke2 smoke3 y1 y2 y3 yrswrk1 yrswrk2 yrswrk3
      age expd sex hgt;
   datalines;
```

```
 0  0  0  0  0  0  28.1  33.1  39.1  48 1 1 165.0
 0  0  0  0  .  0   5.1  10.1  16.1  45 1 0 147.0
 0  0  0  0  .  0  26.0  31.0  37.0  46 1 0 156.0

   ... more lines ...

 1  1  1  0  .  .   6.0  11.0  17.0  25 0 1 180.0
 0  0  0  0  .  .  20.0  25.0  31.0  42 0 0 159.0
;
```

The following variables are included in the data set:

- y1, y2, and y3: dichotomous outcome at the three time periods, which takes the value 1 if the worker has dyspnea, 0 if not (there are missing values in y2 and y3)
- smoke1, smoke2, smoke3: smoking status (0=no, and 1=yes)
- yrswrk1, yrswrk2, yrswrk3: years worked at the cotton factory
- age: age of the worker
- expd: cotton dust exposure (0=no, 1=yes)
- sex: gender (0=female, 1=male)
- hgt: height of the worker

Prior to the analysis, three missing data indicator variables (r1, r2, and r3, one for each of the response variables) are created, and they are set to 1 if the response variable is missing, and 0 otherwise. The covariates age, hgt, yrswrk1, yrswkr2, and yrswrk3 are standardized:

```
data dyspnea;
   array y[3] y1-y3;
   array r[3];
   set dyspnea;
   do i = 1 to 3;
      if y[i] = . then r[i] = 1;
      else r[i] = 0;
   end;
   output;
run;

proc standard data=dyspnea out=dyspnea mean=0 std=1;
   var age hgt yrswrk:;
run;
```

There are no missing values in response variable y1, 128 missing values in y2, and 131 in y3. Ibrahim, Chen, and Lipsitz (2001) used a logistic regression for each of the response variables, where δ_i is a scalar random effect on the observational level:

$$
\begin{aligned}
\mathrm{y}_{ki} &\sim \text{binary}(p_{ki}) \quad k = 1,2,3; \quad i = 1,\ldots,912 \\
p_{ki} &= \text{logistic}(\mu_{ki} + \delta_i) \\
\mu_{ki} &= \beta_1 + \beta_2 \cdot \text{expd}_i + \beta_3 \cdot \text{sex}_i + \beta_4 \cdot \text{hgt}_i + \beta_5 \cdot \text{age}_i + \beta_6 \cdot \text{yrswrk}_{ki} + \beta_7 \cdot \text{smoke}_{ki} \\
\delta_i &\sim \text{n}(0, \sigma^2)
\end{aligned}
$$

Ibrahim, Chen, and Lipsitz (2001) noted that taking δ_i to be higher dimensional (3) would make the model either not identifiable or nearly not identifiable because of the multiple missing values for some subjects.

The first response variable y1 does not contain any missing values, making it meaningless to model the corresponding r1 because every value is 1. Hence, only r2 and r3 are considered in the missing mechanism part of the model. Ibrahim, Chen, and Lipsitz (2001) suggest the following logistic regression for r2 and r3, where the regression mean for each r depends not only on the current response variable y but also the response from previous time period:

$$
\begin{aligned}
\mathsf{r}_{ki} &\sim \text{binary}(q_{ki}) \quad k = 2,3; \quad i = 1,\ldots,912 \\
q_{ki} &= \text{logistic}(\nu_{ki}) \\
\mu_c &= \phi_1 + \phi_2 \cdot \text{expd}_i + \phi_3 \cdot \text{sex}_i + \phi_4 \cdot \text{hgt}_i + \phi_5 \cdot \text{age}_i + \phi_6 \cdot \text{yrswrk}_{ki} + \phi_7 \cdot \text{smoke}_{ki} \\
\nu_{2i} &= \mu_c + \phi_8 \cdot \mathsf{y}_{1i} + \phi_9 \cdot \mathsf{y}_{2i} \\
\nu_{3i} &= \mu_c + \phi_9 \cdot \mathsf{y}_{2i} + \phi_{10} \cdot \mathsf{y}_{3i}
\end{aligned}
$$

The missing mechanism model introduces an additional 10 parameters to the model. Normal priors with large standard deviations are used here.

The following statements fit a nonignorably missing model to the dyspnea data set:

```
ods select MissDataInfo REParameters Postsumint;
proc mcmc data=dyspnea seed=17 outpost=dysp2 nmc=20000
     propcov=simplex diag=none monitor=(beta1-beta7);
   array p[3];
   array yrswrk[3];
   array smoke[3];

   parms beta1-beta7 s2;
   parms phi1-phi10;
   prior beta: phi: ~ n(0, var=1e6);
   prior s2 ~ igamma(2, scale=2);
   random d ~ n(0, var=s2) subject=_obs_;
   mu = beta1 + beta2*expd + beta3*sex + beta4*hgt + beta5*age + d;
   do j = 1 to 3;
      p[j] = logistic(mu + beta6*yrswrk[j] + beta7*smoke[j]);
   end;
   model y1 ~ binary(p1);
   model y2 ~ binary(p2);
   model y3 ~ binary(p3);

   nu = phi1 + phi2*expd + phi3*sex + phi4*hgt + phi5*age;
   q2 = logistic(nu + phi6*yrswrk[2] + phi7*smoke[2] +
                 phi8*y1 + phi9*y2);
   model r2 ~ binary(q2);
   q3 = logistic(nu + phi6*yrswrk[3] + phi7*smoke[3] +
                 phi9*y2 + phi10*y3);
   model r3 ~ binary(q3);
run;
```

The first ARRAY statement declares an array p of size 3. This arrays stores three binary probabilities of the response variables. The next two ARRAY statements create storage arrays for some of yrswrk and smoke variables for later programming convenience. The first PARMS statement declares eight parameters, $\beta_1 - \beta_7$

and σ^2. The second PARMS statement declares the 10 ϕ parameters for the missing mechanism model. The PRIOR statements assign prior distributions to these parameters.

The RANDOM statement defines an observational-level random effect d that has a normal prior with variance s2. The SUBJECT=_OBS_ option enables the specification of individual random effects without an explicit input data set variable.

The MU assignment statement and the following DO loop statements calculate the binary probabilities for the three response variables. Note that different yrswrk and smoke variables are used in the DO loop for different years. The three MODEL statements assign three binary distributions to the response variables.

The NU assignment statement starts the calculation for the regression mean in the logistic model for r2 and r3. The variables q2 and q3 are the binary probabilities for the missing mechanisms. Note that their calculations are conditional on the response variables y (selection model). The last two MODEL statements for r2 and r3 complete the specification of the models.

Missing data information and random-effects parameters information are displayed in Output 80.11.1. You can read the total number of missing observations from each variable and its indices from the table. The missing values are sampled using the inverse CDF method. There are 912 random-effects parameters in the model.

Output 80.11.1 Missing Data and Random-Effects Information

Nonignorably Missing Data Analysis

The MCMC Procedure

Missing Data Information Table

Variable	Number of Missing Obs	Observation Indices	Sampling Method
y2	128	2 3 9 11 13 19 20 21 30 31 32 35 39 40 43 56 58 71 75 95 ...	Inverse CDF
y3	131	9 14 16 20 21 29 31 32 43 45 56 72 86 115 117 121 124 142 149 160 ...	Inverse CDF

Random Effect Parameters

Parameter	Sampling Method	Subject	Number of Subjects	Subject Values	Prior Distribution
d	N-Metropolis	_OBS_	912	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ...	normal(0, var=s2)

The posterior summary and interval statistics of all the β parameters are shown in Output 80.11.2. There are a number of significant regression coefficients in modeling the probability of a worker developing dyspnea, including those for expd (β_2), sex (β_3), age (β_5), and smoke (β_7).

Output 80.11.2 Posterior Summary Statistics for β

Nonignorably Missing Data Analysis

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta1	20000	-2.3365	0.1752	-2.6819	-1.9912
beta2	20000	0.5367	0.1523	0.2570	0.8335
beta3	20000	-0.5920	0.2560	-1.0681	-0.0768
beta4	20000	-0.0660	0.1044	-0.2728	0.1437
beta5	20000	0.6210	0.1730	0.3005	0.9720
beta6	20000	-0.1700	0.1651	-0.4627	0.1833
beta7	20000	0.5799	0.2186	0.1521	1.0029

Example 80.12: Change Point Models

(View the complete code for this example (mcmcex12.sas) in the example repository.)

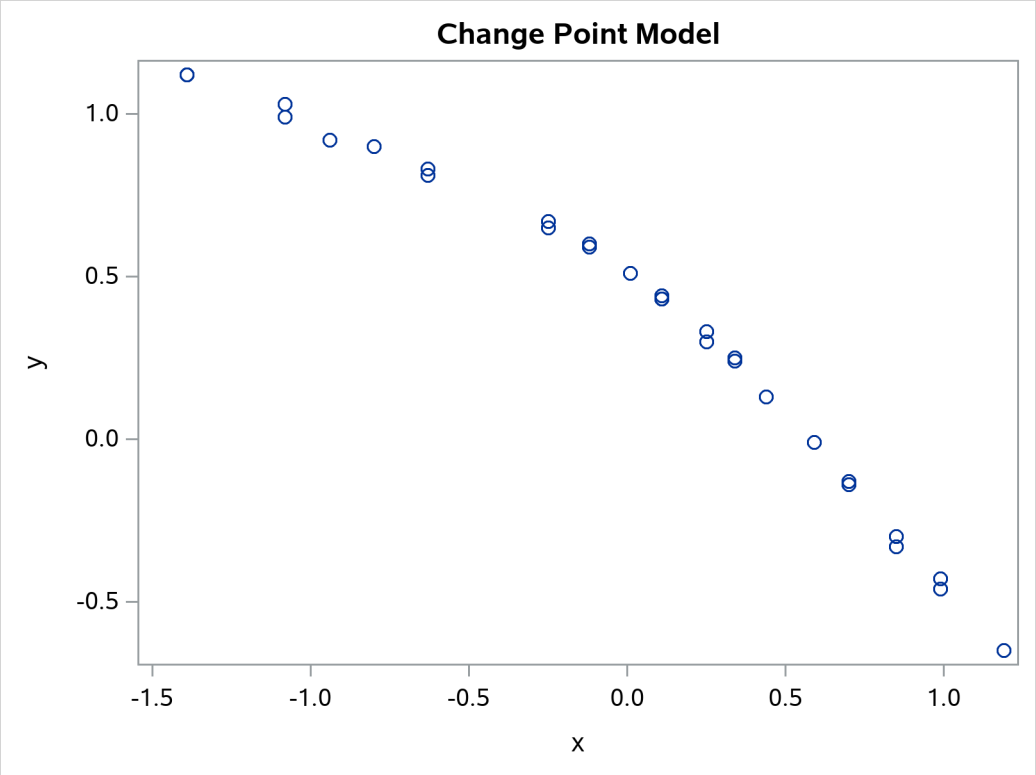
Consider the data set from Bacon and Watts (1971), where y_i is the logarithm of the height of the stagnant surface layer and the covariate x_i is the logarithm of the flow rate of water. The following statements create the data set:

```
title 'Change Point Model';
data stagnant;
   input y x @@;
   ind = _n_;
   datalines;
 1.12  -1.39   1.12  -1.39   0.99  -1.08   1.03  -1.08
 0.92  -0.94   0.90  -0.80   0.81  -0.63   0.83  -0.63
 0.65  -0.25   0.67  -0.25   0.60  -0.12   0.59  -0.12
 0.51   0.01   0.44   0.11   0.43   0.11   0.43   0.11
 0.33   0.25   0.30   0.25   0.25   0.34   0.24   0.34
 0.13   0.44  -0.01   0.59  -0.13   0.70  -0.14   0.70
-0.30   0.85  -0.33   0.85  -0.46   0.99  -0.43   0.99
-0.65   1.19
;
```

A scatter plot (Output 80.12.1) shows the presence of a nonconstant slope in the data. This suggests a change point regression model (Carlin, Gelfand, and Smith 1992). The following statements generate the scatter plot in Output 80.12.1:

```
proc sgplot data=stagnant;
   scatter x=x y=y;
run;
```

Output 80.12.1 Scatter Plot of the Stagnant Data Set



Let the change point be cp. Following formulation by Spiegelhalter et al. (1996b), the regression model is as follows:

$$y_i \sim \begin{cases} \text{normal}(\alpha + \beta_1(x_i - \text{cp}), \sigma^2) & \text{if } x_i < \text{cp} \\ \text{normal}(\alpha + \beta_2(x_i - \text{cp}), \sigma^2) & \text{if } x_i >= \text{cp} \end{cases}$$

You might consider the following diffuse prior distributions:

$$\begin{aligned} \text{cp} &\sim \text{uniform}(-1.3, 1.1) \\ \alpha, \beta_1, \beta_2 &\sim \text{normal}(0, \text{var} = 1e6) \\ \sigma^2 &\sim \text{uniform}(0, 5) \end{aligned}$$

The following statements generate Output 80.12.2:

```
proc mcmc data=stagnant outpost=postout seed=24860 ntu=1000
          nmc=20000;
   ods select PostSumInt;
   ods output PostSumInt=ds;

   array beta[2];
   parms alpha cp beta1 beta2;
   parms s2;

   prior cp ~ unif(-1.3, 1.1);
   prior s2  ~ uniform(0, 5);
```

```
   prior alpha beta:  ~ normal(0, v = 1e6);

   j = 1 + (x >= cp);
   mu = alpha + beta[j] * (x - cp);
   model y ~ normal(mu, var=s2);
run;
```

The PROC MCMC statement specifies the input data set (Stagnant), the output data set (Postout), a random number seed, a tuning sample of 1000, and an MCMC sample of 20000. The ODS SELECT statement displays only the summary statistics table. The ODS OUTPUT statement saves the summary statistics table to the data set Ds.

The ARRAY statement allocates an array of size 2 for the beta parameters. You can use beta1 and beta2 as parameter names without allocating an array, but having the array makes it easier to construct the likelihood function. The two PARMS statements put the five model parameters in two blocks. The three PRIOR statements specify the prior distributions for these parameters.

The symbol j indicates the segment component of the regression. When x is less than the change point, `(x >= cp)` returns 0 and j is assigned the value 1; if x is greater than or equal to the change point, `(x >= cp)` returns 1 and j is 2. The symbol mu is the mean for the jth segment, and beta[j] changes between the two regression coefficients depending on the segment component. The MODEL statement assigns the normal model to the response variable y.

Posterior summary statistics are shown in Output 80.12.2.

Output 80.12.2 MCMC Estimates of the Change Point Regression Model

Change Point Model

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
alpha	20000	0.5349	0.0249	0.4843	0.5813
cp	20000	0.0283	0.0314	-0.0353	0.0846
beta1	20000	-0.4200	0.0146	-0.4482	-0.3911
beta2	20000	-1.0136	0.0167	-1.0476	-0.9817
s2	20000	0.000451	0.000145	0.000220	0.000735

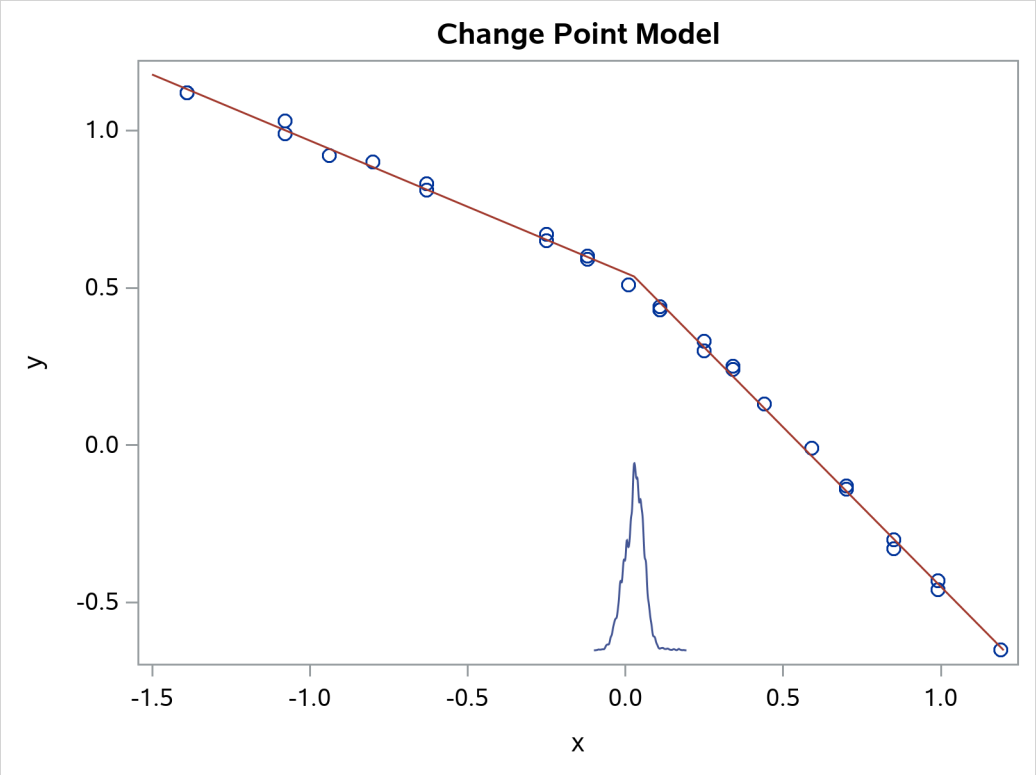
You can use PROC SGPLOT to visualize the model fit. Output 80.12.3 shows the fitted regression lines over the original data. In addition, on the bottom of the plot is the kernel density of the posterior marginal distribution of cp, the change point. The kernel density plot shows the relative variability of the posterior distribution on the data plot. You can use the following statements to create the plot:

```
data _null_;
   set ds;
   call symputx(parameter, mean);
run;

data b;
   missing A;
   input x1 @@;
```

```
   if x1 eq .A then x1 = &cp;
   if _n_ <= 2 then y1 = &alpha + &beta1 * (x1 - &cp);
   else y1 = &alpha + &beta2 * (x1 - &cp);
   datalines;
   -1.5 A 1.2
;

proc kde data=postout;
   univar cp / out=m1 (drop=count);
run;

data m1;
   set m1;
   density = (density / 25) - 0.653;
run;

data all;
   set stagnant b m1;
run;

proc sgplot data=all noautolegend;
   scatter x=x y=y;
   series x=x1 y=y1 / lineattrs = graphdata2;
   series x=value y=density / lineattrs = graphdata1;
run;
```

The macro variables &alpha, &beta1, &beta2, and &cp store the posterior mean estimates from the data set Ds. The data set b contains three predicted values, at the minimum and maximum values of x and the estimated change point &cp. These input values give you fitted values from the regression model. Data set M1 contains the kernel density estimates of the parameter cp. The density is scaled down so the curve would fit in the plot. Finally, you use PROC SGPLOT to overlay the scatter plot, regression line and kernel density plots in the same graph.

Output 80.12.3 Estimated Fit to the Stagnant Data Set



Example 80.13: Exponential and Weibull Survival Analysis

(View the complete code for this example (mcmcex13.sas) in the example repository.)

This example covers two commonly used survival analysis models: the exponential model and the Weibull model. The deviance information criterion (DIC) is used to do model selections, and you can also find programs that visualize posterior quantities. Exponential and Weibull models are widely used for survival analysis. This example shows you how to use PROC MCMC to analyze the treatment effect for the E1684 melanoma clinical trial data. These data were collected to assess the effectiveness of using interferon alpha-2b in chemotherapeutic treatment of melanoma. The following statements create the data set:

```
data e1684;
   input t t_cen treatment @@;
   if t = . then do;
      t = t_cen;
      v = 0;
   end;
   else
      v = 1;
   ifn = treatment - 1;
   et = exp(t);
   lt = log(t);
   drop t_cen;
   datalines;
```

```
1.57808  0.00000  2   1.48219  0.00000  2    .       7.33425  1
2.23288  0.00000  1    .       9.38356  2   3.27671  0.00000  1
 .       9.64384  1   1.66575  0.00000  2   0.94247  0.00000  1
1.68767  0.00000  2   2.34247  0.00000  2   0.89863  0.00000  1

  ... more lines ...

3.39178  0.00000  1   .        4.36164  2   .        4.81918  2
;
```

The data set E1684 contains the following variables: t is the failure time that equals the censoring time whether the observation was censored, v indicates whether the observation is an actual failure time or a censoring time, treatment indicates two levels of treatments, and ifn indicates the use of interferon as a treatment. The variables et and lt are the exponential and logarithm transformation of the time t. The published data contains other potential covariates that are not listed here. This example concentrates on the effectiveness of the interferon treatment.

Exponential Survival Model

The density function for exponentially distributed survival times is as follows:

$$p(t_i|\lambda_i) = \lambda_i \exp(-\lambda_i t_i)$$

Note that this formulation of the exponential distribution is different from what is used in the SAS probability function PDF. The definition used in PDF for the exponential distributions is as follows:

$$p(t_i|\nu_i) = \frac{1}{\nu_i}\exp(-\frac{t_i}{\nu_i})$$

The relationship between λ and ν is as follows:

$$\lambda_i = \frac{1}{\nu_i}$$

The corresponding survival function, using the λ_i formulation, is as follows:

$$S(t_i|\lambda_i) = \exp(-\lambda_i t_i)$$

If you have a sample $\{t_i\}$ of n independent exponential survival times, each with mean λ_i, then the likelihood function in terms of λ is as follows:

$$\begin{aligned} L(\lambda|t) &= \Pi_{i=1}^{n} p(t_i|\lambda_i)^{\nu_i} S(t_i|\lambda_i)^{1-\nu_i} \\ &= \Pi_{i=1}^{n} (\lambda_i \exp(-\lambda_i t_i))^{\nu_i} (\exp(-\lambda_i t_i))^{1-\nu_i} \\ &= \Pi_{i=1}^{n} \lambda_i^{\nu_i} \exp(-\lambda_i t_i) \end{aligned}$$

If you link the covariates to λ with $\lambda_i = \exp \mathbf{x}_i'\boldsymbol{\beta}$, where $\mathbf{x}_i$ is the vector of covariates corresponding to the ith observation and $\boldsymbol{\beta}$ is a vector of regression coefficients, then the log-likelihood function is as follows:

$$l(\boldsymbol{\beta}|t,\mathbf{x}) = \sum_{i=1}^{n} \nu_i \mathbf{x}_i'\boldsymbol{\beta} - t_i \exp(\mathbf{x}_i'\boldsymbol{\beta})$$

In the absence of prior information about the parameters in this model, you can choose diffuse normal priors for the β:

$$\beta \sim \text{normal}(0, \text{sd} = 10000)$$

There are two ways to program the log-likelihood function in PROC MCMC. You can use the SAS functions LOGPDF and LOGSDF. Alternatively, you can use the simplified log-likelihood function, which is more computationally efficient. You get identical results by using either approaches.

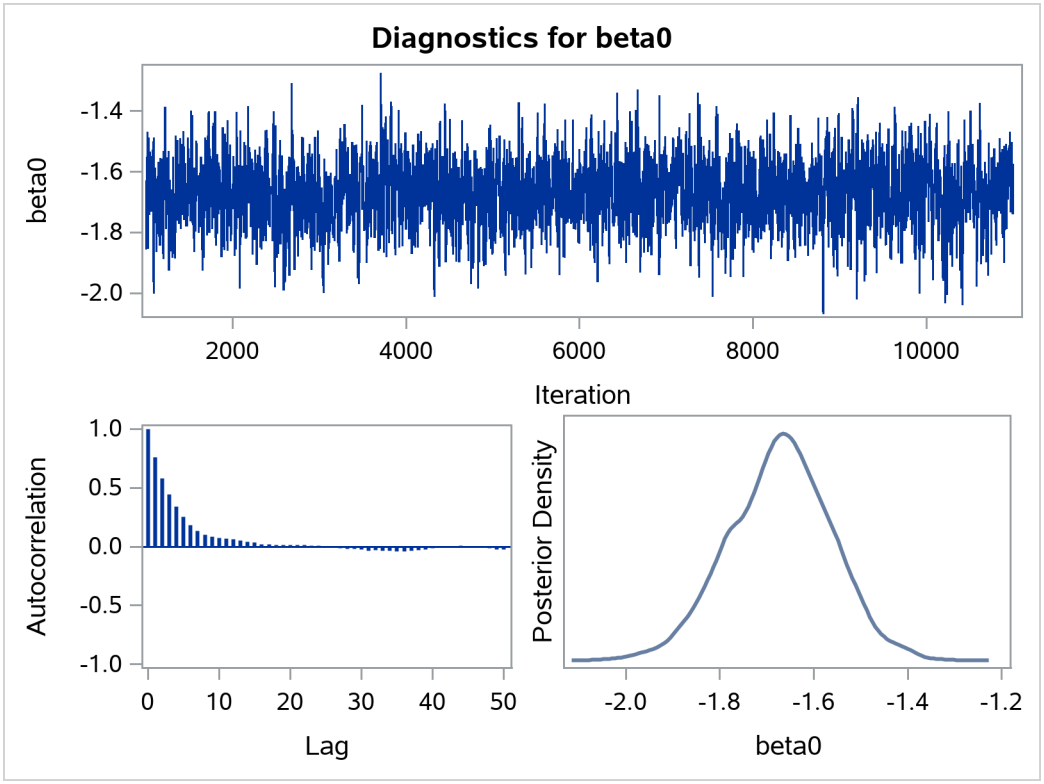
The following PROC MCMC statements fit an exponential model with simplified log-likelihood function:

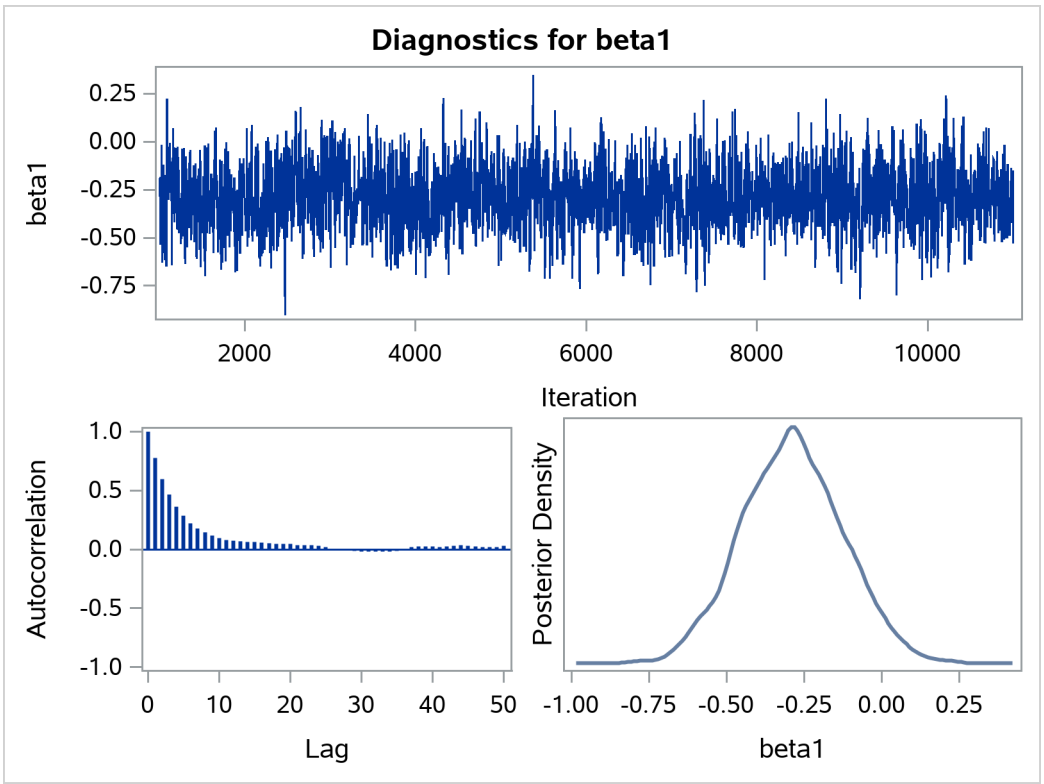
```
title 'Exponential Survival Model';
ods graphics on;
proc mcmc data=e1684 outpost=expsurvout nmc=10000 seed=4861
   diag=(mcse ess);
   ods select PostSumInt TADpanel
              ess mcse;
   parms (beta0 beta1) 0;
   prior beta: ~ normal(0, sd = 10000);
   /*****************************************************/
   /* (1) the logpdf and logsdf functions are not used  */
   /*****************************************************/
   /*     nu = 1/exp(beta0 + beta1*ifn);
          llike = v*logpdf("exponential", t, nu) +
                  (1-v)*logsdf("exponential", t, nu);
   */
   /*****************************************************/
   /* (2) the simplified likelihood formula is used    */
   /*****************************************************/
   l_h = beta0 + beta1*ifn;
   llike = v*(l_h) -  t*exp(l_h);
   model general(llike);
run;
```

The two assignment statements that are commented out calculate the log-likelihood function by using the SAS functions LOGPDF and LOGSDF for the exponential distribution. The next two assignment statements calculate the log likelihood by using the simplified formula. The first approach is slower because of the redundant calculation involved in calling both LOGPDF and LOGSDF.

An examination of the trace plots for β_0 and β_1 (see Output 80.13.1) reveals that the sampling has gone well with no particular concerns about the convergence or mixing of the chains.

Output 80.13.1 Posterior Plots for β_0 and β_1 in the Exponential Survival Analysis





The MCMC results are shown in Output 80.13.2.

Output 80.13.2 Posterior Summary and Interval Statistics

Exponential Survival Model

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta0	10000	-1.6715	0.1091	-1.8930	-1.4673
beta1	10000	-0.2879	0.1615	-0.6104	0.0169

The Monte Carlo standard errors and effective sample sizes are shown in Output 80.13.3. The posterior means for β_0 and β_1 are estimated with high precision, with small standard errors with respect to the standard deviation. This indicates that the mean estimates have stabilized and do not vary greatly in the course of the simulation. The effective sample sizes are roughly the same for both parameters.

Output 80.13.3 MCSE and ESS

Exponential Survival Model

The MCMC Procedure

Monte Carlo Standard Errors

Parameter	MCSE	Standard Deviation	MCSE/SD
beta0	0.00302	0.1091	0.0277
beta1	0.00485	0.1615	0.0301

Effective Sample Sizes

Parameter	ESS	Autocorrelation Time	Efficiency
beta0	1304.1	7.6682	0.1304
beta1	1107.2	9.0319	0.1107

The next part of this example shows fitting a Weibull regression to the data and then comparing the two models with DIC to see which one provides a better fit to the data.

Weibull Survival Model

The density function for Weibull distributed survival times is as follows:

$$p(t_i|\alpha,\lambda_i) = \alpha t_i^{\alpha-1}\exp(\lambda_i - \exp(\lambda_i)t_i^{\alpha})$$

Note that this formulation of the Weibull distribution is different from what is used in the SAS probability function PDF. The definition used in PDF is as follows:

$$p(t_i|\alpha,\gamma_i) = \exp\left(-\left(\frac{t_i}{\gamma_i}\right)^{\alpha}\right)\frac{\alpha}{\gamma_i}\left(\frac{t_i}{\gamma_i}\right)^{\alpha-1}$$

The relationship between λ and γ in these two parameterizations is as follows:

$$\lambda_i = -\alpha\log\gamma_i$$

The corresponding survival function, using the λ_i formulation, is as follows:

$$S(t_i|\alpha,\lambda_i) = \exp(-\exp(\lambda_i)t_i^{\alpha})$$

If you have a sample $\{t_i\}$ of n independent Weibull survival times, with parameters α, and λ_i, then the likelihood function in terms of α and λ is as follows:

$$\begin{aligned} L(\alpha,\lambda|t) &= \Pi_{i=1}^{n} p(t_i|\alpha,\lambda_i)^{\nu_i} S(t_i|\alpha,\lambda_i)^{1-\nu_i} \\ &= \Pi_{i=1}^{n} (\alpha t_i^{\alpha-1}\exp(\lambda_i - \exp(\lambda_i)t_i^{\alpha}))^{\nu_i} (\exp(-\exp(\lambda_i)t_i^{\alpha}))^{1-\nu_i} \\ &= \Pi_{i=1}^{n} (\alpha t_i^{\alpha-1}\exp(\lambda_i))^{\nu_i} (\exp(-\exp(\lambda_i)t_i^{\alpha})) \end{aligned}$$

If you link the covariates to λ with $\lambda_i = \mathbf{x}_i'\boldsymbol{\beta}$, where $\mathbf{x}_i$ is the vector of covariates corresponding to the ith observation and $\boldsymbol{\beta}$ is a vector of regression coefficients, the log-likelihood function becomes this:

$$l(\alpha,\boldsymbol{\beta}|t,\mathbf{x}) = \sum_{i=1}^{n} \nu_i(\log(\alpha) + (\alpha-1)\log(t_i) + \mathbf{x}_i'\boldsymbol{\beta}) - \exp(\mathbf{x}_i'\boldsymbol{\beta})t_i^{\alpha})$$

As with the exponential model, in the absence of prior information about the parameters in this model, you can use diffuse normal priors on β. You might want to choose a diffuse gamma distribution for α. Note that when $\alpha = 1$, the Weibull survival likelihood reduces to the exponential survival likelihood. Equivalently, by looking at the posterior distribution of α, you can conclude whether fitting an exponential survival model would be more appropriate than the Weibull model.

PROC MCMC also enables you to make inference on any functions of the parameters. Quantities of interest in survival analysis include the value of the survival function at specific times for specific treatments and the relationship between the survival curves for different treatments. With PROC MCMC, you can compute a sample from the posterior distribution of the interested survival functions at any number of points. The data in this example range from about 0 to 10 years, and the treatment of interest is the use of interferon.

Like in the previous exponential model example, there are two ways to fit this model: using the SAS functions LOGPDF and LOGSDF, or using the simplified log likelihood functions. The example uses the latter method. The following statements run PROC MCMC and produce Output 80.13.4:

```
title 'Weibull Survival Model';
proc mcmc data=e1684 outpost=weisurvout nmc=10000 seed=1234
          monitor=(_parms_ surv_ifn surv_noifn) stats=(summary intervals);
   ods select PostSummaries;
   ods output PostSummaries=ds PostIntervals=is;
   array surv_ifn[10];
   array surv_noifn[10];
   parms alpha 1 (beta0 beta1) 0;
   prior beta: ~ normal(0, var=10000);
   prior alpha ~ gamma(0.001,is=0.001);

   beginnodata;
      do t1 = 1 to 10;
         surv_ifn[t1] = exp(-exp(beta0+beta1)*t1**alpha);
         surv_noifn[t1] = exp(-exp(beta0)*t1**alpha);
      end;
   endnodata;
```

```
   lambda = beta0 + beta1*ifn;
   /*****************************************************/
   /* (1) the logpdf and logsdf functions are not used  */
   /*****************************************************/
   /*     gamma =  exp(-lambda /alpha);
          llike = v*logpdf('weibull', t, alpha, gamma) +
                  (1-v)*logsdf('weibull', t, alpha, gamma);
   */
   /*****************************************************/
   /* (2) the simplified likelihood formula is used     */
   /*****************************************************/
   llike  = v*(log(alpha) + (alpha-1)*log(t) + lambda) -
            exp(lambda)*(t**alpha);
   model general(llike);
run;
```

The MONITOR= option indicates the parameters and quantities of interest that PROC MCMC tracks. The symbol _PARMS_ specifies all model parameters. The array surv_ifn stores the expected survival probabilities for patients who received interferon over a period of 10 years. Similarly, surv_noifn stores the expected survival probabilities for patients who did not received interferon.

The BEGINNODATA and ENDNODATA statements enclose the calculations for the survival probabilities. The assignment statements proceeding the MODEL statement calculate the log likelihood for the Weibull survival model. The MODEL statement specifies the log likelihood that you programmed.

An examination of the trace plots for α, β_0, and β_1 (not displayed here) reveals that the sampling has gone well, with no particular concerns about the convergence or mixing of the chains.

Output 80.13.4 displays the posterior summary statistics.

Output 80.13.4 Posterior Summary Statistics

Weibull Survival Model

The MCMC Procedure

Posterior Summaries

Parameter	N	Mean	Standard Deviation	Percentiles 25	50	75
alpha	10000	0.7891	0.0539	0.7514	0.7880	0.8260
beta0	10000	-1.3581	0.1369	-1.4519	-1.3597	-1.2624
beta1	10000	-0.2512	0.1541	-0.3541	-0.2606	-0.1521
surv_ifn1	10000	0.8175	0.0227	0.8027	0.8187	0.8331
surv_ifn2	10000	0.7066	0.0291	0.6874	0.7072	0.7265
surv_ifn3	10000	0.6203	0.0331	0.5983	0.6205	0.6436
surv_ifn4	10000	0.5495	0.0360	0.5253	0.5497	0.5747
surv_ifn5	10000	0.4899	0.0381	0.4635	0.4895	0.5170
surv_ifn6	10000	0.4390	0.0396	0.4118	0.4382	0.4666
surv_ifn7	10000	0.3949	0.0406	0.3669	0.3934	0.4223
surv_ifn8	10000	0.3564	0.0413	0.3281	0.3551	0.3840
surv_ifn9	10000	0.3225	0.0416	0.2940	0.3212	0.3505
surv_ifn10	10000	0.2926	0.0416	0.2638	0.2911	0.3208
surv_noifn1	10000	0.7719	0.0274	0.7535	0.7736	0.7913
surv_noifn2	10000	0.6401	0.0339	0.6171	0.6415	0.6635
surv_noifn3	10000	0.5415	0.0374	0.5161	0.5428	0.5662
surv_noifn4	10000	0.4635	0.0395	0.4365	0.4636	0.4890
surv_noifn5	10000	0.4001	0.0406	0.3725	0.3995	0.4261
surv_noifn6	10000	0.3475	0.0411	0.3195	0.3459	0.3745
surv_noifn7	10000	0.3034	0.0411	0.2758	0.3012	0.3299
surv_noifn8	10000	0.2661	0.0406	0.2384	0.2630	0.2921
surv_noifn9	10000	0.2342	0.0399	0.2069	0.2311	0.2592
surv_noifn10	10000	0.2069	0.0389	0.1803	0.2035	0.2312

An examination of the α parameter reveals that the exponential model might not be inappropriate here. The estimated posterior mean of α is 0.7856 with a posterior standard deviation of 0.0533. As noted previously, if $\alpha = 1$, then the Weibull survival distribution is the exponential survival distribution. With these data, you can see that the evidence is in favor of $\alpha < 1$. The value 1 is almost 4 posterior standard deviations away from the posterior mean. The following statements compute the posterior probability of the hypothesis that $\alpha < 1$.:

```
proc format;
   value alphafmt low-<1 = 'alpha < 1' 1-high = 'alpha >= 1';
run;

proc freq data=weisurvout;
   tables alpha /nocum;
   format alpha alphafmt.;
run;
```

The PROC FREQ results are shown in Output 80.13.5.

Output 80.13.5 Frequency Analysis of α

Weibull Survival Model

The FREQ Procedure

alpha	Frequency	Percent
alpha < 1	9998	99.98
alpha >= 1	2	0.02

The output from PROC FREQ shows that 100% of the 10000 simulated values for α are less than 1. This is a very strong indication that the exponential model is too restrictive to model these data well.

You can examine the estimated survival probabilities over time individually, either through the posterior summary statistics or by looking at the kernel density plots. Alternatively, you might find it more informative to examine these quantities in relation with each other. For example, you can use a side-by-side box plot to display these posterior distributions by using PROC SGPLOT. For more information, see the section "Statistical Graphics Using ODS" on page 665 in Chapter 24, "Statistical Graphics Using ODS." First you need to take the posterior output data set Weisurvout and stack variables that you want to plot. For example, to plot all the survival times for patients who received interferon, you want to stack surv_inf1–surv_inf10. The macro %Stackdata takes an input data set dataset, stacks the wanted variables vars, and outputs them into the output data set.

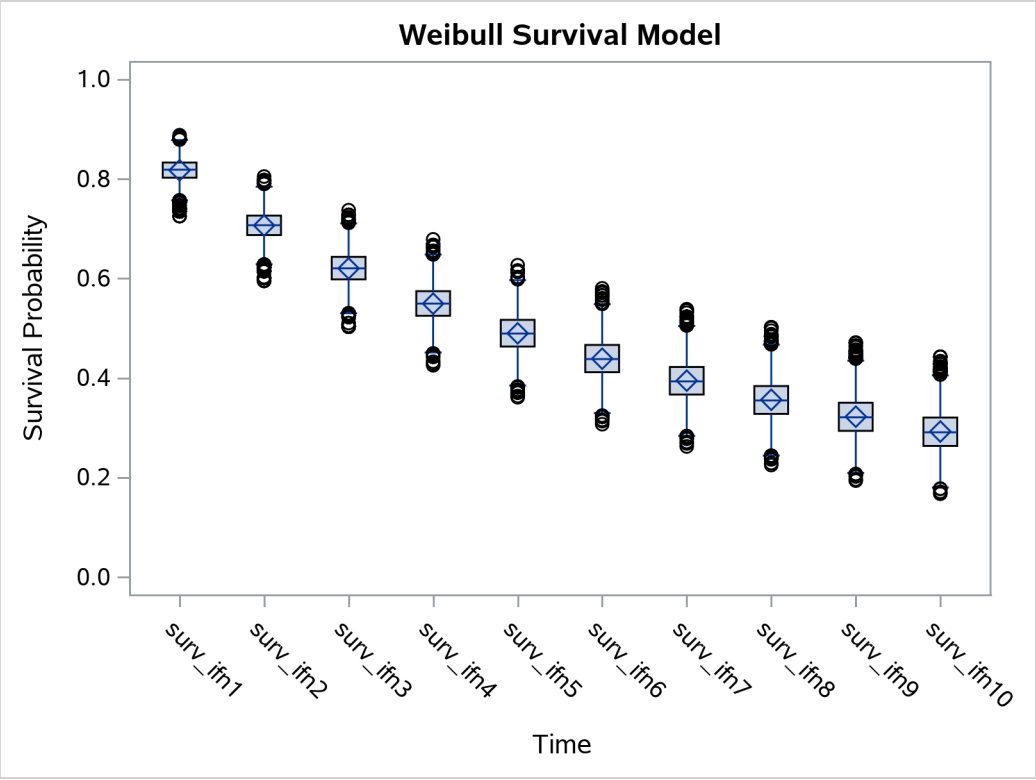
The following statements define the macro stackdata:

```
/* define macro stackdata */
%macro StackData(dataset,output,vars);
   data &output;
      length var $ 32;
      if 0 then set &dataset nobs=nnn;
      array lll[*] &vars;
      do jjj=1 to dim(lll);
         do iii=1 to nnn;
            set &dataset point=iii;
            value = lll[jjj];
            call vname(lll[jjj],var);
            output;
         end;
      end;
      stop;
      keep var value;
   run;
%mend;

/* stack the surv_ifn variables and saved them to survifn. */
%StackData(weisurvout, survifn, surv_ifn1-surv_ifn10);
```

Once you stack the data, use PROC SGPLOT to create the side-by-side box plots. The following statements generate Output 80.13.6:

```
proc sgplot data=survifn;
   yaxis label='Survival Probability' values=(0 to 1 by 0.2);
   xaxis label='Time' discreteorder=data;
   vbox value / category=var;
run;
```

Output 80.13.6 Side-by-Side Box Plots of Estimated Survival Probabilities



There is a clear decreasing trend over time of the survival probabilities for patients who receive the treatment. You might ask how does this group compare to those who did not receive the treatment? In this case, you want to overlay the two predicted curves for the two groups of patients and add the corresponding credible interval. See Output 80.13.7. To generate the graph, you first take the posterior mean estimates from the ODS output table ds and the lower and upper HPD interval estimates is, store them in the data set Surv, and draw the figure by using PROC SGPLOT.

The following statements generate data set Surv:

```
data surv;
   set ds;
   if _n_ >= 4 then do;
      set is point=_n_;
      group = 'with interferon   ';
      time = _n_ - 3;
      if time > 10 then do;
         time = time - 10;
         group = 'without interferon';
      end;
```

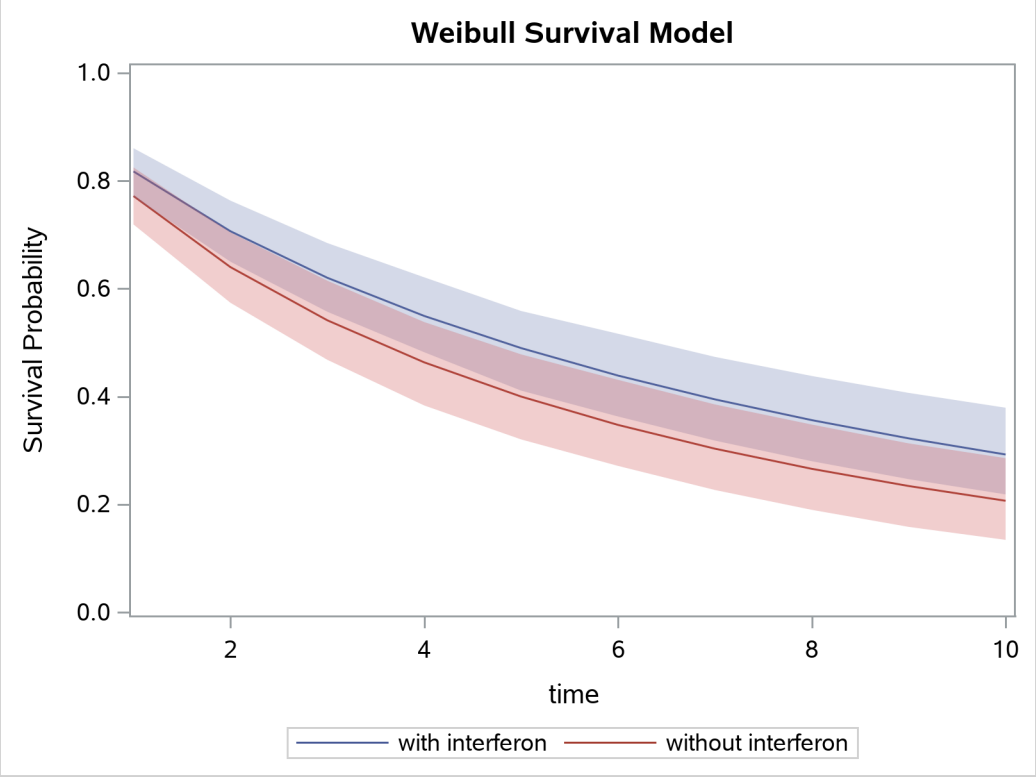
```
      output;
   end;
   keep time group mean hpdlower hpdupper;
run;
```

The following SGPLOT statements generate Output 80.13.7:

```
proc sgplot data=surv;
   yaxis label="Survival Probability" values=(0 to 1 by 0.2);
   series x=time y=mean / group = group name='i';
   band x=time lower=hpdlower upper=hpdupper / group = group transparency=0.7;
   keylegend 'i';
run;
ods graphics off;
```

In Output 80.13.7, the solid line is the survival curve for patients who received interferon; the shaded region centers at the solid line is the 95% HPD intervals; the medium-dashed line is the survival curve for patients who did not receive interferon; and the shaded region around the dashed line is the corresponding 95% HPD intervals.

Output 80.13.7 Predicted Survival Probability Curves with 95% HPD Intervals



The plot suggests that there is an effect of using interferon because patients who received interferon have sustained better survival probabilities than those who did not. However, the effect might not be very significant, as the 95% credible intervals of the two groups do overlap. For more on these interferon studies, see Ibrahim, Chen, and Lipsitz (2001).

Weibull or Exponential?

Although the evidence from the Weibull model fit shows that the posterior distribution of α has a significant amount of density mass less than 1, suggesting that the Weibull model is a better fit to the data than the exponential model, you might still be interested in comparing the two models more formally. You can use the Bayesian model selection criterion (see the section "Deviance Information Criterion (DIC)" on page 177 in Chapter 8, "Introduction to Bayesian Analysis Procedures") to determine which model fits the data better.

The PROC MCMC DIC option requests the calculation of DIC, and the procedure displays the ODS output table DIC. The table includes the posterior mean of the deviation, $\overline{D(\boldsymbol{\theta})}$, deviation at the estimate, $D(\overline{\boldsymbol{\theta}})$, effective number of parameters, p_D, and DIC. It is important to remember that the standardizing term, $p(\mathbf{y})$, which is a function of the data alone, is not taken into account in calculating the DIC. This term is irrelevant only if you compare two models that have the *same* likelihood function. If you do not have identical likelihood functions, using DIC for model selection purposes without taking this standardizing term into account can produce incorrect results. In addition, you want to be careful in interpreting the DIC whenever you use the GENERAL function to construct the log-likelihood, as the case in this example. Using the GENERAL function, you can obtain identical posterior samples with two log-likelihood functions that differ only by a constant. This difference translates to a difference in the DIC calculation, which could be very misleading.

If $\alpha = 1$, the Weibull likelihood is identical to the exponential likelihood. It is safe in this case to directly compare DICs from these two models. However, if you do not want to work out the mathematical detail or you are uncertain of the equivalence, a better way of comparing the DICs is to run the Weibull model twice: once with α being a parameter and once with $\alpha = 1$. This ensures that the likelihood functions are the same, and the DIC comparison is meaningful.

The following statements fit a Weibull model:

```
title 'Model Comparison between Weibull and Exponential';
proc mcmc data=e1684 outpost=weisurvout nmc=10000 seed=4861 dic;
   ods select dic;
   parms alpha 1 (beta0 beta1) 0;
   prior beta: ~ normal(0, var=10000);
   prior alpha ~ gamma(0.001,is=0.001);

   lambda = beta0 + beta1*ifn;
   llike  = v*(log(alpha) + (alpha-1)*log(t) + lambda) -
            exp(lambda)*(t**alpha);
   model general(llike);
run;
```

The DIC option requests the calculation of DIC, and the table is displayed in Output 80.13.8.

Output 80.13.8 DIC Table from the Weibull Model

Model Comparison between Weibull and Exponential

The MCMC Procedure

Deviance Information Criterion	
Dbar (posterior mean of deviance)	858.623
Dmean (deviance evaluated at posterior mean)	855.633
pD (effective number of parameters)	2.990
DIC (smaller is better)	861.614

The GENERAL or DGENERAL function is used in this program. To make meaningful comparisons, you must ensure that all GENERAL or DGENERAL functions include appropriate normalizing constants. Otherwise, DIC comparisons can be misleading.

The note in Output 80.13.8 reminds you of the importance of ensuring identical likelihood functions when you use the GENERAL function. The DIC value is 861.6.

Based on the same set of code, the following statements fit an exponential model by setting $\alpha = 1$:

```
proc mcmc data=e1684 outpost=expsurvout nmc=10000 seed=4861 dic;
   ods select dic;
   parms beta0 beta1 0;
   prior beta: ~ normal(0, var=10000);
   begincnst;
      alpha = 1;
   endcnst;

   lambda = beta0 + beta1*ifn;
   llike  = v*(log(alpha) + (alpha-1)*log(t) + lambda) -
            exp(lambda)*(t**alpha);
   model general(llike);
run;
```

Output 80.13.9 displays the DIC table.

Output 80.13.9 DIC Table from the Exponential Model

Model Comparison between Weibull and Exponential

The MCMC Procedure

Deviance Information Criterion	
Dbar (posterior mean of deviance)	870.133
Dmean (deviance evaluated at posterior mean)	868.190
pD (effective number of parameters)	1.943
DIC (smaller is better)	872.075

The GENERAL or DGENERAL function is used in this program. To make meaningful comparisons, you must ensure that all GENERAL or DGENERAL functions include appropriate normalizing constants. Otherwise, DIC comparisons can be misleading.

The DIC value of 872.075 is greater than 861. A smaller DIC indicates a better fit to the data; hence, you can conclude that the Weibull model is more appropriate for this data set. You can see the equivalencing of the exponential model you fitted in "Exponential Survival Model" on page 6425 by running the following comparison.

The following statements are taken from the section "Exponential Survival Model" on page 6425, and they fit the same exponential model:

```
proc mcmc data=e1684 outpost=expsurvout1 nmc=10000 seed=4861 dic;
   ods select none;
   parms (beta0 beta1) 0;
   prior beta: ~ normal(0, sd = 10000);
   l_h = beta0 + beta1*ifn;
   llike = v*(l_h) -  t*exp(l_h);
   model general(llike);
run;

proc compare data=expsurvout compare=expsurvout1;
   var beta0 beta1;
run;
```

The posterior samples of beta0 and beta1 in the data set Expsurvout1 are identical to those in the data set Expsurvout. The comparison results are not shown here.

Example 80.14: Time Independent Cox Model

(View the complete code for this example (mcmcex14.sas) in the example repository.)

This example has two purposes. One is to illustrate how to use PROC MCMC to fit a Cox proportional hazard model. Specifically, the time independent model. See "Example 80.15: Time Dependent Cox Model" on page 6444 for an example on fitting time dependent Cox model. Note that it is much easier to fit a Bayesian Cox model by specifying the BAYES statement in PROC PHREG (see Chapter 92, "The PHREG Procedure"). If you are interested only in fitting a Cox regression survival model, you should use PROC PHREG.

The second objective of this example is to demonstrate how to model data that are not independent. That is the case where the likelihood for observation i depends on other observations in the data set. In other words, if you work with a likelihood function that cannot be broken down simply as $L(\mathbf{y}) = \prod_i^n L(y_i)$, you can use this example for illustrative purposes. By default, PROC MCMC assumes that the programming statements and model specification is intended for a single row of observations in the data set. The Cox model is chosen because the complexity in the data structure requires more elaborate coding.

The Cox proportional hazard model is widely used in the analysis of survival time, failure time, or other duration data to explain the effect of exogenous explanatory variables. The data set used in this example is taken from Krall, Uthoff, and Harley (1975), who analyzed data from a study on myeloma in which researchers treated 65 patients with alkylating agents. Of those patients, 48 died during the study and 17 survived. The following statements generate the data set that is used in this example:

```
data Myeloma;
   input Time Vstatus LogBUN HGB Platelet Age LogWBC Frac
         LogPBM Protein SCalc;
   label Time='survival time'
```

```
        VStatus='0=alive 1=dead';
   datalines;
 1.25  1  2.2175   9.4  1  67  3.6628  1  1.9542  12  10
 1.25  1  1.9395  12.0  1  38  3.9868  1  1.9542  20  18
 2.00  1  1.5185   9.8  1  81  3.8751  1  2.0000   2  15
 2.00  1  1.7482  11.3  0  75  3.8062  1  1.2553   0  12

   ... more lines ...

77.00  0  1.0792  14.0  1  60  3.6812  0  0.9542   0  12
;

proc sort data = Myeloma;
   by descending time;
run;

data _null_;
   set Myeloma nobs=_n;
   call symputx('N', _n);
   stop;
run;
```

The variable Time represents the survival time in months from diagnosis. The variable VStatus consists of two values, 0 and 1, indicating whether the patient was alive or dead, respectively, at the end of the study. If the value of VStatus is 0, the corresponding value of Time is censored. The variables thought to be related to survival are LogBUN ($\log$(BUN) at diagnosis), HGB (hemoglobin at diagnosis), Platelet (platelets at diagnosis: 0=abnormal, 1=normal), Age (age at diagnosis in years), LogWBC (log(WBC) at diagnosis), Frac (fractures at diagnosis: 0=none, 1=present), LogPBM (log percentage of plasma cells in bone marrow), Protein (proteinuria at diagnosis), and SCalc (serum calcium at diagnosis). Interest lies in identifying important prognostic factors from these explanatory variables. In addition, there are 65 (&n) observations in the data set Myeloma. The likelihood used in these examples is the Breslow likelihood:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \left[\prod_{j=1}^{d_i} \frac{\exp(\boldsymbol{\beta}' \mathbf{Z}_j(t_i))}{\sum_{l \in \mathcal{R}_i} \exp(\boldsymbol{\beta}' \mathbf{Z}_l(t_i))} \right]^{v_i}$$

where

- $\boldsymbol{\beta}$ is the vector parameters
- n is the total number of observations in the data set
- t_i is the ith time, which can be either event time or censored time
- $\mathbf{Z}_l(t)$ is the vector explanatory variables for the lth individual at time t
- d_i is the multiplicity of failures at t_i. If there are no ties in time, d_i is 1 for all i.
- $\mathcal{R}_i$ is the risk set for the ith time t_i, which includes all observations that have survival time greater than or equal to t_i
- v_i indicates whether the patient is censored. The value 0 corresponds to censoring. Note that the censored time t_i enters the likelihood function only through the formation of the risk set $\mathcal{R}_i$.

Priors on the coefficients are independent normal priors with very large variance (1e6). Throughout this example, the symbol bZ represents the regression term $\boldsymbol{\beta}'\mathbf{Z}_j(t_i)$ in the likelihood, and the symbol S represents the term $\sum_{l\in\mathcal{R}_i}\exp(\boldsymbol{\beta}'\mathbf{Z}_l(t_i))$.

The regression model considered in this example uses the following formula:

$$\begin{aligned}\boldsymbol{\beta}'\mathbf{Z}_j &= \beta_1\text{logbun} + \beta_2\text{hgb} + \beta_3\text{platelet} + \beta_4\text{age} + \\ &\quad \beta_5\text{logwbc} + \beta_6\text{frac} + \beta_7\text{logpbm} + \beta_8\text{protein} + \beta_9\text{scalc}\end{aligned}$$

The hard part of coding this in PROC MCMC is the construction of the risk set $\mathcal{R}_i$. $\mathcal{R}_i$ contains all observations that have survival time greater than or equal to t_i. First suppose that there are no ties in time. Sorting the data set by the variable time into descending order gives you $\mathcal{R}_i$ that is in the right order. Observation i's risk set consists of all data points j such that $j <= i$ in the data set. You can cumulatively increment S in the SAS statements.

With potential ties in time, at observation i, you need to know whether any subsequent observations, $i+1$ and so on, have the same survival time as t_i. Suppose that the ith, the $i+1$, and the $i+2$ observations all have the same survival time; all three of them need to be included in the risk set calculation. This means that to calculate the likelihood for some observations, you need to access both the previous and subsequent observations in the data set. There are two ways to do this. One is to use the LAG function; the other is to use the option JOINTMODEL.

The LAG function returns values from a queue (see *SAS Functions and CALL Routines: Reference*). So for the ith observation, you can use LAG1 to access variables from the previous row in the data set. You want to compare the lag1 value of time with the current time value. Depending on whether the two time values are equal, you can add correction terms in the calculation for the risk set S.

The following statements sort the data set by time into descending order, with the largest survival time on top:

```
title 'Cox Model with Time Independent Covariates';
proc freq data=myeloma;
   ods select none;
   tables time / out=freqs;
run;

proc sort data = freqs;
   by descending time;
run;

data myelomaM;
   set myeloma;
   ind = _N_;
run;
ods select all;
```

The following statements run PROC MCMC and produce Output 80.14.1:

```
proc mcmc data=myelomaM outpost=outi nmc=50000 ntu=3000 seed=1;
   ods select PostSumInt;
   array beta[9];
   parms beta: 0;
   prior beta: ~ normal(0, var=1e6);
```

```
   bZ = beta1 * LogBUN + beta2 * HGB + beta3 * Platelet
      + beta4 * Age + beta5 * LogWBC + beta6 * Frac +
      beta7 * LogPBM + beta8 * Protein + beta9 * SCalc;

   if ind = 1 then do;        /* first observation         */
      S = exp(bZ);
      l = vstatus * bZ;
      v = vstatus;
   end;
   else if (1 < ind < &N) then do;
      if (lag1(time) ne time) then do;
         l = vstatus * bZ;
         l = l - v * log(S);  /* correct the loglike value   */
         v = vstatus;         /* reset v count value         */
         S = S + exp(bZ);
      end;
      else do;                /* still a tie                 */
         l = vstatus * bZ;
         S = S + exp(bZ);
         v = v + vstatus;     /* add # of nonsensored values */
      end;
   end;
   else do;                   /* last observation            */
      if (lag1(time) ne time) then do;
         l = - v * log(S);    /* correct the loglike value   */
         S = S + exp(bZ);
         l = l + vstatus * (bZ - log(S));
      end;
      else do;
         S = S + exp(bZ);
         l = vstatus * bZ - (v + vstatus) * log(S);
      end;
   end;
   model general(l);
run;
```

The symbol bZ is the regression term, which is independent of the time variable. The symbol ind indexes observation numbers in the data set. The symbol S keeps track of the risk set term for every observation. The symbol l calculates the log likelihood for each observation. Note that the value of l for observation ind is not necessarily the correct log likelihood value for that observation, especially in cases where the observation ind is in the tied times. Correction terms are added to subsequent values of l when the time variable becomes different in order to make up the difference. The total sum of l calculated over the entire data set is correct. The symbol v keeps track of the sum of vstatus, as censored data do not enter the likelihood and need to be taken out.

You use the function LAG1 to detect if two adjacent time values are different. If they are, you know that the current observation is in a different risk set than the last one. You then need to add a correction term to the log likelihood value of l. The IF-ELSE statements break the observations into three parts: the first observation, the last observation and everything in the middle.

Output 80.14.1 Summary Statistics on Cox Model with Time Independent Explanatory Variables and Ties in the Survival Time, Using PROC MCMC

Cox Model with Time Independent Covariates

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta1	50000	1.7600	0.6441	0.5117	3.0465
beta2	50000	-0.1308	0.0720	-0.2746	0.00524
beta3	50000	-0.2017	0.5148	-1.2394	0.7984
beta4	50000	-0.0126	0.0194	-0.0512	0.0245
beta5	50000	0.3373	0.7256	-1.1124	1.7291
beta6	50000	0.3992	0.4337	-0.4385	1.2575
beta7	50000	0.3749	0.4861	-0.5423	1.3689
beta8	50000	0.0106	0.0271	-0.0451	0.0616
beta9	50000	0.1272	0.1064	-0.0763	0.3406

An alternative to using the LAG function is to use the PROC option JOINTMODEL. With this option, the log-likelihood function you specify applies not to a single observation but to the entire data set. See "Modeling Joint Likelihood" on page 6304 for details on how to properly use this option. The basic idea is that you store all necessary data set variables in arrays and use only the arrays to construct the log likelihood of the entire data set. This approach works here because for every observation *i*, you can use index to access different values of arrays to construct the risk set S. To use the JOINTMODEL option, you need to do some additional data manipulation. You want to create a stop variable for each observation, which indicates the observation number that should be included in S for that observation. For example, if observations 4, 5, 6 all have the same survival time, the stop value for all of them is 6.

The following statements generate a new data set MyelomaM that contains the stop variable:

```
data myelomaM;
   merge myelomaM freqs(drop=percent);
   by descending time;
   retain stop;
   if first.time then do;
      stop = _n_ + count - 1;
   end;
run;
```

The following SAS program fits the same Cox model by using the JOINTMODEL option:

```
data a;
run;

proc mcmc data=a outpost=outa nmc=50000 ntu=3000 seed=1 jointmodel;
   ods select none;
   array beta[9];
   array data[1] / nosymbols;
   array timeA[1] / nosymbols;
   array vstatusA[1] / nosymbols;
   array stopA[1] / nosymbols;
```

```
   array bZ[&n];
   array S[&n];

   begincnst;
   rc = read_array("myelomam", data, "logbun", "hgb", "platelet",
                   "age", "logwbc", "frac", "logpbm", "protein", "scalc");
   rc = read_array("myelomam", timeA, "time");
   rc = read_array("myelomam", vstatusA, "vstatus");
   rc = read_array("myelomam", stopA, "stop");
   endcnst;

   parms (beta:) 0;
   prior beta: ~ normal(0, var=1e6);

   jl = 0;
   /* calculate each bZ and cumulatively adding S as if there are no ties.*/
   call mult(data, beta, bZ);
   S[1] = exp(bZ[1]);
   do i = 2 to &n;
      S[i] = S[i-1] + exp(bZ[i]);
   end;

   do i = 1 to &n;
      /* correct the S[i] term, when needed. */
      if(stopA[i] > i) then do;
         do j = (i+1) to stopA[i];
            S[i] = S[i] + exp(bZ[j]);
         end;
      end;
      jl = jl + vstatusA[i] * (bZ[i] - log(S[i]));
   end;
   model general(jl);
run;
ods select all;
```

No output tables were produced because this PROC MCMC run produces identical posterior samples as does the previous example.

Because the JOINTMODEL option is specified here, you do not need to specify myelomaM as the input data set. An empty data set a is used to speed up the procedure run.

Multiple ARRAY statements allocate array symbols that are used to store the parameters (beta), the response and the covariates (data, timeA, vstatusA, and stopA), and the work space (bZ and S). The data, timeA, vstatusA, and stopA arrays are declared with the /NOSYMBOLS option. This option enables PROC MCMC to dynamically resize these arrays to match the dimensions of the input data set. See the section "READ_ARRAY Function" on page 6235. The bZ and S arrays store the regression term and the risk set term for every observation.

The BEGINCNST and ENDCNST statements enclose programming statements that read the data set variables into these arrays. The rest of the programming statements construct the log likelihood for the entire data set.

The CALL MULT function calculates the regression term in the model and stores the result in the array bZ. In the first DO loop, you sum the risk set term S as if there are no ties in time. This underevaluates some of the S elements. For observations that have a tied time, you make the necessary correction to the corresponding S values. The correction takes place in the second DO loop. Any observation that has a tied time also has

a stopA[i] that is different from i. You add the right terms to S and sum up the joint log likelihood jl. The MODEL statement specifies that the log likelihood takes on the value of jl.

To see that you get identical results from these two approaches, use PROC COMPARE to compare the posterior samples from two runs:

```
proc compare data=outi compare=outa;
   ods select comparesummary;
   var beta1-beta9;
run;
```

The output is not shown here.

Generally, the JOINTMODEL option can be slightly faster than using the default setup. The savings come from avoiding the overhead cost of accessing the data set repeatedly at every iteration. However, the speed gain is not guaranteed because it largely depends on the efficiency of your programs.

PROC PHREG fits the same model, and you get very similar results to PROC MCMC. The following statements fit the model using PROC PHREG and produce Output 80.14.2:

```
proc phreg data=Myeloma;
   ods select PostSumInt;
   model Time*VStatus(0)=LogBUN HGB Platelet Age LogWBC
                         Frac LogPBM Protein Scalc;
   bayes seed=1 nmc=10000 outpost=phout;
run;
```

Output 80.14.2 Summary Statistics for Cox Model with Time Independent Explanatory Variables and Ties in the Survival Time, Using PROC PHREG

Cox Model with Time Independent Covariates

The PHREG Procedure

Bayesian Analysis

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
LogBUN	10000	1.7610	0.6593	0.4107	2.9958
HGB	10000	-0.1279	0.0727	-0.2801	0.00599
Platelet	10000	-0.2179	0.5169	-1.1871	0.8341
Age	10000	-0.0130	0.0199	-0.0519	0.0251
LogWBC	10000	0.3150	0.7451	-1.1783	1.7483
Frac	10000	0.3766	0.4152	-0.4273	1.2021
LogPBM	10000	0.3792	0.4909	-0.5939	1.3241
Protein	10000	0.0102	0.0267	-0.0405	0.0637
SCalc	10000	0.1248	0.1062	-0.0846	0.3322

Example 80.15: Time Dependent Cox Model

(View the complete code for this example (mcmcex15.sas) in the example repository.)

This example uses the same Myeloma data set as in "Example 80.14: Time Independent Cox Model" on page 6437, and illustrates the fitting of a time dependent Cox model. The following statements generate the data set once again:

```
data Myeloma;
   input Time Vstatus LogBUN HGB Platelet Age LogWBC Frac
         LogPBM Protein SCalc;
   label Time='survival time'
         VStatus='0=alive 1=dead';
   datalines;
 1.25  1  2.2175   9.4  1  67  3.6628  1  1.9542  12  10
 1.25  1  1.9395  12.0  1  38  3.9868  1  1.9542  20  18
 2.00  1  1.5185   9.8  1  81  3.8751  1  2.0000   2  15
 2.00  1  1.7482  11.3  0  75  3.8062  1  1.2553   0  12

   ... more lines ...

77.00  0  1.0792  14.0  1  60  3.6812  0  0.9542   0  12
;
```

To model $\mathbf{Z}_i(t_i)$ as a function of the survival time, you can relate time t_i to covariates by using this formula:

$$\boldsymbol{\beta}'\mathbf{Z}_j(t_i) = (\beta_1 + \beta_2 t_i)\text{logbun} + (\beta_3 + \beta_4 t_i)\text{hgb} + (\beta_5 + \beta_6 t_i)\text{platelet}$$

For illustrational purposes, only three explanatory variables, LOGBUN, HBG, and PLATELET, are used in this example.

Since $\mathbf{Z}_j(t_i)$ depends on t_i, every term in the summation of $\sum_{l \in \mathcal{R}_i} \exp(\boldsymbol{\beta}'\mathbf{Z}_l(t_i))$ is a product of the current time t_i and all observations that are in the risk set. You can use the JOINTMODEL option, as in the last example, or you can modify the input data set such that every row contains not only the current observation but also all observations that are in the corresponding risk set. When you construct the log likelihood for each observation, you have all the relevant data at your disposal.

The following statements illustrate how you can create a new data set with different risk sets at different rows:

```
title 'Cox Model with Time Dependent Covariates';
proc sort data = Myeloma;
   by descending time;
run;

data _null_;
   set Myeloma nobs=_n;
   call symputx('N', _n);
   stop;
run;

ods select none;
proc freq data=myeloma;
   tables time / out=freqs;
```

```
run;
ods select all;

proc sort data = freqs;
   by descending time;
run;

data myelomaM;
   set myeloma;
   ind = _N_;
run;

data myelomaM;
   merge myelomaM freqs(drop=percent); by descending time;
   retain stop;
   if first.time then do;
      stop = _n_ + count - 1;
   end;
run;

%macro array(list);
   %global mcmcarray;
   %let mcmcarray = ;
   %do i = 1 %to 32000;
      %let v = %scan(&list, &i, %str( ));
      %if %nrbquote(&v) ne %then %do;
         array _&v[&n];
         %let mcmcarray = &mcmcarray array _&v[&n] _&v.1 - _&v.&n%str(;);
         do i = 1 to stop;
            set myelomaM(keep=&v) point=i;
            _&v[i] = &v;
         end;
      %end;
      %else %let i = 32001;
   %end;
%mend;

data z;
   set myelomaM;
   %array(logbun hgb platelet);
   drop vstatus logbun hgb platelet count stop;
run;

data myelomaM;
   merge myelomaM z; by descending time;
run;
```

The data set MyelomaM contains 65 observations and 209 variables. For each observation, you see added variables stop, _logbun1 through _logbun65, _hgb1 through _hgb65, and _platelet1 through _platelet65. The variable stop indicates the number of observations that are in the risk set of the current observation. The rest are transposed values of model covariates of the entire data set. The data set contains a number of missing values. This is due to the fact that only the relevant observations are kept, such as _logbun*1* to _logbun*stop*. The rest of the cells are filled in with missing values. For example, the first observation has a unique survival time of 92 and stop is 1, making it a risk set of itself. You see nonmissing values only in

_logbun1, _hgb1, and _platelet1.

The following statements fit the Cox model by using PROC MCMC:

```
proc mcmc data=myelomaM outpost=outi nmc=50000 ntu=3000 seed=17
          missing=ac;
   ods select PostSumInt;
   array beta[6];
   &mcmcarray
   parms (beta:) 0;
   prior beta: ~ normal(0, prec=1e-6);

   b = (beta1 + beta2 * time) * logbun +
       (beta3 + beta4 * time) * hgb +
       (beta5 + beta6 * time) * platelet;
   S = 0;
   do i = 1 to stop;
      S = S + exp( (beta1 + beta2 * time) * _logbun[i] +
                   (beta3 + beta4 * time) * _hgb[i] +
                   (beta5 + beta6 * time) * _platelet[i]);
   end;
   loglike = vstatus * (b - log(S));

   model general(loglike);
run;
```

Note that the option MISSING= is set to AC. This is due to missing cells in the input data set. You must use this option so that PROC MCMC retains observations that contain missing values.

The macro variable &mcmcarray is defined in the earlier part in this example. You can use a **%put** statement to print its value:

```
%put &mcmcarray;
```

This statement prints the following:

```
array _logbun[65] _logbun1 - _logbun65; array _hgb[65] _hgb1 - _hgb65; array
_platelet[65] _platelet1 - _platelet65;
```

The macro uses the ARRAY statement to allocate three arrays, each of which links their corresponding data set variables. This makes it easier to reference these data set variables in the program. The PARMS statement puts all the parameters in the same block. The PRIOR statement gives them normal priors with large variance. The symbol b is the regression term, and S is cumulatively added from 1 to stop for each observation in the DO loop. The symbol loglike completes the construction of log likelihood for each observation and the MODEL statement completes the model specification.

Posterior summary and interval statistics are shown in Output 80.15.1.

Output 80.15.1 Summary Statistics on Cox Model with Time Dependent Explanatory Variables and Ties in the Survival Time, Using PROC MCMC

Cox Model with Time Dependent Covariates

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta1	50000	3.2397	0.8226	1.6664	4.8752
beta2	50000	-0.1411	0.0471	-0.2294	-0.0458
beta3	50000	-0.0369	0.1017	-0.2272	0.1685
beta4	50000	-0.00409	0.00360	-0.0112	0.00264
beta5	50000	0.3548	0.7359	-1.0394	1.8100
beta6	50000	-0.0417	0.0359	-0.1122	0.0269

You can also use the option JOINTMODEL to get the same inference and avoid transposing the data for every observation:

```
proc mcmc data=myelomaM outpost=outa nmc=50000 ntu=3000 seed=17 jointmodel;
   ods select none;
   array beta[6];      array timeA[&n];      array vstatusA[&n];
   array logbunA[&n];  array hgbA[&n];       array plateletA[&n];
   array stopA[&n];    array bZ[&n];         array S[&n];

   begincnst;
      timeA[ind]=time;          vstatusA[ind]=vstatus;
      logbunA[ind]=logbun;      hgbA[ind]=hgb;
      plateletA[ind]=platelet;  stopA[ind]=stop;
   endcnst;

   parms (beta:) 0;
   prior beta: ~ normal(0, prec=1e-6);

   jl = 0;
   do i = 1 to &n;
      v1 = beta1 + beta2 * timeA[i];
      v2 = beta3 + beta4 * timeA[i];
      v3 = beta5 + beta6 * timeA[i];
      bZ[i] = v1 * logbunA[i] + v2 * hgbA[i] + v3 * plateletA[i];

      /* sum over risk set without considering ties in time. */
      S[i] = exp(bZ[i]);
      if (i > 1) then do;
         do j = 1 to (i-1);
            b1 = v1 * logbunA[j] + v2 * hgbA[j] + v3 * plateletA[j];
            S[i] = S[i] + exp(b1);
         end;
      end;
   end;

   /* make correction to the risk set due to ties in time. */
```

```
   do i = 1 to &n;
      if(stopA[i] > i) then do;
         v1 = beta1 + beta2 * timeA[i];
         v2 = beta3 + beta4 * timeA[i];
         v3 = beta5 + beta6 * timeA[i];
         do j = (i+1) to stopA[i];
            b1 = v1 * logbunA[j] + v2 * hgbA[j] + v3 * plateletA[j];
            S[i] = S[i] + exp(b1);
         end;
      end;
      jl = jl + vstatusA[i] * (bZ[i] - log(S[i]));
   end;
   model general(jl);
run;
```

The multiple ARRAY statements allocate array symbols that are used to store the parameters (beta), the response (timeA), the covariates (vstatusA, logbunA, hgbA, plateletA, and stopA), and work space (bZ and S). The bZ and S arrays store the regression term and the risk set term for every observation. Programming statements in the BEGINCNST and ENDCNST statements input the response and covariates from the data set to the arrays.

Using the same technique shown in the example "Example 80.14: Time Independent Cox Model" on page 6437, the next DO loop calculates the regression term and corresponding S for every observation, pretending that there are no ties in time. This means that the risk set for observation *i* involves only observation 1 to *i*. The correction terms are added to the corresponding S[i] in the second DO loop, conditional on whether the stop variable is greater than the observation count itself. The symbol jl cumulatively adds the log likelihood for the entire data set, and the MODEL statement specifies the joint log-likelihood function.

The following statements run PROC COMPARE and show that the output data set outa contains identical posterior samples as outi:

```
proc compare data=outi compare=outa;
   ods select comparesummary;
   var beta1-beta6;
run;
```

The results are not shown here.

The following statements use PROC PHREG to fit the same time dependent Cox model:

```
proc phreg data=Myeloma;
   ods select PostSumInt;
   model Time*VStatus(0)=LogBUN z2 hgb z3 platelet z4;
   z2 = Time*logbun;
   z3 = Time*hgb;
   z4 = Time*platelet;
   bayes seed=1 nmc=10000 outpost=phout;
run;
```

Coding is simpler than PROC MCMC. See Output 80.15.2 for posterior summary and interval statistics:

Output 80.15.2 Summary Statistics on Cox Model with Time Dependent Explanatory Variables and Ties in the Survival Time, Using PROC PHREG

Cox Model with Time Dependent Covariates

The PHREG Procedure

Bayesian Analysis

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
LogBUN	10000	3.2171	0.8154	1.6453	4.8061
z2	10000	-0.1387	0.0470	-0.2308	-0.0478
HGB	10000	-0.0390	0.1001	-0.2407	0.1539
z3	10000	-0.00403	0.00358	-0.0112	0.00279
Platelet	10000	0.3656	0.7689	-1.1521	1.8144
z4	10000	-0.0412	0.0373	-0.1148	0.0312

Example 80.16: Piecewise Exponential Frailty Model

(View the complete code for this example (mcmcex16.sas) in the example repository.)

This example illustrates how to fit a piecewise exponential frailty model using PROC MCMC. Part of the notation and presentation in this example follows Clayton (1991) and the Luek example in Spiegelhalter et al. (1996a).

Generally speaking, the proportional hazards model assumes the hazard function,

$$\lambda_i(t|\mathbf{z}_i) = \lambda_0(t)\exp\left\{\boldsymbol{\beta}'\mathbf{z}_i\right\}$$

where $i = 1,\ldots,n$ indexes subject, $\lambda_0(t)$ is the baseline hazard function, and $\mathbf{z}_i$ are the covariates for subject i. If you define $N_i(t)$ to be the number of observed failures of the ith subject up to time t, then the hazard function for the ith subject can be seen as a special case of a *multiplicative intensity model* (Clayton 1991). The intensity process for $N_i(t)$ becomes

$$I_i(t) = Y_i(t)\lambda_0(t)\exp(\boldsymbol{\beta}'\mathbf{z}_i)$$

where $Y_i(t)$ indicates observation of the subject at time t (taking the value of 1 if the subject is observed and 0 otherwise). Under *noninformative censoring*, the corresponding likelihood is proportional to

$$\prod_{i=1}^{n}\left[\prod_{t\geq 0} I_i(t)\right]^{dN_i(t)} \exp\left[-\int_{t\geq 0} I_i(t)dt\right]$$

where $dN_i(t)$ is the increment of $N_i(t)$ over the small time interval $[t, t+dt)$: it takes a value of 1 if the subject i fails in the time interval, 0 otherwise. This is a Poisson kernel with the random variable being the increments of dN_i and the means $I_i(t)dt$

$$dN_i(t) \sim \text{Poisson}(I_i(t)dt)$$

where

$$I_i(t)dt = Y_i(t)\exp(\boldsymbol{\beta}'\mathbf{z})d\Lambda_0(t)$$

and

$$\Lambda_0(t) = \int_0^t \lambda_0(u)du.$$

The integral is the increment in the integrated baseline hazard function that occurs during the time interval $[t, t + dt)$.

This formulation provides an alternative way to fit a piecewise exponential model. You partition the time axis to a few intervals, where each interval has its own hazard rate, $\Lambda_0(t)$. You count the $Y_i(t)$ and $dN_i(t)$ in each interval, and fit a Poisson model to each count.

The following DATA step creates the data set Blind (Lin 1994) that represents 197 diabetic patients who have a high risk of experiencing blindness in both eyes as defined by DRS criteria:

```
title 'Piecewise Exponential Model';
data Blind;
   input ID Time Status DiabeticType Treatment @@;
   datalines;
   5 46.23 0 1 1    5 46.23 0 1 0   14 42.50 0 0 1   14 31.30 1 0 0
  16 42.27 0 0 1   16 42.27 0 0 0   25 20.60 0 0 1   25 20.60 0 0 0
  29 38.77 0 0 1   29  0.30 1 0 0   46 65.23 0 0 1   46 54.27 1 0 0
  49 63.50 0 0 1   49 10.80 1 0 0   56 23.17 0 0 1   56 23.17 0 0 0

   ... more lines ...

1705  8.00 0 0 1 1705  8.00 0 0 0 1717 51.60 0 1 1 1717 42.33 1 1 0
1727 49.97 0 1 1 1727  2.90 1 1 0 1746 45.90 0 0 1 1746  1.43 1 0 0
1749 41.93 0 1 1 1749 41.93 0 1 0
;
```

One eye of each patient is treated with laser photocoagulation. The hypothesis of interest is whether the laser treatment delays the occurrence of blindness. The following variables are included in Blind:

- ID, patient's identification
- Time, failure time
- Status, event indicator (0=censored and 1=uncensored)
- Treatment, treatment received (1=laser photocoagulation and 0=otherwise)
- DiabeticType, type of diabetes (0=juvenile onset with age of onset at 20 or under, and 1= adult onset with age of onset over 20)

For illustrational purposes, a piecewise exponential model that ignores the patient-level frailties is first fit to the entire data set. The formulation of the Poisson counting process makes it straightforward to add the frailty terms, as it is demonstrated later.

The following statements create a partition (of length 8) along the time axis, with $s_0 < s_1 < s_2 < \cdots < s_J$, with $s_0 = 0.1 < y_i$ and $s_J = 80 > y_i$ for all i. The time intervals are stored in the Partition data set:

```
data partition;
   input int_1-int_9;
   datalines;
   0.1  6.545  13.95  26.47  38.8  45.88  54.35  62 80
;
```

To obtain reasonable estimates, placing an equal number of observations in each interval is recommended. You can find the partition points by calculating the percentile statistics of the time variable (for example, by using the UNIVARIATE procedure).

The following regression model and prior distributions are used in the analysis:

$$\begin{aligned} \boldsymbol{\beta}'\mathbf{z}_i &= \beta_1 \text{treatment} + \beta_2 \text{diabetictype} + \beta_3 \text{treatment * diabetictype} \\ \beta_1, \beta_2, \beta_3 &\sim \text{normal}(0, \text{var } = 1e6) \\ \lambda_j &\sim \text{gamma}(\text{shape} = 0.01, \text{iscale} = 0.01) \quad \text{for} \quad j = 1, \ldots, 8 \end{aligned}$$

The following statements calculate $Y_i(t)$ for each observation i, at every time point t in the Partition data set. The statements also find the observed failure time interval, $dN_i(t)$, for each observation:

```
%let n = 8;
data _a;
   set blind;
   if _n_ eq 1 then set partition;
   array int[*] int_:;
   array Y[&n];
   array dN[&n];
   do k = 1 to (dim(int)-1);
      Y[k] = (time - int[k] + 0.001 >= 0);
      dN[k] = Y[k] * ( int[k+1] - time - 0.001 >= 0) * status;
   end;
   output;
   drop int_: k;
run;
```

The DATA step reads in the Blind data set. At the first observation, it also reads in the Partition data set. The first ARRAY statement creates the int array and name the elements int_:. Because the names match the variable names in the Partition data set, all values of the int_: variables (there is only one observation) in the Partition data set are therefore stored in the int array. The next two ARRAY statements create arrays Y and dN, each with length 8. They store values of $Y_i(t)$ and $dN_i(t)$, resulting from each failure time in the Blind data set.

The following statements print the first 10 observations of the constructed data set _a and display them in Output 80.16.1:

```
proc print data=_a(obs=10);
run;
```

Output 80.16.1 First 10 Observations of the Data Set _a

Piecewise Exponential Model

Obs	ID	Time	Status	DiabeticType	Treatment	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	dN1	dN2	dN3	dN4	dN5	dN6	dN7	dN8
1	5	46.23	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	5	46.23	0	1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
3	14	42.50	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
4	14	31.30	1	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0
5	16	42.27	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
6	16	42.27	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
7	25	20.60	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
8	25	20.60	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
9	29	38.77	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
10	29	0.30	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

The first subject in _a experienced blindness in the left eye at time 46.23, and the time falls in the sixth interval as defined in the Partition data set. Therefore, Y1 through Y6 all take a value of 1, and Y7 and Y8 are 0. The variable dN# takes on a value of 1 if the subject is observed to go blind in that interval. Since the first observation is censored (status == 1), the actual failure time is unknown. Hence all dN# are 0. The first observed failure time occurs in observation number 4 (the right eye of the second subject), where the time variable takes a value of 31.30, Y1 through Y4 are 1, and dN4 is 1.

Note that each observation in the _a data set has 8 Y and 8 dN, meaning that you would need eight MODEL statements in a PROC MCMC call, each for a Poisson likelihood. Alternatively, you can expand _a, put one Y and one dN in every observation, and fit the data using a single MODEL statement in PROC MCMC. The following statements expand the data set _a and save the results in the data set _b:

```
data _b;
   set _a;
   array y[*] y:;
   array dn[*] dn:;
   do i = 1 to (dim(y));
      y_val       = y[i];
      dn_val      = dn[i];
      int_index   = i;
      output;
   end;
   keep y_: dn_: diabetictype treatment int_index id;
run;

data _b;
   set _b;
   rename y_val=Y dn_val=dN;
run;
```

You can use the following PROC PRINT statements to see the first few observations in _b:

```
proc print data=_b(obs=10);
run;
```

Output 80.16.2 First 20 Observations of the Data Set _b

Obs	ID	DiabeticType	Treatment	Y	dN	int_index
1	5	1	1	1	0	1
2	5	1	1	1	0	2
3	5	1	1	1	0	3
4	5	1	1	1	0	4
5	5	1	1	1	0	5
6	5	1	1	1	0	6
7	5	1	1	0	0	7
8	5	1	1	0	0	8
9	5	1	0	1	0	1
10	5	1	0	1	0	2

The data set _b now contains 3,152 observations (see Output 80.16.2 for the first few observations). The Time and Status variables are no longer needed; hence they are discarded from the data set. The int_index variable is an index variable that indicates interval membership of each observation.

Because the variable Y does not contribute to the likelihood calculation when it takes a value of 0 (it amounts to a Poisson likelihood that has a mean and response variable that are both 0), you can remove these observations. This speeds up the calculation in PROC MCMC:

```
data inputdata;
   set _b;
   if Y > 0;
run;
```

The data set Inputdata has 1,775 observations, as opposed to 3,152 observations in _b. The following statements fit a piecewise exponential model in PROC MCMC:

```
proc mcmc data=inputdata nmc=10000 outpost=postout seed=12351
     maxtune=5;
   ods select PostSumInt ESS;
   parms beta1-beta3 0;
   prior beta: ~ normal(0, var = 1e6);
   random lambda ~ gamma(0.01, iscale = 0.01) subject=int_index;
   bZ = beta1*treatment + beta2*diabetictype + beta3*treatment*diabetictype;
   idt = exp(bz) * lambda;
   model dN ~ poisson(idt);
run;
```

The PARMS statement declares three regression parameters, beta1–beta3. The PRIOR statement specifies a noninformative normal prior on the regression coefficients. The RANDOM statement specifies the random effect, lambda, its prior distribution, and interval membership which is indexed by the data set variable int_index.

The symbol bZ calculates the regression mean, and the symbol idt is the mean of the Poisson likelihood. It corresponds to the equation

$$I_i(t)dt = Y_i(t)\exp(\boldsymbol{\beta}'\mathbf{z})d\Lambda_0(t)$$

Note that the $Y_i(t)$ term is omitted in the assignment statement because Y takes only the value of 1 in the input data set.

Output 80.16.3 displays posterior estimates of the three regression parameters.

Output 80.16.3 Posterior Summary Statistics

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta1	10000	-0.4174	0.2129	-0.8121	0.0203
beta2	10000	0.3138	0.1956	-0.0885	0.6958
beta3	10000	-0.7899	0.3308	-1.4300	-0.1046

To understand the results, you can create a 2×2 table (Table 80.55) and plug in the posterior mean estimates to the regression model. A –0.41 estimate for subjects who received laser treatment and had juvenile diabetes suggests that the laser treatment is effective in delaying blindness. And the effect is much more pronounced (–0.80) for adult subjects who have diabetes and received treatment.

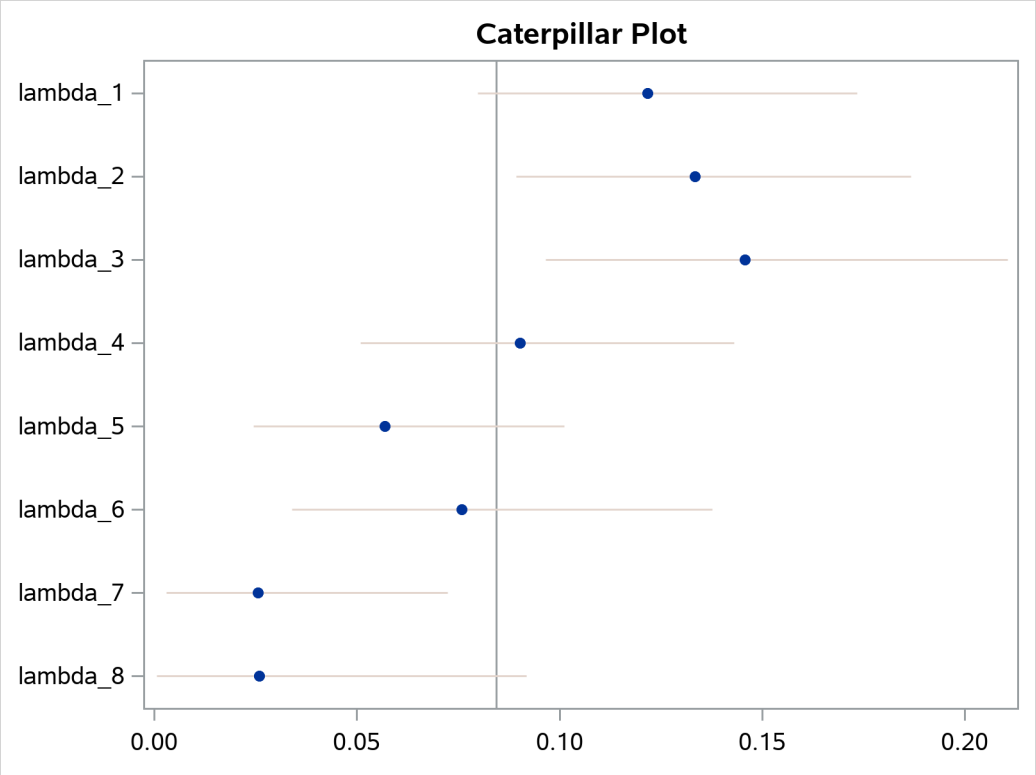
Table 80.55 Estimates of Regression Effects in the Survival Model

$\hat{\beta}'\mathbf{Z}$		Diabetic Type	
		0	1
Treatment	0	0	0.32
	1	–0.41	–0.80

You can also use the macro %CATER (“Caterpillar Plot” on page 6327) to draw a caterpillar plot to visualize the eight hazards in the model:

```
%cater(data=postout, var=lambda_:);
```

Output 80.16.4 Caterpillar Plot of the Hazards in the Piecewise Exponential Model



The fitted hazards show a nonconstant underlying hazard function (read along the y-axis as lambda_# are hazards along the time-axis) in the model.

Now suppose you want to include patient-level information and fit a frailty model to the blind data set, where the random effect enters the model through the regression term, where the subject is indexed by the variable ID in the data.

$$\begin{aligned} \boldsymbol{\beta}'\mathbf{z}_i &= \beta_1 \text{treatment} + \beta_2 \text{diabetictype} + \beta_3 \text{treatment} * \text{diabetictype} + u_{id} \\ u_{id} &\sim \text{normal}(0, \text{var} = \sigma^2) \\ \sigma^2 &\sim \text{igamma}(\text{shape} = 0.01, \text{scale} = 0.01) \end{aligned}$$

where id indexes patient.

The actual coding in PROC MCMC of a piecewise exponential frailty model is rather straightforward:

```
ods select none;
proc mcmc data=inputdata nmc=10000 outpost=postout seed=12351
     stats=summary diag=none;
   parms beta1-beta3 0 s2;
   prior beta: ~ normal(0, var = 1e6);
   prior s2 ~ igamma(0.01, scale=0.01);
   random lambda ~ gamma(0.01, iscale = 0.01) subject=int_index;
   random u ~ normal(0, var=s2) subject=id;
   bZ = beta1*treatment + beta2*diabetictype + beta3*treatment*diabetictype + u;
   idt = exp(bZ) * lambda;
```

```
   model dN ~ poisson(idt);
run;
```

A second RANDOM statement defines a subject-level random effect u, and the random-effects parameters enter the model in the term for the regression mean, bZ. An additional model parameter, s2, the variance of the random-effects parameters, is needed for the model. The results are not shown here.

Example 80.17: Normal Regression with Interval Censoring

(View the complete code for this example (mcmcex17.sas) in the example repository.)

You can use PROC MCMC to fit failure time data that can be right-, left-, or interval-censored. To illustrate, a normal regression model is used in this example.

You can use either of two approaches to fit interval-censored data. One is to specify the marginal model, and the other is to treat the censored data as missing values.

Assume that you have a simple regression model with no covariates,

$$\mathbf{y} = \mu + \sigma\boldsymbol{\epsilon}$$

where $\mathbf{y}$ is a vector of response values (the failure times), μ is the grand mean, σ is an unknown scale parameter, and $\boldsymbol{\epsilon}$ are errors from the standard normal distribution. Instead of observing y_i directly, you observe only a truncated value t_i. If the true y_i occurs after the censored time t_i, the data are called *right-censored*. If y_i occurs before the censored time, the data are called *left-censored*. A failure time y_i can be censored at both ends, and these data are called *interval-censored*. The likelihood for y_i is

$$p(y_i|\mu) = \begin{cases} \phi(y_i|\mu,\sigma) & \text{if } y_i \text{ is uncensored} \\ S(t_{l,i}|\mu) & \text{if } y_i \text{ is right-censored by } t_{l,i} \\ 1 - S(t_{r,i}|\mu) & \text{if } y_i \text{ is left-censored by } t_{r,i} \\ S(t_{l,i}|\mu) - S(t_{r,i}|\mu) & \text{if } y_i \text{ is interval-censored by } t_{l,i} \text{ and } t_{r,i} \end{cases}$$

where $S(\cdot)$ is the survival function and $S(t) = \Pr(T > t)$. When a datum is uncensored, you use a normal likelihood. When a datum is censored, you use the cumulative distribution to account for the likelihood.

Gentleman and Geyer (1994) uses the following data on cosmetic deterioration for early breast cancer patients who are treated with radiotherapy:

```
title 'Normal Regression with Interval Censoring';
data cosmetic;
   t = .;
   label tl = 'Time to Event (Months)';
   input tl tr @@;
   datalines;
45  .   6 10   .  7  46  .  46  .   7 16  17  .   7 14
37 44   .  8   4 11  15  .  11 15  22  .  46  .  46  .
25 37  46  .  26 40  46  .  27 34  36 44  46  .  36 48
37  .  40  .  17 25  46  .  11 18  38  .   5 12  37  .
 .  5  18  .  24  .  36  .   5 11  19 35  17 25  24  .
32  .  33  .  19 26  37  .  34  .  36  .
;
```

The data consist of time interval endpoints (in months). Nonmissing equal endpoints (tl = tr) indicate noncensoring; a nonmissing lower endpoint (tl $\neq$.) and a missing upper endpoint (tr = .) indicate right-censoring; a missing lower endpoint (tl = .) and a nonmissing upper endpoint (tr $\neq$.) indicate left-censoring; and nonmissing unequal endpoints (tl $\neq$ tr) indicate interval censoring. In this data set, all observations are censored (all t are missing).

With this data set, you can consider using proper but diffuse priors on both μ and σ. For example,

$$\begin{aligned} \mu &\sim \text{normal}(0, \text{sd} = 1000) \\ \sigma &\sim \text{gamma}(0.001, \text{iscale} = 0.001) \end{aligned}$$

The following SAS statements fit an interval-censoring model by using its marginal distribution and generate Output 80.17.1:

```
proc mcmc data=cosmetic outpost=postout seed=1 nmc=20000 missing=AC;
   ods select PostSumInt;
   parms mu 60 sigma 50;

   prior mu ~ normal(0, sd=1000);
   prior sigma ~ gamma(shape=0.001,iscale=0.001);

   if (tl^=. and tr^=. and tl=tr) then
      llike = logpdf('normal',tr,mu,sigma);
   else if (tl^=. and tr=.) then
      llike = logsdf('normal',tl,mu,sigma);
   else if (tl=. and tr^=.) then
      llike = logcdf('normal',tr,mu,sigma);
   else
      llike = log(sdf('normal',tl,mu,sigma) -
         sdf('normal',tr,mu,sigma));

   model general(llike);
run;
```

Because there are missing cells in the input data, you want to use the MISSING=AC option so that PROC MCMC does not delete any observations that contain missing values. The IF-ELSE statements distinguish different censoring cases for y_i according to the likelihood. The SAS functions LOGCDF, LOGSDF, LOGPDF, and SDF are useful here. The MODEL statement assigns llike as the log likelihood to the response. The Markov chain appears to have converged in this example (evidence not shown here), and the posterior estimates are shown in Output 80.17.1.

Output 80.17.1 Interval Censoring

Normal Regression with Interval Censoring

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
mu	20000	41.7807	5.7882	31.3604	53.6115
sigma	20000	29.1122	6.0503	19.4041	41.6742

The marginal model approach is more efficient because the censored observations are integrated out. However, you might not always have the cumulative distributions readily available in all scenarios. One general and alternative approach is to treat all censored variables as latent variables (or missing data). You fit the same model by imputing the would-be values in the censored data and estimating the model parameters. The censoring information, which specifies the restricted range of the unobserved variables, is specified in the CLOWER= and CUPPER= options.

The following SAS statements fit censored data by using the missing data approach:

```
proc mcmc data=cosmetic outpost=postout seed=117207154
   nmc=20000 missing=ACMODELY;
   ods select none;
   parms mu 60 sigma 50;

   prior mu ~ normal(0, sd=1000);
   prior sigma ~ gamma(shape=0.001,iscale=0.001);

   model t ~ normal(mu, sd=sigma, clower=tl, cupper=tr);
run;
```

By default, PROC MCMC discards observations that have missing values in covariate variables (a covariate is a data set variable that appears in the program but not to the left of the tilde in a MODEL statement). To keep observations that have missing tl or tr values, specify the MISSING=ACMODELY option, which keeps all observations and models the missing response variable. This approach produces estimates that are equivalent to those from the marginal model approach. The results are not shown here.

Example 80.18: Constrained Analysis

(View the complete code for this example (mcmcex18.sas) in the example repository.)

Conjoint analysis uses regression techniques to model consumer preferences and to estimate consumer utility functions. A problem with conventional conjoint analysis is that sometimes your estimated utilities do not make sense. Your results might suggest, for example, that the consumers would prefer to spend more on a product than to spend less. With PROC MCMC, you can specify constraints on the part-worth utilities (parameter estimates). Suppose that the consumer product being analyzed is an off-road motorcycle. The relevant attributes are how large each motorcycle is (less than 300cc, 301–550cc, and more than 551cc), how much it costs (less than $5000, $5001–$6000, $6001–$7000, and more than $7000), whether or not it has an electric starter, whether or not the engine is counter-balanced, and whether the bike is from Japan or Europe. The preference variable is a ranking of the bikes. You could perform an ordinary conjoint analysis with PROC TRANSREG (see Chapter 126, "The TRANSREG Procedure") as follows:

```
options validvarname=any;
proc format;
   value sizef  1 = '< 300cc' 2 = '300-550cc' 3 = '> 551cc';
   value pricef 1 = '< $5000' 2 = '$5000 - $6000'
                3 = '$6001 - $7000' 4 = '> $7000';
   value startf 1 = 'Electric Start' 2 = 'Kick Start';
   value balf   1 = 'Counter Balanced' 2 = 'Unbalanced';
   value orif   1 = 'Japanese' 2 = 'European';
run;
```

```
data bikes;
   input Size Price Start Balance Origin Rank @@;
   format size sizef. price pricef. start startf.
          balance balf. origin orif.;
   datalines;
2 1 2 1 2  3  1 4 2 2 2  7  1 2 1 1 2  6
3 3 1 1 2  1  1 3 2 1 1  5  3 4 2 2 2 12
2 3 2 2 1  9  1 1 1 2 1  8  2 2 1 2 2 10
2 4 1 1 1  4  3 1 1 2 1 11  3 2 2 1 1  2
;

title 'Ordinary Conjoint Analysis by PROC TRANSREG';
proc transreg data=bikes utilities cprefix=0 lprefix=0;
   ods select Utilities;
   model identity(rank / reflect) =
         class(size price start balance origin / zero=sum);
   output out=coded(drop=intercept) replace;
run;
```

The DATA step reads the experimental design and dependent variable Rank and assigns formats to label the factor levels. PROC TRANSREG is run specifying UTILITIES, which requests a conjoint analysis. The rank variable is reflected around its mean ($1 \rightarrow 12$, $2 \rightarrow 11$, $\ldots$, $12 \rightarrow 1$) so that in the analysis, larger part-worth utilities correspond to higher preference. The OUT=CODED data set contains the reflected ranks and a binary coding of the factors that can be used in other analyses. See Kuhfeld (2010) for more information about conjoint analysis and coding with PROC TRANSREG.

The Utilities table from the conjoint analysis is shown in Output 80.18.1. Notice the part-worth utilities for price. The part-worth utility for < \$5000 is 0.25. As price increases to the \$5000–\$6000 range, utility decreases to –0.5. Then as price increases to the \$6001–\$7000 range, part-worth utility *increases* to 0.5. Finally, for the most expensive bikes, utility decreases again to –0.25. In cases like this, you might want to impose constraints on the solution so that the part-worth utility for price never increases as prices go up.

Output 80.18.1 Ordinary Conjoint Analysis by PROC TRANSREG

Ordinary Conjoint Analysis by PROC TRANSREG

The TRANSREG Procedure

Utilities Table Based on the Usual Degrees of Freedom

Label	Utility	Standard Error	Importance (% Utility Range)	Variable
Intercept	6.5000	0.95743		Intercept
< 300cc	-0.0000	1.35401	0.000	Class.< 300cc
300-550cc	-0.0000	1.35401		Class.300-550cc
> 551cc	0.0000	1.35401		Class.> 551cc
< $5000	0.2500	1.75891	13.333	Class.< $5000
$5000 - $6000	-0.5000	1.75891		Class.$5000 - $6000
$6001 - $7000	0.5000	1.75891		Class.$6001 - $7000
> $7000	-0.2500	1.75891		Class.> $7000
Electric Start	-0.1250	1.01550	3.333	Class.Electric Start
Kick Start	0.1250	1.01550		Class.Kick Start
Counter Balanced	3.0000	1.01550	80.000	Class.Counter Balanced
Unbalanced	-3.0000	1.01550		Class.Unbalanced
Japanese	-0.1250	1.01550	3.333	Class.Japanese
European	0.1250	1.01550		Class.European

You could run PROC TRANSREG again, specifying monotonicity constraints on the part-worth utilities for price:

```
title 'Constrained Conjoint Analysis by PROC TRANSREG';
proc transreg data=bikes utilities cprefix=0 lprefix=0;
   ods select ConservUtilities;
   model identity(rank / reflect) =
         monotone(price / tstandard=center)
         class(size start balance origin / zero=sum);
run;
```

The output from this PROC TRANSREG step is shown in Output 80.18.2.

Output 80.18.2 Constrained Conjoint Analysis by PROC TRANSREG

Constrained Conjoint Analysis by PROC TRANSREG

The TRANSREG Procedure

Utilities Table Based on Conservative Degrees of Freedom

Label	Utility	Standard Error	Importance (% Utility Range)	Variable
Intercept	6.5000	0.97658		Intercept
Price	-0.1581	.	7.143	Monotone(Price)
< $5000	0.2500	.		
$5000 - $6000	0.0000	.		
$6001 - $7000	0.0000	.		
> $7000	-0.2500	.		
< 300cc	-0.0000	1.38109	0.000	Class.< 300cc
300-550cc	-0.0000	1.38109		Class.300-550cc
> 551cc	0.0000	1.38109		Class.> 551cc
Electric Start	-0.2083	1.00663	5.952	Class.Electric Start
Kick Start	0.2083	1.00663		Class.Kick Start
Counter Balanced	3.0000	0.97658	85.714	Class.Counter Balanced
Unbalanced	-3.0000	0.97658		Class.Unbalanced
Japanese	-0.0417	1.00663	1.190	Class.Japanese
European	0.0417	1.00663		Class.European

This monotonicity constraint is one of the few constraints on the part-worth utilities that you can specify in PROC TRANSREG. In contrast, PROC MCMC enables you to specify any constraint that can be written in the DATA step language. You can perform the restricted conjoint analysis with PROC MCMC by using the coded factors that were output from PROC TRANSREG. The data set is Coded.

The likelihood is a simple regression model:

$$\text{rank}_i \sim \text{normal}(\mathbf{x}_i'\boldsymbol{\beta}, \sigma)$$

where rank is the response, the covariates are '< 300cc'n, '300-500cc'n, '< $5000'n, '$5000 - $6000'n, '$6001 - $7000'n, 'Electric Start'n, 'Counter Balanced'n, and Japanese. Note that OPTIONS VALIDVARNAME=ANY enables PROC TRANSREG to create names for the coded variables with blanks and special characters. That is why the name-literal notation ('*variable-name*'n) is used for the input data set variables.

Suppose that there are two constraints you want to put on some of the parameters: one is that the parameters for '< $5000'n, '$5000 - $6000'n, and '$6001 - $7000'n decrease in order, and the other is that the parameter for 'Counter Balanced'n is strictly positive. You can consider a truncated multivariate normal prior as follows:

$$(\beta_{\text{'< \$5000'n}}, \beta_{\text{'\$5000 - \$6000'n}}, \beta_{\text{'\$6001 - \$7000'n}}, \beta_{\text{'Counter Balanced'n}}) \sim \text{MVN}(0, \sigma\mathbf{I})$$

with the following set of constraints:

$$\begin{aligned} \beta_{\text{'< \$5000'n}} &> \beta_{\text{'\$5000 - \$6000'n}} > \beta_{\text{'\$6001 - \$7000'n}} > 0 \\ \beta_{\text{'Counter Balanced'n}} &> 0 \end{aligned}$$

The condition that $\beta_{\text{'\$6001 - \$7000'n}} > 0$ reflects an implied constraint that, by definition, 0 is the utility for the highest price range, > \$7000, which is the reference level for the binary coded price variable. The following statements fit the desired model:

```
title 'Bayesian Constrained Conjoint Analysis by PROC MCMC';
proc mcmc data=coded outpost=bikesout ntu=3000 nmc=50000
   propcov=quanew seed=448 diag=none;
   ods select PostSumInt;
   array pw[4] pw5000 pw5000_6000 pw6001_7000 pwCounterBalanced;
   array sigma[4,4];
   array mu[4];

   begincnst;
      call identity(sigma);
      call mult(sigma, 100, sigma);
      call zeromatrix(mu);
   endcnst;

   parms intercept pw300cc pw300_550cc pwElectricStart pwJapanese tau 1;
   parms pw5000 0.3 pw5000_6000 0.2 pw6001_7000 0.1 pwCounterBalanced 1;

   beginnodata;
   prior intercept pw300: pwE: pwJ: ~ normal(0, var=100);
   if (pw5000      >= pw5000_6000 & pw5000_6000 >= pw6001_7000 &
       pw6001_7000 >= 0           & pwCounterBalanced > 0) then
       lp = lpdfmvn(pw, mu, sigma);
   else
       lp = .;
   prior pw5000 pw5000_6000 pw6001_7000 pwC: ~ general(lp);
   prior tau  ~ gamma(0.01, iscale=0.01);
   endnodata;

   mean = intercept +
          pw300cc           * '< 300cc'n          +
          pw300_550cc       * '300-550cc'n        +
          pw5000            * '< $5000'n          +
          pw5000_6000       * '$5000 - $6000'n    +
          pw6001_7000       * '$6001 - $7000'n    +
          pwElectricStart   * 'Electric Start'n   +
          pwCounterBalanced * 'Counter Balanced'n +
          pwJapanese        * Japanese;
   model rank ~ normal(mean, prec=tau);
run;
```

The two ARRAY statements allocate a 4×4 dimensional array for the prior covariance and an array of size 4 for the prior means. In the BEGINCNST and ENDCNST statements, the CALL IDENTITY function sets sigma to be an identity matrix; the CALL MULT function sets sigma's diagonal elements to be 100 (the diagonal variance terms); and the CALL ZEROMATRIX function sets mu to be a vector of zeros (the prior means). For matrix functions in PROC MCMC, see the section "Matrix Functions in PROC MCMC" on page 6298.

There are two PARMS statements, with each of them naming a block of parameters. The first PARMS statement blocks the following: the intercept, the two size parameters, the one start-type parameter, the one origin parameter, and the precision. The second PARMS statement blocks the three price parameters and the

one balance parameter, parameters that have the constraint multivariate normal prior. The second PARMS statement also specifies initial values for the parameter estimates. The initial values reflect the constraints on these parameters. The initial part-worth utilities all decrease from 0.3 to 0.2 to 0.1 to 0.0 (for the implicit reference level) as the prices increase. Also, the initial part-worth utility for the counter-balanced engine is set to a positive value, 1.

In the PRIOR statements, regression coefficients without constraints are given an independent normal prior with mean at 0 and variance of 100. The next IF-ELSE construction imposes the constraints. When these constraints are met, pw5000, pw5000_6000, pw6001_7000, pwCounterBalanced are jointly distributed as a multivariate normal prior with mean mu and covariance sigma. Otherwise, the prior is not defined and lp is assigned a missing value. The parameter tau is given a gamma prior, which is a conjugate prior for that parameter.

The model specification is linear. The mean is comprised of an intercept and the sum of terms like pw300cc * '< 300cc'n, which is a parameter times an input data set variable. The MODEL statement specifies that the linear model for rank is normally distributed with mean mean and precision tau.

The MCMC results are shown in Output 80.18.3.

Output 80.18.3 MCMC Results

Bayesian Constrained Conjoint Analysis by PROC MCMC

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
intercept	50000	2.1679	2.5351	-2.8636	7.3002
pw300cc	50000	0.1095	2.4763	-4.7698	5.1516
pw300_550cc	50000	0.0961	2.4206	-4.9501	4.8323
pwElectricStart	50000	-1.0173	2.1034	-5.3289	3.0740
pwJapanese	50000	-0.4418	2.1367	-4.8215	3.9265
tau	50000	0.1134	0.0763	0.00805	0.2639
pw5000	50000	4.1792	2.1701	0.6291	8.4404
pw5000_6000	50000	2.5997	1.6701	0.1724	5.8191
pw6001_7000	50000	1.4686	1.2512	0.000053	3.9253
pwCounterBalanced	50000	5.7915	1.9897	1.4204	9.4765

The estimates of the part-worth utility for the price categories are ordered as expected. This agrees with the intuition that there is a higher preference for a less expensive motor bike when all other things are equal, and that is what you see when you look at the estimated posterior means for the price part-worths. The estimated standard deviations of the price part-worths in this model are of approximately the same order of magnitude as the posterior means. This indicates that the part-worth utilities for this subject are not significantly far from each other, and that this subject's ranking of the options was not significantly influenced by the difference in price.

One advantage of Bayesian analysis is that you can incorporate prior information in the data analysis. Constraints on the parameter space are one possible source of information that you might have before you examine the data. This example shows that it can be accomplished in PROC MCMC.

Example 80.19: Implement a New Sampling Algorithm

(View the complete code for this example (mcmcex19.sas) in the example repository.)

This example illustrates using the UDS statement to implement a new Markov chain sampler. The algorithm demonstrated here is proposed by Holmes and Held (2006), hereafter referred to as HH. They presented a Gibbs sampling algorithm for generating draws from the posterior distribution of the parameters in a probit regression model. The notation follows closely to HH.

The data used here is the remission data set from a PROC LOGISTIC example:

```
title 'Implement a New Sampling Algorithm';
data inputdata;
   input remiss cell smear infil li blast temp;
   ind = _n_;
   cnst = 1;
   label remiss='Complete Remission';
   datalines;
   1  0.8   0.83  0.66  1.9  1.1    0.996

   ... more lines ...

   0  1     0.73  0.73  0.7  0.398  0.986
;
```

The variable remiss is the cancer remission indicator variable with a value of 1 for remission and a value of 0 for nonremission. There are six explanatory variables: cell, smear, infil, li, blast, and temp. These variables are the risk factors thought to be related to cancer remission. The binary regression model is as follows:

$$\text{remiss}_i \sim \text{binary}(p_i)$$

where the covariates are linked to p_i through a probit transformation:

$$\text{probit}(p_i) = \mathbf{x}'\boldsymbol{\beta}$$

$\boldsymbol{\beta}$ are the regression coefficients and $\mathbf{x}'$ the explanatory variables. Suppose you want to use independent normal priors on the regression coefficients:

$$\beta_i \sim \text{normal}(0, \text{var} = 25)$$

Fitting a probit model with PROC MCMC is straightforward. You can use the following statements:

```
proc mcmc data=inputdata nmc=100000 propcov=quanew seed=17
          outpost=mcmcout;
   ods select PostSumInt ess;
   parms beta0-beta6;
   prior beta: ~ normal(0,var=25);
   mu = beta0 + beta1*cell + beta2*smear +
        beta3*infil +  beta4*li + beta5*blast +  beta6*temp;
   p = cdf('normal', mu, 0, 1);
   model remiss ~ bern(p);
run;
```

The expression mu is the regression mean, and the CDF function links mu to the probability of remission p in the binary likelihood.

The summary statistics and effective sample sizes tables are shown in Output 80.19.1. There are high autocorrelations among the posterior samples, and efficiency is relatively low. The correlation time is reduced only after a large amount of thinning.

Output 80.19.1 Random Walk Metropolis

Implement a New Sampling Algorithm

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta0	100000	-2.0107	3.8405	-9.2214	5.7105
beta1	100000	2.5452	2.8012	-2.8579	8.0920
beta2	100000	-0.8095	3.2102	-7.0811	5.4883
beta3	100000	1.5889	3.5031	-5.3397	8.4183
beta4	100000	2.0270	0.8836	0.3722	3.8051
beta5	100000	-0.2896	0.9572	-2.1911	1.5439
beta6	100000	-3.2557	3.8146	-10.4698	4.5242

Implement a New Sampling Algorithm

The MCMC Procedure

Effective Sample Sizes

Parameter	ESS	Autocorrelation Time	Efficiency
beta0	4525.6	22.0963	0.0453
beta1	4687.0	21.3358	0.0469
beta2	3476.1	28.7677	0.0348
beta3	3941.4	25.3719	0.0394
beta4	3602.6	27.7580	0.0360
beta5	3437.2	29.0931	0.0344
beta6	4290.3	23.3086	0.0429

As an alternative to the random walk Metropolis, you can use the Gibbs algorithm to sample from the posterior distribution. The Gibbs algorithm is described in the section "Gibbs Sampler" on page 157 in Chapter 8, "Introduction to Bayesian Analysis Procedures." While the Gibbs algorithm generally applies to a wide range of statistical models, the actual implementation can be problem-specific. In this example, performing a Gibbs sampler involves introducing a class of auxiliary variables (also known as latent variables). You first reformulate the model by adding a z_i for each observation in the data set:

$$
\begin{aligned}
y_i &= \begin{cases} 1 & \text{if } z_i > 0 \\ 0 & \text{otherwise} \end{cases} \\
z_i &= \mathbf{x}_i'\boldsymbol{\beta} + \epsilon_i \\
\epsilon &\sim \text{normal}(0, 1) \\
\boldsymbol{\beta} &\sim \pi(\boldsymbol{\beta})
\end{aligned}
$$

If $\boldsymbol{\beta}$ has a normal prior, such as $\pi(\boldsymbol{\beta}) = N(\mathbf{b}, \mathbf{v})$, you can work out a closed form solution to the full conditional distribution of $\boldsymbol{\beta}$ given the data and the latent variables z_i. The full conditional distribution is also a multivariate normal, due to the conjugacy of the problem. See the section "Conjugate Priors" on page 151 in Chapter 8, "Introduction to Bayesian Analysis Procedures." The formula is shown here:

$$\begin{aligned} \boldsymbol{\beta}|\mathbf{z},\mathbf{x} &\sim \text{normal}(\mathbf{B},\mathbf{V}) \\ \mathbf{B} &= \mathbf{V}((v)^{-1}\mathbf{b} + \mathbf{x}'\mathbf{z}) \\ \mathbf{V} &= (\mathbf{v}^{-1} + \mathbf{x}'\mathbf{x})^{-1} \end{aligned}$$

The advantage of creating the latent variables is that the full conditional distribution of $\mathbf{z}$ is also easy to work with. The distribution is a truncated normal distribution:

$$z_i|\boldsymbol{\beta}, \mathbf{x}_i, y_i \sim \begin{cases} \text{normal}(\mathbf{x}_i\boldsymbol{\beta}, 1)I(z_i > 0) & \text{if } y_i = 1 \\ \text{normal}(\mathbf{x}_i\boldsymbol{\beta}, 1)I(z_i \le 0) & \text{otherwise} \end{cases}$$

You can sample $\boldsymbol{\beta}$ and z iteratively, by drawing $\boldsymbol{\beta}$ given z and vice verse. HH point out that a high degree of correlation could exist between $\boldsymbol{\beta}$ and z, and it makes this iterative way of sampling inefficient. As an improvement, HH proposed an algorithm that samples $\boldsymbol{\beta}$ and $\mathbf{z}$ jointly. At each iteration, you sample z_i from the posterior marginal distribution (this is the distribution that is conditional only on the data and not on any parameters) and then sample $\boldsymbol{\beta}$ from the same posterior full conditional distribution as described previously:

1. Sample z_i from its posterior marginal distribution:

$$\begin{aligned} z_i|\mathbf{z}_{-i}, y_i &\sim \begin{cases} \text{normal}(m_i, v_i)I(z_i > 0) & \text{if } y_i = 1 \\ \text{normal}(m_i, v_i)I(z_i \le 0) & \text{otherwise} \end{cases} \\ m_i &= \mathbf{x}_i\mathbf{B} - w_i(z_i - \mathbf{x}_i\mathbf{B}) \\ v_i &= 1 + w_i \\ w_i &= h_i/(1 - h_i) \\ h_i &= (\mathbf{H})_i i, \mathbf{H} = \mathbf{x}\mathbf{V}\mathbf{x}' \end{aligned}$$

2. Sample $\boldsymbol{\beta}$ from the same posterior full conditional distribution described previously.

For a detailed description of each of the conditional terms, refer to the original paper.

PROC MCMC cannot sample from the probit model by using this sampling scheme but you can implement the algorithm by using the UDS statement. To sample z_i from its marginal, you need a function that draws random variables from a truncated normal distribution. The functions, RLTNORM and RRTNORM, generate left- and right-truncated normal variates, respectively. The algorithm is taken from Robert (1995).

The functions are written in PROC FCMP (see the FCMP Procedure in the *Base SAS Procedures Guide*):

```
proc fcmp outlib=sasuser.funcs.uds;
   /*****************************************/
   /* Generate left-truncated normal variate */
   /*****************************************/
   function rltnorm(mu,sig,lwr);
   if lwr<mu then do;
      ans = lwr-1;
```

```
      do while(ans<lwr);
         ans = rand('normal',mu,sig);
      end;
   end;
   else do;
      mul = (lwr-mu)/sig;
      alpha = (mul + sqrt(mul**2 + 4))/2;
      accept=0;
      do while(accept=0);
         z = mul + rand('exponential')/alpha;
         lrho = -(z-alpha)**2/2;
         u = rand('uniform');
         lu = log(u);
         if lu <= lrho then accept=1;
      end;
      ans = sig*z + mu;
   end;
   return(ans);
   endsub;

   /****************************************/
   /* Generate right-truncated normal variate */
   /****************************************/
   function rrtnorm(mu,sig,uppr);
   ans = 2*mu - rltnorm(mu,sig, 2*mu-uppr);
   return(ans);
   endsub;
run;
```

The function call to RLTNORM(mu,sig,lwr) generates a random number from the left-truncated normal distribution:

$$\theta \sim \text{normal}(\text{mu}, \text{sd} = \text{sig}) I(\theta > \text{lwr})$$

Similarly, the function call to RRTNORM(mu,sig,uppr) generates a random number from the right-truncated normal distribution:

$$\theta \sim \text{normal}(\text{mu}, \text{sd} = \text{sig}) I(\theta < \text{uppr})$$

These functions are used to generate the latent variables z_i.

Using the algorithm A1 from the HH paper as an example, Output 80.56 lists a line-by-line implementation with the PROC MCMC coding style. The table is broken into three portions: set up the constants, initialize the parameters, and sample one draw from the posterior distribution. The left column of the table is identical to the A1 algorithm stated in the appendix of HH. The right column of the table lists SAS statements.

Table 80.56 Holmes and Held (2006), algorithm A1. Side-by-Side Comparison to SAS

Define Constants	In the BEGINCNST/ENDCNST Statements

$V \leftarrow (X'X + v^{-1})^{-1}$	`call transpose(x,xt); /* xt = transpose(x) */`
	`call mult(xt,x,xtx);`
	`call inv(v,v); /* v = inverse(v) */`
	`call addmatrix(xtx,v,xtx); /* xtx = xtx+v */`
	`call inv(xtx,v); /* v = inverse(xtx) */`

$L \leftarrow \text{Chol}(V)$	`call chol(v,L);`
$S \leftarrow VX'$	`call mult(v,xt,S);`

FOR $j = 1$ to n
$H[j] \leftarrow X[j,]S[,j]$
$W[j] \leftarrow H[j]/(1 - H[j])$
$Q[j] \leftarrow W[j] + 1$
END
	`call mult(x,S,HatMat);`
	`do j=1 to &n;`
	`H = HatMat[j,j];`
	`W[j] = H/(1-H);`
	`sQ[j] = sqrt(W[j] + 1); /* use s.d.  in SAS */`
	`end;`

Initial Values	In the BEGINCNST/ENDCNST Statements

$Z \sim \text{normal}(0, I_n)\text{Ind}(Y, Z)$	`do j=1 to &n;`
	`if(y[j]=1) then`
	`Z[j] = rltnorm(0,1,0);`
	`else`
	`Z[j] = rrtnorm(0,1,0);`
	`end;`

$B \leftarrow SZ$	`call mult(S,Z,B);`

Table 80.56 *continued*

Draw One Sample	Subroutine HH

FOR $j = 1$ to n
$z_{old} \leftarrow Z[j]$
$m \leftarrow X[j,]B$
$m \leftarrow m - W[j](Z[j] - m)$
$Z[j] \sim \text{normal}(m, Q[j])\text{Ind}(Y[j], Z[j])$
$B \leftarrow B + (Z[j] - z_{old})S[, j]$
END
$T \sim \text{normal}(0, I_p)$
$\beta[, i] \leftarrow B + LT$
	`do j=1 to &n;`
	`  zold = Z[j];`
	`  m = 0;`
	`  do k= 1 to &p;`
	`  m = m + X[j,k] * B[k];`
	`  end;`
	`  m = m - W[j]*(Z[j]-m);`
	`  if (y[j]=1) then`
	`  Z[j] = rltnorm(m,sQ[j],0);`
	`  else`
	`  Z[j] = rrtnorm(m,sQ[j],0);`
	`  diff = Z[j] - zold;`
	`  do k= 1 to &p;`
	`  B[k] = B[k] + diff * S[k,j];`
	`  end;`
	`  end;`
	`do j = 1 to &p;`
	`  T[j] = rand('normal');`
	`  end;`
	`call mult(L,T,T);`
	`call addmatrix(B,T,beta);`

The following statements define the subroutine HH (algorithm A1) in PROC FCMP and store it in library **sasuser.funcs.uds**:

```
/* define the HH algorithm in PROC FCMP. */
%let n = 27;
%let p = 7;
options cmplib=sasuser.funcs;
proc fcmp outlib=sasuser.funcs.uds;
   subroutine HH(beta[*],Z[*],B[*],x[*,*],y[*],W[*],sQ[*],S[*,*],L[*,*]);
   outargs beta, Z, B;
   array T[&p] / nosym;
   do j=1 to &n;
      zold = Z[j];
      m = 0;
      do k = 1 to &p;
         m = m + X[j,k] * B[k];
      end;
      m = m - W[j]*(Z[j]-m);
      if (y[j]=1) then
         Z[j] = rltnorm(m,sQ[j],0);
      else
         Z[j] = rrtnorm(m,sQ[j],0);
      diff = Z[j] - zold;
      do k = 1 to &p;
```

```
         B[k] = B[k] + diff * S[k,j];
      end;
   end;
   do j=1 to &p;
      T[j] = rand('normal');
   end;
   call mult(L,T,T);
   call addmatrix(B,T,beta);
   endsub;
run;
```

Note that one-dimensional array arguments take the form of *name*[*] and two-dimensional array arguments take the form of *name*[*,*]. Three variables, beta, Z, and B, are OUTARGS variables, making them the only arguments that can be modified in the subroutine. For the UDS statement to work, all OUTARGS variables have to be model parameters. Technically, only beta and Z are model parameters, and B is not. The reason that B is declared as an OUTARGS is because the array must be updated throughout the simulation, and this is the only way to modify its values. The input array x contains all of the explanatory variables, and the array y stores the response. The rest of the input arrays, W, sQ, S, and L, store constants as detailed in the algorithm. The following statements illustrate how to fit a Bayesian probit model by using the HH algorithm:

```
options cmplib=sasuser.funcs;

proc mcmc data=inputdata nmc=5000 monitor=(beta) outpost=hhout;
   ods select PostSumInt ess;
   array xtx[&p,&p];      /* work space                          */
   array xt[&p,&n];       /* work space                          */
   array v[&p,&p];        /* work space                          */
   array HatMat[&n,&n];   /* work space                          */
   array S[&p,&n];        /* V * Xt                              */
   array W[&n];
   array y[1]/ nosymbols; /* y stores the response variable      */
   array x[1]/ nosymbols; /* x stores the explanatory variables */
   array sQ[&n];          /* sqrt of the diagonal elements of Q */
   array B[&p];           /* conditional mean of beta            */
   array L[&p,&p];        /* Cholesky decomp of conditional cov */
   array Z[&n];           /* latent variables Z                  */
   array beta[&p] beta0-beta6;   /* regression coefficients      */

   begincnst;
      call streaminit(83101);
      if ind=1 then do;
         rc = read_array("inputdata", x, "cnst", "cell", "smear", "infil",
                         "li", "blast", "temp");
         rc = read_array("inputdata", y, "remiss");
         call identity(v);
         call mult(v, 25, v);
         call transpose(x,xt);
         call mult(xt,x,xtx);
         call inv(v,v);
         call addmatrix(xtx,v,xtx);
         call inv(xtx,v);
         call chol(v,L);
         call mult(v,xt,S);
```

```
            call mult(x,S,HatMat);
            do j=1 to &n;
               H = HatMat[j,j];
               W[j] = H/(1-H);
               sQ[j] = sqrt(W[j] + 1);
            end;

            do j=1 to &n;
               if(y[j]=1) then
                   Z[j] = rltnorm(0,1,0);
               else
                   Z[j] = rrtnorm(0,1,0);
            end;
            call mult(S,Z,B);
         end;
      endcnst;

      uds HH(beta,Z,B,x,y,W,sQ,S,L);
      parms z: beta: 0 B1-B7 / uds;
      prior z: beta: B1-B7 ~ general(0);

      model general(0);
   run;
```

The OPTIONS statement names the catalog of FCMP subroutines to use. The cmplib library stores the subroutine HH. You do not need to set a random number seed in the PROC MCMC statement because all random numbers are generated from the HH subroutine. The initialization of the rand function is controlled by the streaminit function, which is called in the program with a seed value of 83101.

A number of arrays are allocated. Some of them, such as xtx, xt, v, and HatMat, allocate work space for constant arrays. Other arrays are used in the subroutine sampling. Explanations of the arrays are shown in comments in the statements.

In the BEGINCNST and ENDCNST statement block, you read data set variables in the arrays x and y, calculate all the constant terms, and assign initial values to Z and B. For the READ_ARRAY function, see the section "READ_ARRAY Function" on page 6235. For listings of all array functions and their definitions, see the section "Matrix Functions in PROC MCMC" on page 6298.

The UDS statement declares that the subroutine HH is used to sample the parameters beta, Z, and B. You also specify the UDS option in the PARMS statement. Because all parameters are updated through the UDS interface, it is not necessary to declare the actual form of the prior for any of the parameters. Each parameter is declared to have a prior of general(0). Similarly, it is not necessary to declare the actual form of the likelihood. The MODEL statement also takes a flat likelihood of the form `general(0)`.

Summary statistics and effective sample sizes are shown in Output 80.19.2. The posterior estimates are very close to what was shown in Output 80.19.1. The HH algorithm produces samples that are much less correlated.

Output 80.19.2 Holms and Held

Implement a New Sampling Algorithm

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta0	5000	-2.0567	3.8260	-9.4031	5.2733
beta1	5000	2.7254	2.8079	-2.3940	8.5828
beta2	5000	-0.8318	3.2017	-6.6219	5.8170
beta3	5000	1.6319	3.5108	-5.7117	7.9353
beta4	5000	2.0567	0.8800	0.3155	3.7289
beta5	5000	-0.3473	0.9490	-2.1478	1.5889
beta6	5000	-3.3787	3.7991	-10.6821	4.1930

Implement a New Sampling Algorithm

The MCMC Procedure

Effective Sample Sizes

Parameter	ESS	Autocorrelation Time	Efficiency
beta0	3651.3	1.3694	0.7303
beta1	1563.8	3.1973	0.3128
beta2	5005.9	0.9988	1.0012
beta3	4853.2	1.0302	0.9706
beta4	2611.2	1.9148	0.5222
beta5	3049.2	1.6398	0.6098
beta6	3503.2	1.4273	0.7006

It is interesting to compare the two approaches of fitting a generalized linear model. The random walk Metropolis on a seven-dimensional parameter space produces autocorrelations that are substantially higher than the HH algorithm. A much longer chain is needed to produce roughly equivalent effective sample sizes. On the other hand, the Metropolis algorithm is faster to run. The running time of these two examples is roughly the same, with the random walk Metropolis with 100000 samples, a 20-fold increase over that in the HH algorithm example. The speed difference can be attributed to a number of factors, ranging from the implementation of the software and the overhead cost of calling PROC FCMP subroutine and functions. In addition, the HH algorithm requires more parameters by creating an equal number of latent variables as the sample size. Sampling more parameters takes time. A larger number of parameters also increases the challenge in convergence diagnostics, because it is imperative to have convergence in all parameters before you make valid posterior inferences. Finally, you might feel that coding in PROC MCMC is easier. However, this really is not a fair comparison to make here. Writing a Metropolis algorithm from scratch would have probably taken just as much, if not more, effort than the HH algorithm.

Example 80.20: Using a Transformation to Improve Mixing

(View the complete code for this example (mcmcex20.sas) in the example repository.)

Proper transformations of parameters can often improve the mixing in PROC MCMC. You already saw this in "Example 80.6: Nonlinear Poisson Regression Models" on page 6392, which sampled using the log scale of parameters that priors that are strictly positive, such as the gamma priors. This example shows another useful transformation: the logit transformation on parameters that take a uniform prior on [0, 1].

The data set is taken from Sharples (1990). It is used in Chaloner and Brant (1988) and Chaloner (1994) to identify outliers in the data set in a two-level hierarchical model. Congdon (2003) also uses this data set to demonstrates the same technique. This example uses the data set to illustrate how mixing can be improved using transformation and does not address the question of outlier detection as in those papers. The following statements create the data set:

```
data inputdata;
   input nobs grp y @@;
   ind = _n_;
   datalines;
1 1 24.80 2 1 26.90 3 1 26.65
4 1 30.93 5 1 33.77 6 1 63.31
1 2 23.96 2 2 28.92 3 2 28.19
4 2 26.16 5 2 21.34 6 2 29.46
1 3 18.30 2 3 23.67 3 3 14.47
4 3 24.45 5 3 24.89 6 3 28.95
1 4 51.42 2 4 27.97 3 4 24.76
4 4 26.67 5 4 17.58 6 4 24.29
1 5 34.12 2 5 46.87 3 5 58.59
4 5 38.11 5 5 47.59 6 5 44.67
;
```

There are five groups (grp, $j = 1, \ldots, 5$) with six observations (nobs, $i = 1, \ldots, 6$) in each. The two-level hierarchical model is specified as follows:

$$
\begin{aligned}
y_{ij} &\sim \text{normal}(\theta_j, \text{prec} = \tau_w) \\
\theta_j &\sim \text{normal}(\mu, \text{prec} = \tau_b) \\
\mu &\sim \text{normal}(0, \text{prec} = 1e-6) \\
\tau &\sim \text{gamma}(0.001, \text{iscale} = 0.001) \\
p &\sim \text{uniform}(0, 1)
\end{aligned}
$$

with the precision parameters related to each other in the following way:

$$
\begin{aligned}
\tau_b &= \tau / p \\
\tau_w &= \tau_b - \tau
\end{aligned}
$$

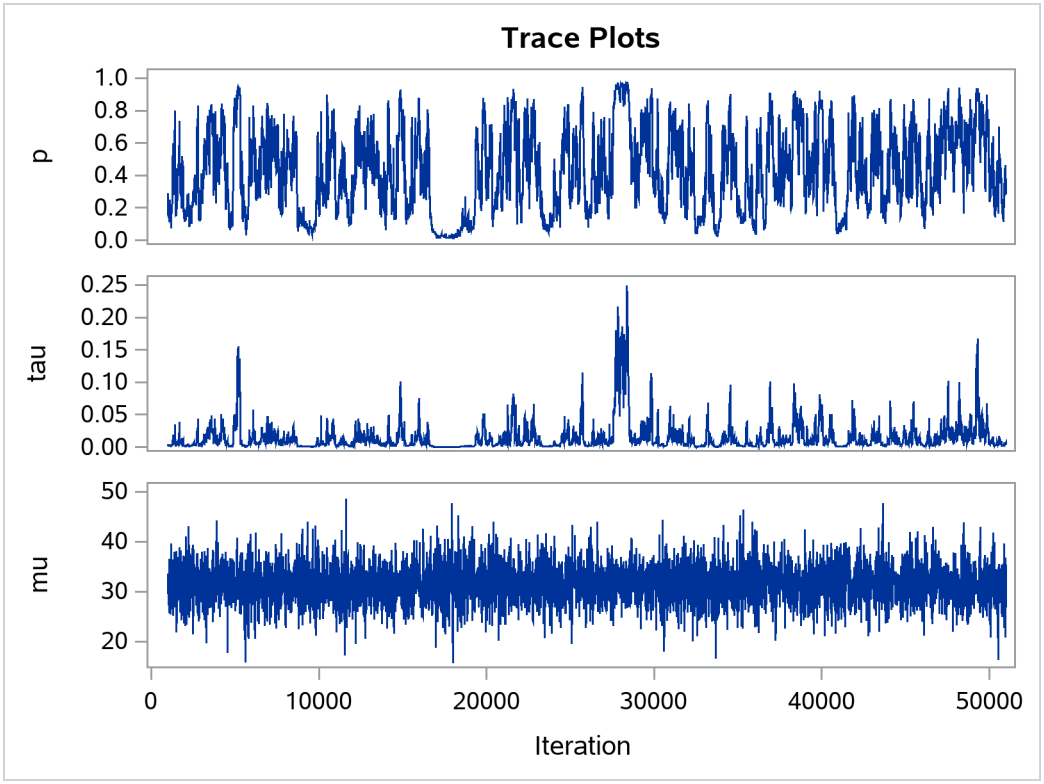
The total number of parameters in this model is eight: $\theta_1, \ldots, \theta_5, \mu, \tau$, and p.

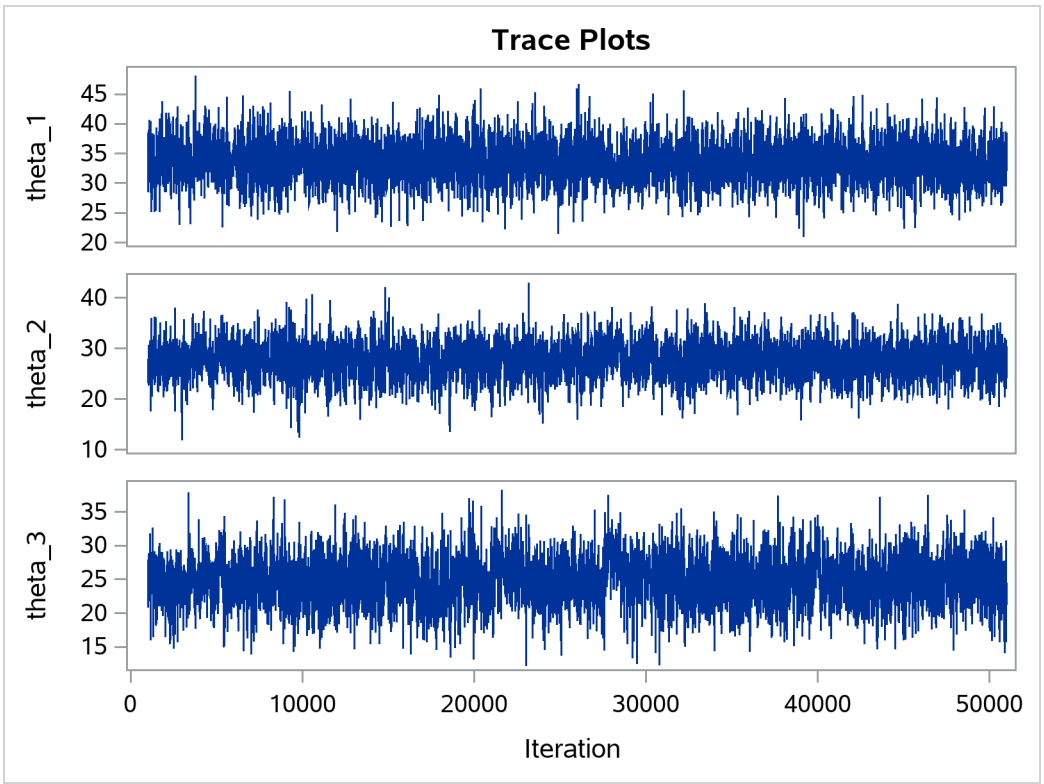
The following statements fit the model:

```
ods graphics on;
proc mcmc data=inputdata nmc=50000 thin=10 ntu=2000
          outpost=m1 seed=17797 plot=trace;
   parms p;
   parms tau;
   parms mu;

   prior p ~ uniform(0,1);
   prior tau ~ gamma(shape=0.001,iscale=0.001);
   prior mu ~ normal(0,prec=0.00000001);
   beginnodata;
   taub = tau/p;
   tauw = taub-tau;
   endnodata;

   random theta ~ normal(mu, prec=taub) subject=grp monitor=(theta);
   model y ~ normal(theta,prec=tauw);
run;
```

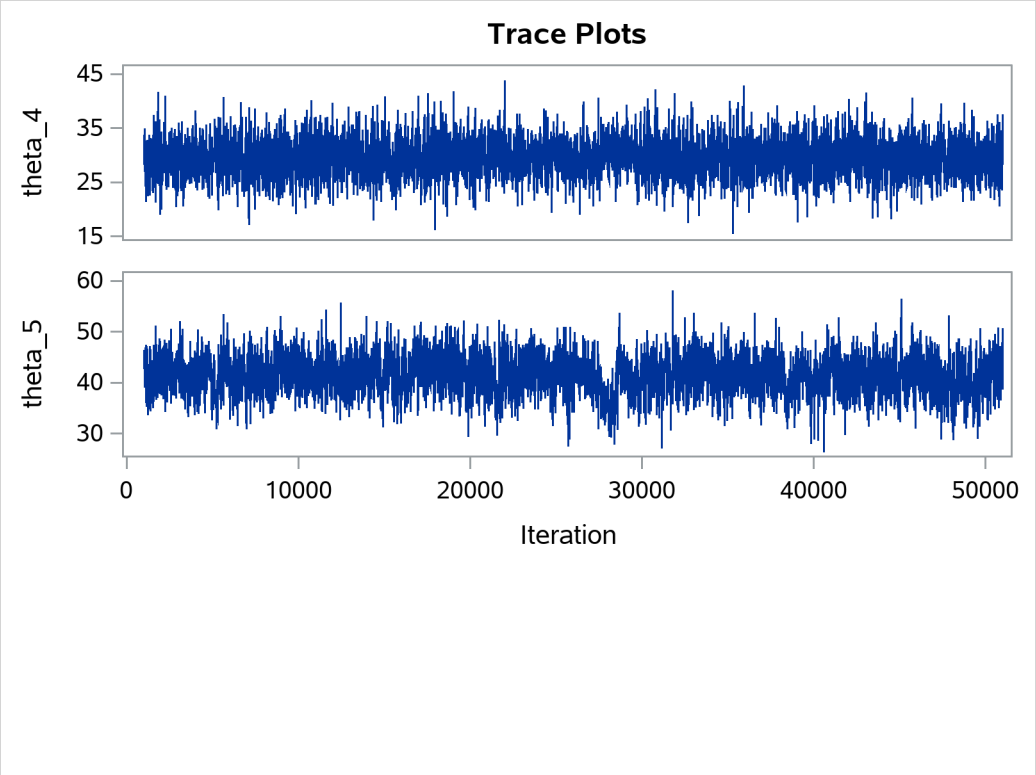
The ODS SELECT statement displays the effective sample size table and the trace plots. The ODS GRAPHICS ON statement enables ODS Graphics. The PROC MCMC statement specifies the usual options for the procedure run and produces trace plots (PLOTS=TRACE). The three PARMS statements put three model parameters, p, tau, and mu, in three different blocks. The PRIOR statements specify the prior distributions, and the programming statements enclosed with the BEGINNODATA and ENDNODATA statements calculate the transformation to taub and tauw. The RANDOM statement specifies the random effect, its prior distribution, and the subject variable. The resulting trace plots are shown in Output 80.20.1, and the effective sample size table is shown in Output 80.20.2.

Output 80.20.1 Trace Plots





Output 80.20.1 *continued*



Output 80.20.2 Bad Effective Sample Sizes

The MCMC Procedure

Effective Sample Sizes

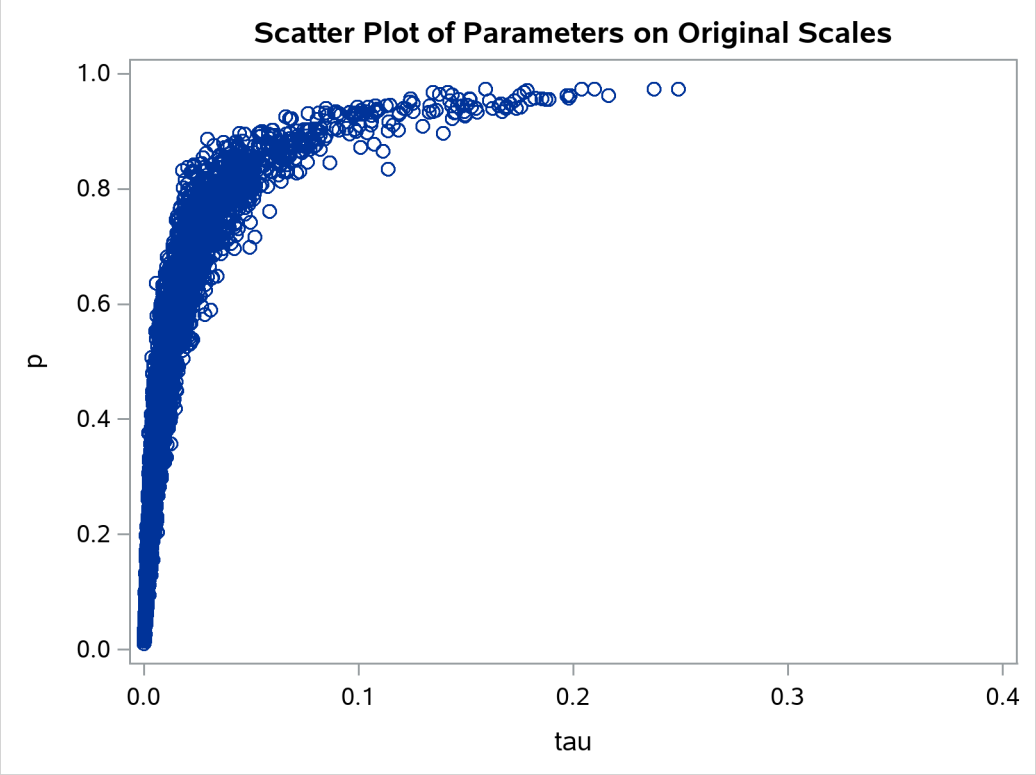
Parameter	ESS	Autocorrelation Time	Efficiency
p	76.3	65.5445	0.0153
tau	100.1	49.9391	0.0200
mu	4901.2	1.0202	0.9802
theta_1	4330.1	1.1547	0.8660
theta_2	4575.3	1.0928	0.9151
theta_3	1273.1	3.9274	0.2546
theta_4	5000.0	1.0000	1.0000
theta_5	621.8	8.0412	0.1244

The trace plots show that most parameters have relatively good mixing. Two exceptions appear to be p and τ. The trace plot of p shows a slow periodic movement. The τ parameter does not have good mixing either. When the values are close to zero, the chain stays there for periods of time. An inspection of the effective sample sizes table reveals the same conclusion: p and τ have much smaller ESSs than the rest of the parameters.

A scatter plot of the posterior samples of p and τ reveals why mixing is bad in these two dimensions. The following statements generate the scatter plot in Output 80.20.3:

```
title 'Scatter Plot of Parameters on Original Scales';

proc sgplot data=m1;
   yaxis label = 'p';
   xaxis label = 'tau' values=(0 to 0.4 by 0.1);
   scatter x = tau y = p;
run;
```

Output 80.20.3 Scatter Plot of τ versus p



The two parameters clearly have a nonlinear relationship. It is not surprising that the Metropolis algorithm does not work well here. The algorithm is designed for cases where the parameters are linearly related with each other.

To improve on mixing, you can sample on the log of τ, instead of sampling on τ. The formulation is:

$$\begin{aligned} \tau &\sim \text{gamma}(\text{shape} = 0.001, \text{iscale} = 0.001) \\ \log(\tau) &\sim \text{egamma}(\text{shape} = 0.001, \text{iscale} = 0.001) \end{aligned}$$

See the section "Standard Distributions" on page 6275 for the definitions of the gamma and egamma distributions. In addition, you can sample on the logit of p. Note that

$$p \sim \text{uniform}(0, 1)$$

is equivalent to

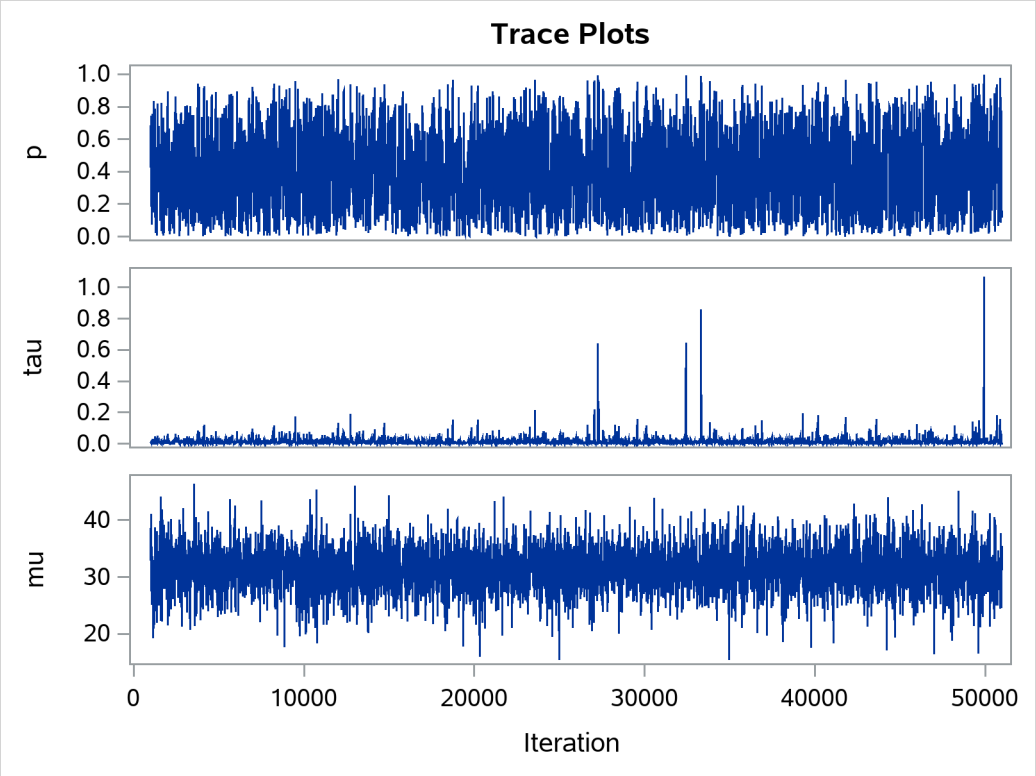
$$\text{lgp} = \text{logit}(p) \sim \text{logistic}(0, 1)$$

The following statements fit the same model by using transformed parameters:

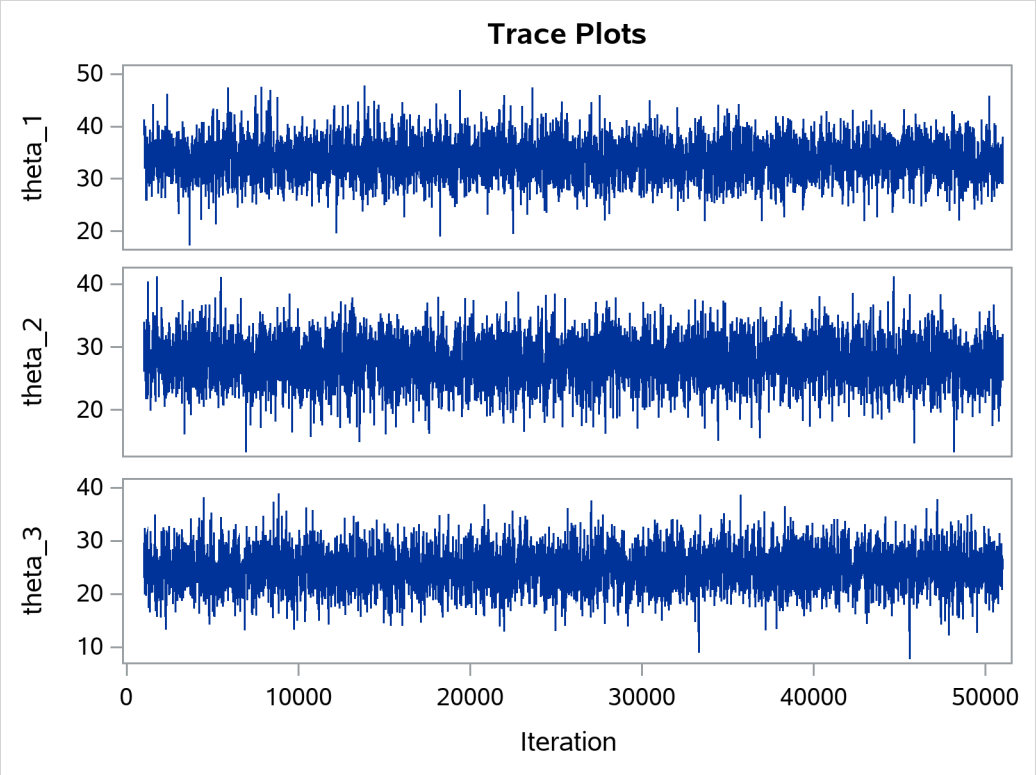
```
proc mcmc data=inputdata nmc=50000 thin=10 outpost=m2 seed=17
          monitor=(p tau mu) plot=trace;
   ods select ess tracepanel;
   parms ltau lgp mu ;

   prior ltau ~ egamma(shape=0.001,iscale=0.001);
   prior lgp ~ logistic(0,1);
   prior mu ~ normal(0,prec=0.00000001);

   beginnodata;
   tau = exp(ltau);
   p = logistic(lgp);
   taub = tau/p;
   tauw = taub-tau;
   endnodata;

   random theta ~ normal(mu, prec=taub) subject=grp monitor=(theta);
   model y ~ normal(theta,prec=tauw);
run;
```

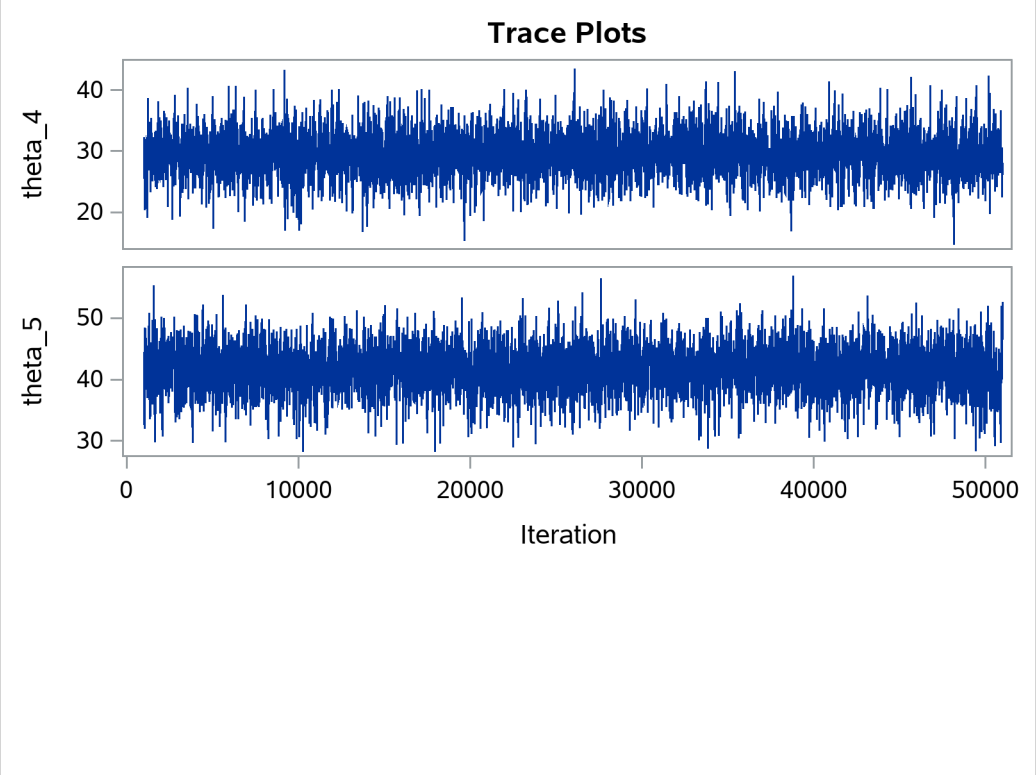
The variable lgp is the logit transformation of p, and ltau is the log transformation of τ. The prior for ltau is egamma, and the prior for lgp is logistic. The TAU and P assignment statements transform the parameters back to their original scales. The rest of the programs remain unchanged. Trace plots (Output 80.20.4) and effective sample size (Output 80.20.5) both show significant improvements in the mixing for both p and τ.

Output 80.20.4 Trace Plots after Transformation



Output 80.20.4 *continued*





Output 80.20.5 Effective Sample Sizes after Transformation

The MCMC Procedure

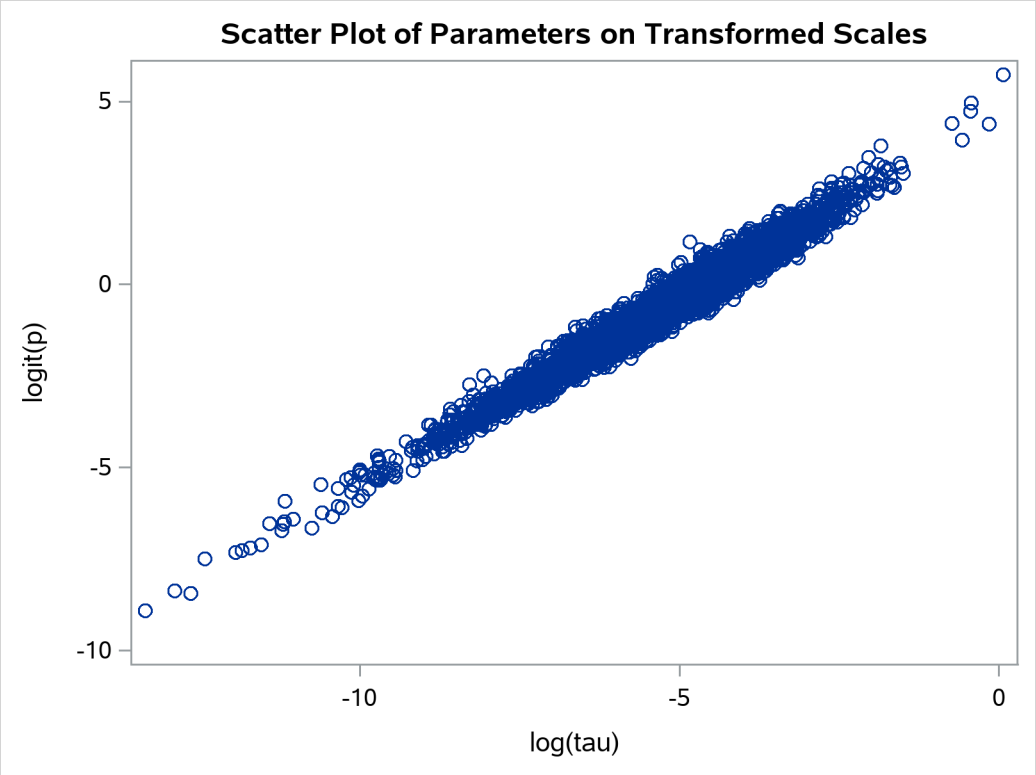
Effective Sample Sizes

Parameter	ESS	Autocorrelation Time	Efficiency
p	3078.7	1.6241	0.6157
tau	2798.9	1.7864	0.5598
mu	5000.0	1.0000	1.0000
theta_1	5127.1	0.9752	1.0254
theta_2	4798.1	1.0421	0.9596
theta_3	4741.2	1.0546	0.9482
theta_4	5196.7	0.9621	1.0393
theta_5	4282.7	1.1675	0.8565

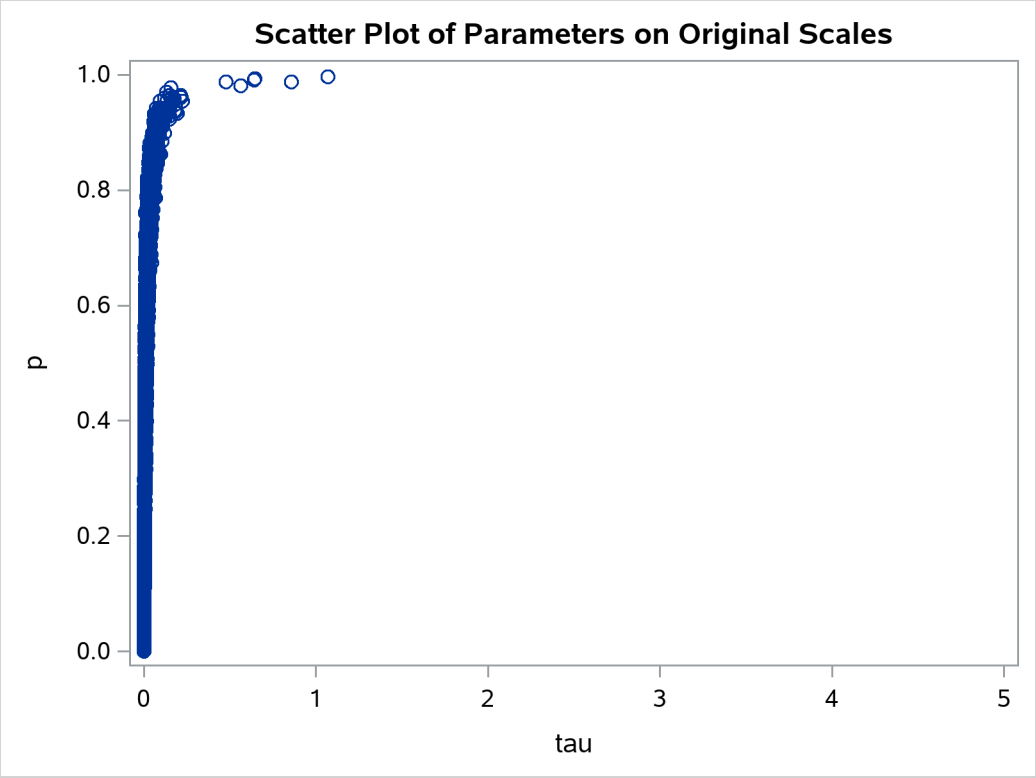
The following statements generate Output 80.20.6 and Output 80.20.7:

```
title 'Scatter Plot of Parameters on Transformed Scales';

proc sgplot data=m2;
   yaxis label = 'logit(p)';
   xaxis label = 'log(tau)';
   scatter x = ltau y = lgp;
run;

title 'Scatter Plot of Parameters on Original Scales';

proc sgplot data=m2;
   yaxis label = 'p';
   xaxis label = 'tau' values=(0 to 5.0 by 1);
   scatter x = tau y = p;
run;
ods graphics off;
```

Output 80.20.6 Scatter Plot of $\log(\tau)$ versus logit(p), After Transformation



Output 80.20.7 Scatter Plot of τ versus p, After Transformation



The scatter plot of $\log(\tau)$ versus $\text{logit}(p)$ shows a linear relationship between the two transformed parameters, and this explains the improvement in mixing. In addition, the transformations also help the Markov chain better explore in the original parameter space. Output 80.20.7 shows a scatter plot of τ versus p. The plot is similar to Output 80.20.3. However, note that tau has a far longer tail in Output 80.20.7, extending all the way to 5 as opposed to 0.15 in Output 80.20.3. This means that the second Markov chain can explore this dimension of the parameter more efficiently, and as a result, you are able to draw more precise inference with an equal number of simulations.

Example 80.21: Gelman-Rubin Diagnostics

(View the complete code for this example (mcmcex21.sas) in the example repository.)

PROC MCMC does not have the Gelman-Rubin test (see the section "Gelman and Rubin Diagnostics" on page 167 in Chapter 8, "Introduction to Bayesian Analysis Procedures") as a part of its diagnostics. The Gelman-Rubin diagnostics rely on parallel chains to test whether they all converge to the same posterior distribution. This example demonstrates how you can carry out this convergence test. The regression model from the section "Simple Linear Regression" on page 6201 is used. The model has three parameters: β_0 and β_1 are the regression coefficients, and σ^2 is the variance of the error distribution.

To run a Gelman-Rubin diagnostic test, you want to start Markov chains at different places in the parameter space. Suppose you want to start β_0 at 10, –15, and 0; β_1 at –5, 10, and 0; and σ^2 at 1, 20, and 50. You can put these starting values in the following Init SAS data set:

```
title 'Simple Linear Regression, Gelman-Rubin Diagnostics';

data init;
   input Chain beta0 beta1 sigma2;
   datalines;
   1   10  -5   1
   2  -15  10  20
   3    0   0  50
;
```

The following statements run PROC MCMC three times, each with starting values specified in the data set Init:

```
/* define constants */
%let nchain = 3;
%let nparm = 3;
%let nsim = 50000;
%let var = beta0 beta1 sigma2;

%macro gmcmc;
   %do i=1 %to &nchain;
      data _null_;
         set init;
         if Chain=&i;
         %do j = 1 %to &nparm;
            call symputx("init&j", %scan(&var, &j));
         %end;
         stop;
```

```
      run;

      proc mcmc data=sashelp.class outpost=out&i init=reinit nbi=0 nmc=&nsim
                stats=none seed=7;
         parms beta0 &init1 beta1 &init2;
         parms sigma2 &init3 / n;
         prior beta0 beta1 ~ normal(0, var = 1e6);
         prior sigma2 ~ igamma(3/10, scale = 10/3);
         mu = beta0 + beta1*height;
         model weight ~ normal(mu, var = sigma2);
      run;
   %end;
%mend;

ods exclude all;
%gmcmc;
ods exclude none;
```

The macro variables nchain, nparm, nsim, and var define the number of chains, the number of parameters, the number of Markov chain simulations, and the parameter names, respectively. The macro GMCMC gets initial values from the data set Init, assigns them to the macro variables init1, init2 and init3, starts the Markov chain at these initial values, and stores the posterior draws to three output data sets: Out1, Out2, and Out3.

In the PROC MCMC statement, the INIT=REINIT option restarts the Markov chain after tuning at the assigned initial values. No burn-in is requested.

You can use the autocall macro GELMAN to calculate the Gelman-Rubin statistics by using the three chains. The GELMAN macro has the following arguments:

```
%macro gelman(dset, nparm, var, nsim, nc=3, alpha=0.05);
```

The argument dset is the name of the data set that stores the posterior samples from all the runs, nparm is the number of parameters, var is the name of the parameters, nsim is the number of simulations, nc is the number of chains with a default value of 3, and alpha is the α significant level in the test with a default value of 0.05. This macro creates two data sets: _Gelman_Ests stores the diagnostic estimates and _Gelman_Parms stores the names of the parameters.

The following statements calculate the Gelman-Rubin diagnostics:

```
data all;
   set out1(in=in1) out2(in=in2) out3(in=in3);
   if in1 then Chain=1;
   if in2 then Chain=2;
   if in3 then Chain=3;
run;

%gelman(all, &nparm, &var, &nsim);

data GelmanRubin(label='Gelman-Rubin Diagnostics');
   merge _Gelman_Parms _Gelman_Ests;
run;

proc print data=GelmanRubin;
run;
```

The Gelman-Rubin statistics are shown in Output 80.21.1.

Output 80.21.1 Gelman-Rubin Diagnostics of the Regression Example

Simple Linear Regression, Gelman-Rubin Diagnostics

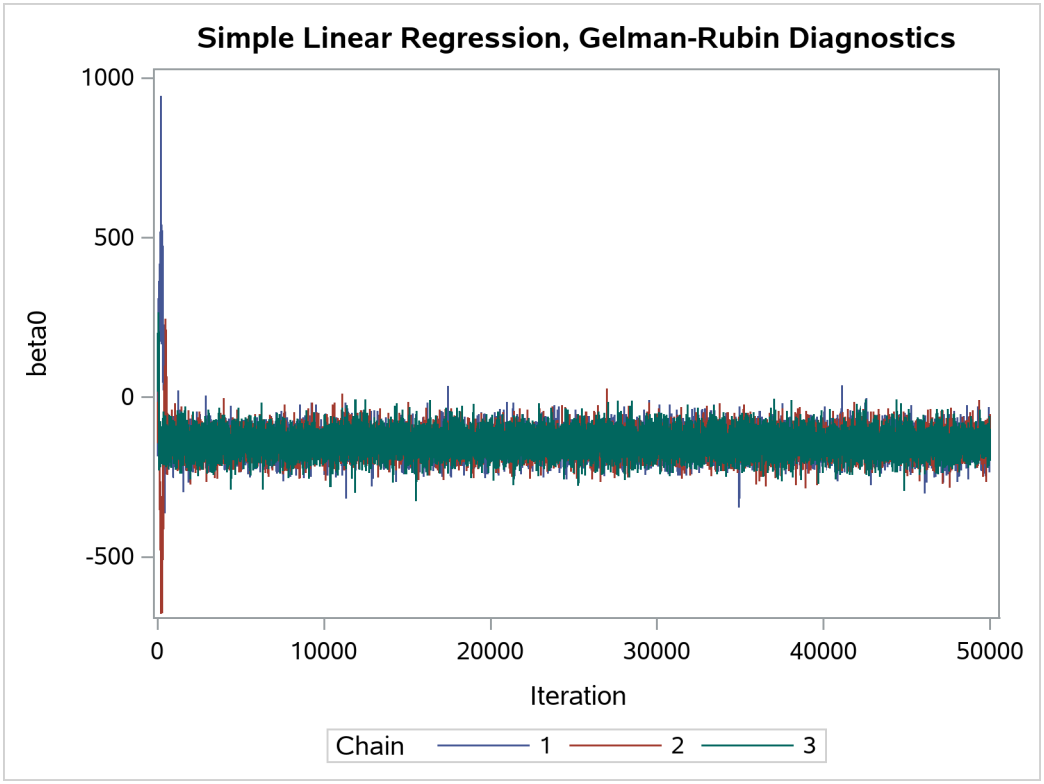
Obs	Parameter	Between-chain	Within-chain	Estimate	UpperBound
1	beta0	5384.76	1168.64	1.0002	1.0001
2	beta1	1.20	0.30	1.0002	1.0002
3	sigma2	8034.41	2890.00	1.0010	1.0011

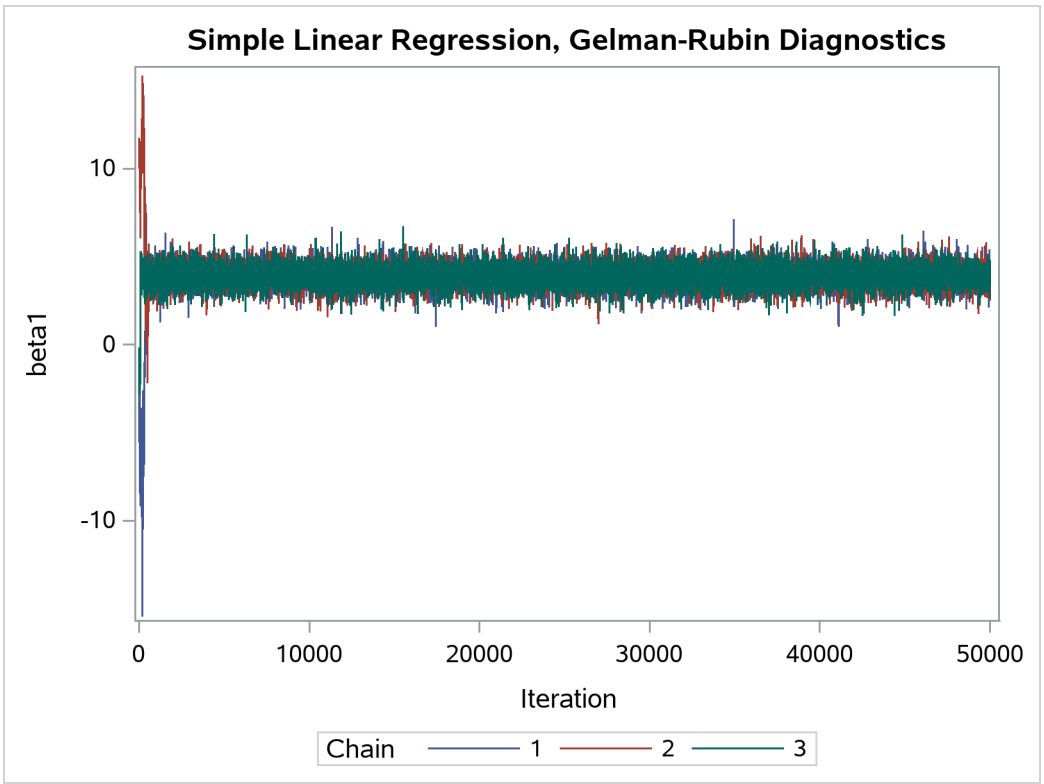
The Gelman-Rubin statistics do not reveal any concerns about the convergence or the mixing of the multiple chains. To get a better visual picture of the multiple chains, you can draw overlapping trace plots of these parameters from the three Markov chains runs.

The following statements create Output 80.21.2:

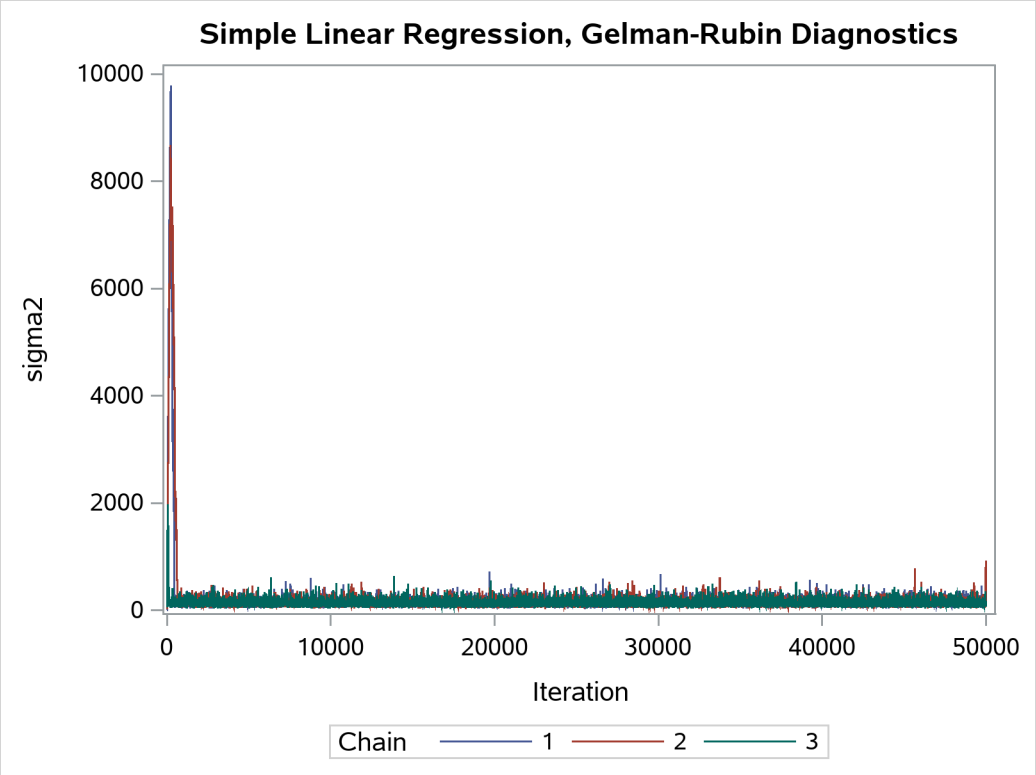
```
/* plot the trace plots of three Markov chains. */
%macro trace;
   %do i = 1 %to &nparm;
      proc sgplot data=all cycleattrs;
         series x=Iteration y=%scan(&var, &i) / group=Chain;
      run;
   %end;
%mend;
%modstyle(name=linestyle, linestyles=Solid)
ods listing style=linestyle;
%trace;
```

Output 80.21.2 Trace Plots of Three Chains for Each of the Parameters





Output 80.21.2 *continued*



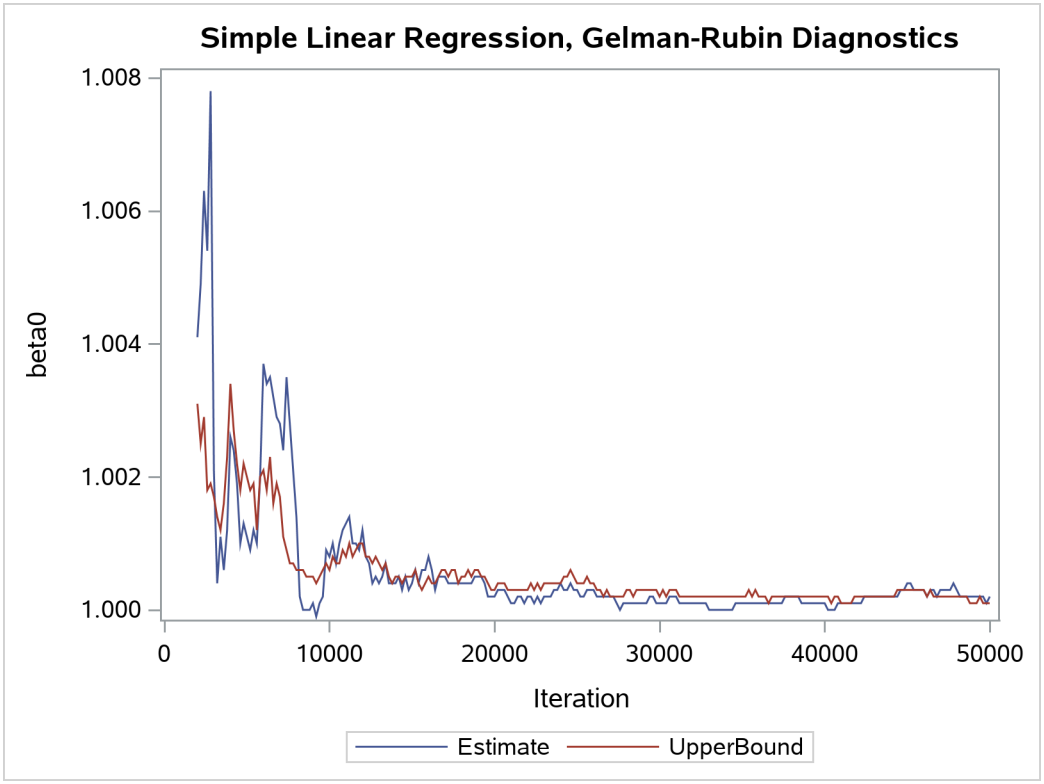
The trace plots show that three chains all eventually converge to the same regions even though they started at very different locations. In addition to the trace plots, you can also plot the potential scale reduction factor (PSRF). See the section "Gelman and Rubin Diagnostics" on page 167 in Chapter 8, "Introduction to Bayesian Analysis Procedures," for definition and details.

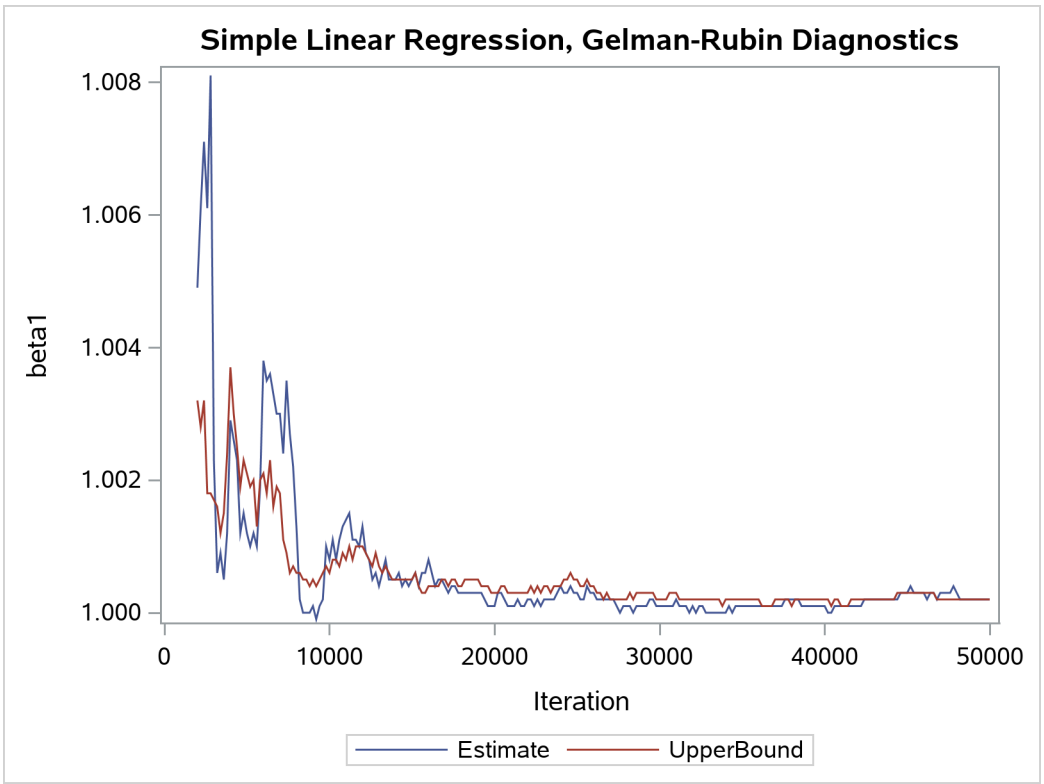
The following statements calculate PSRF for each parameter. They use the GELMAN macro repeatedly and can take a while to run:

```
/* define sliding window size */
%let nwin = 200;
data PSRF;
run;

%macro PSRF(nsim);
   %do k = 1 %to %sysevalf(&nsim/&nwin, floor);
      %gelman(all, &nparm, &var, nsim=%sysevalf(&k*&nwin));
      data GelmanRubin;
         merge _Gelman_Parms _Gelman_Ests;
      run;

      data PSRF;
         set PSRF GelmanRubin;
      run;
   %end;
%mend PSRF;
```

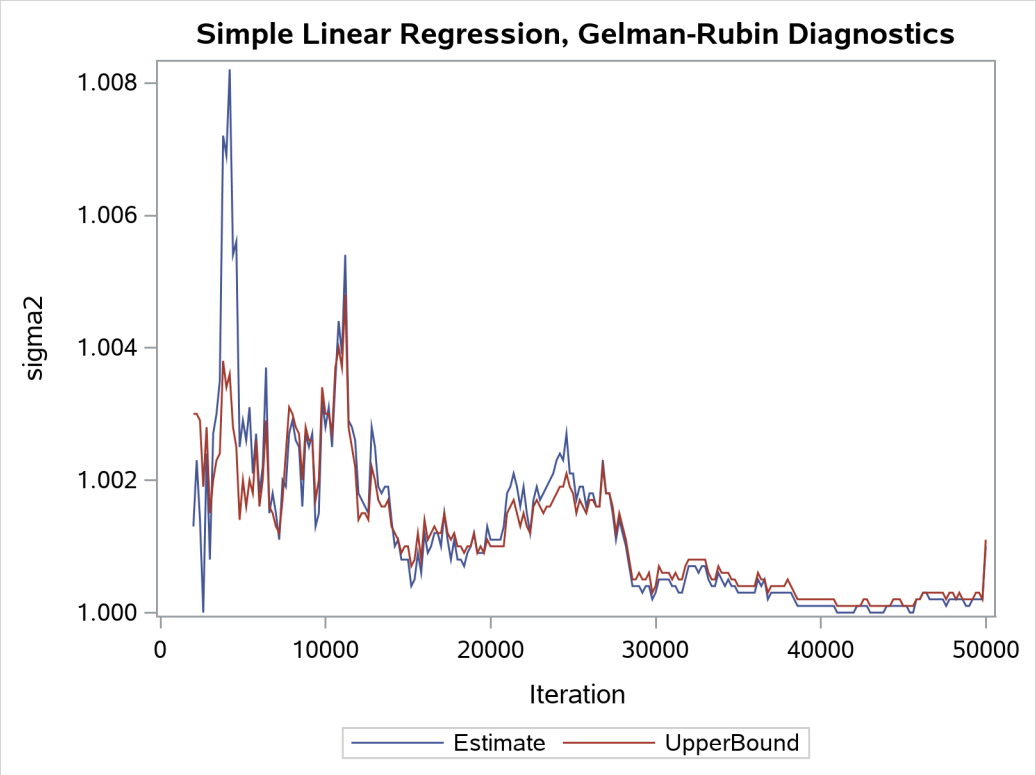
```
options nonotes;
%PSRF(&nsim);
options notes;

data PSRF;
   set PSRF;
   if _n_ = 1 then delete;
run;

proc sort data=PSRF;
   by Parameter;
run;

%macro sepPSRF(nparm=, var=, nsim=);
   %do k = 1 %to &nparm;
      data save&k; set PSRF;
         if _n_ > %sysevalf(&k*&nsim/&nwin, floor) then delete;
         if _n_ < %sysevalf((&k-1)*&nsim/&nwin + 1, floor) then delete;
         Iteration + &nwin;
      run;

      proc sgplot data=save&k(firstobs=10) cycleattrs;
         series x=Iteration y=Estimate;
         series x=Iteration y=upperbound;
         yaxis label="%scan(&var, &k)";
      run;
   %end;
%mend sepPSRF;

%sepPSRF(nparm=&nparm, var=&var, nsim=&nsim);
```

Output 80.21.3 PSRF Plot for Each Parameter





Output 80.21.3 *continued*



PSRF is the square root of the ratio of the between-chain variance and the within-chain variance. A large PSRF indicates that the between-chain variance is substantially greater than the within-chain variance, so that longer simulation is needed. You want the PSRF to converge to 1 eventually, as it appears to be the case in this simulation study.

Example 80.22: One-Compartment Model with Pharmacokinetic Data

(View the complete code for this example (mcmcex22.sas) in the example repository.)

A popular application of nonlinear mixed models is in the field of pharmacokinetics, which studies how a drug disperses through a living individual. This example considers the theophylline data from Pinheiro and Bates (1995). Serum concentrations of the drug theophylline, which is used to treat respiratory diseases, are measured in 12 subjects over a 25-hour period after oral administration. The data are as follows.

```
data theoph;
   input subject time conc dose;
   datalines;
 1  0.00  0.74 4.02
 1  0.25  2.84 4.02
 1  0.57  6.57 4.02
 1  1.12 10.50 4.02
 1  2.02  9.66 4.02
 1  3.82  8.58 4.02
 1  5.10  8.36 4.02
```

```
 1  7.03  7.47 4.02

   ... more lines ...

12 24.15  1.17 5.30
;
```

A commonly used one-compartment model is based on the differential equations

$$\begin{aligned} \frac{dA_0(t)}{dt} &= -K_a A_0(t) \\ \frac{dA(t)}{dt} &= K_a A_0(t) - K_e A(t) \end{aligned}$$

where K_a is the absorption rate and K_e is the elimination rate.

The initial values are

$$\begin{aligned} A_a(t=0) &= x \\ A(t=0) &= 0 \end{aligned}$$

where x is the dosage.

The expected concentration of the substance in the body is computed by dividing the solution to the ODE, $A(t)$, by Cl, the clearance,

$$\begin{aligned} \mu_i(t) &= A_i(t)/Cl \\ y_i(t) &\sim \text{normal}(\mu_i(t), \sigma^2) \end{aligned}$$

where i is the subject index.

Pinheiro and Bates (1995) consider the following first-order compartment model for these data, where $\log(Cl)$ and $\log(K_a)$ are modeled using random effects to account for the patient-to-patient variability:

$$\begin{aligned} Cl_i &= \exp(\beta_1 + b_{i1}) \\ K_{a_i} &= \exp(\beta_2 + b_{i2}) \\ K_{e_i} &= \exp(\beta_3) \end{aligned}$$

Here the βs denote fixed-effects parameters and the b_is denote random-effects parameters with an unknown covariance matrix.

Although there is an analytical solution to this set of differential equations, this example illustrates the use of a general ODE solver that does not require a closed-form solution. To use the ODE solver, you want to first define the set of differential equations in PROC FCMP and store the objective function, called OneComp, in a user library:

```
proc fcmp outlib=sasuser.funcs.PK;
   subroutine OneComp(t,y[*],dy[*],ka,ke);
   outargs dy;
   dy[1] = -ka*y[1];
   dy[2] = ka*y[1]-ke*y[2];
   endsub;
run;
```

The first argument of the OneComp subroutine is the time variable in the differential equation. The second argument is the with-respect-to variable, which can be an array in case a multidimensional problem is required. The third argument is an array that stores the differential equations. This dy argument must also be the OUTARGS variable in the subroutine. The subsequent variables are additional variables, depending on the problem at hand.

In the OneComp subroutine, you can define K_a and K_e as functions of β_1, β_2, and the random-effects parameter b_2. The dy[1] and dy[2] are the differential equation, with dy[1] storing dA_a/dt and dy[2] storing dA_c/dt.

The following PROC MCMC statements use the CALL ODE subroutine to solve the set of differential equations defined in OneComp and then use that solution in the construction of the likelihood function:

```
options cmplib=sasuser.funcs;

proc mcmc data=theoph nmc=10000 seed=27 outpost=theophO diag=none
   nthreads=8;
   ods select PostSumInt;
   array b[2];
   array muB[2] (0 0);
   array cov[2,2];
   array S[2,2] (1 0 0 1);

   array init[2] dose 0;
   array sol[2];

   parms beta1 -3.22 beta2 0.47 beta3 -2.45 ;
   parms cov {0.03 0 0 0.4};
   parms s2y;

   prior beta: ~ normal(0, sd=100);
   prior cov ~ iwish(2, S);
   prior s2y  ~ igamma(shape=3, scale=2);

   random b ~ mvn(muB, cov) subject=subject;
   cl   = exp(beta1 + b1);
   ka   = exp(beta2 + b2);
   ke   = exp(beta3);
   v    = cl/ke;
   call ode('OneComp',sol,init,0,time,ka,ke);
   mu   = (sol[2]/v);
   model conc ~ normal(mu,var=s2y);
run;
```

The INIT array stores the initial values of the two differential equations, with $A_a(t = 0) = \text{dose}$ and $A(t = 0) = 0$. The array is used as an input argument to the CALL ODE subroutine.

The RANDOM statement specifies two-dimensional random effects, b, with a multivariate normal prior. The first random effect, b1, enters the model through the clearance variable, cl. The second random effect, b2, is part of the differential equations. The CALL ODE subroutine solves the OneComp set of differential equations and returns the solution to the SOL array. The second array element, SOL[2], is the solution to $A_i(t)$ for every subject i at every time t.

Posterior summary statistics are reported in Output 80.22.1.

Output 80.22.1 Posterior Summary Statistics

The MCMC Procedure

Posterior Summaries and Intervals

Parameter	N	Mean	Standard Deviation	95% HPD Interval	
beta1	10000	-3.1989	0.0780	-3.3553	-3.0609
beta2	10000	0.4244	0.1915	0.00824	0.7664
beta3	10000	-2.4589	0.0495	-2.5647	-2.3702
cov1	10000	0.1306	0.0634	0.0449	0.2477
cov2	10000	-0.00222	0.0952	-0.1874	0.1888
cov3	10000	-0.00222	0.0952	-0.1874	0.1888
cov4	10000	0.6230	0.3295	0.1778	1.2376
s2y	10000	0.5231	0.0720	0.3888	0.6639

In this problem, the closed-form solution of the ODE is known:

$$C_{it} = \frac{Dk_{e_i}k_{a_i}}{Cl_i(k_{a_i} - k_{e_i})}[\exp(-k_{e_i}t) - \exp(-k_{a_i}t)] + e_{it}$$

You can manually enter the equation in PROC MCMC and use the following program to fit the same model:

```
proc mcmc data=theoph nmc=10000 seed=22 outpost=theophC;
   array b[2];

   array mu[2] (0 0);
   array cov[2,2];
   array S[2,2] (1 0 0 1);

   parms beta1 -3.22 beta2 0.47 beta3 -2.45 ;
   parms cov {0.03 0 0 0.4};
   parms s2y;

   prior beta: ~ normal(0, sd=100);
   prior cov ~ iwish(2, S);
   prior s2y  ~ igamma(shape=3, scale=2);

   random b ~ mvn(mu, cov) subject=subject;
   cl   = exp(beta1 + b1);
   ka   = exp(beta2 + b2);
   ke   = exp(beta3);
```

```
    pred = dose*ke*ka*(exp(-ke*time)-exp(-ka*time))/cl/(ka-ke);
    model conc ~ normal(pred,var=s2y);
run;
```

Because this program makes it unnecessary to numerically solve the ODE at every observation and every iteration, it runs much faster than the program that uses the CALL ODE subroutine. But few pharmacokinetic models have known solutions that enable you to do that.

References

Abuhelwa, A. Y., Foster, D. J., and Upton, R. N. (2015). "ADVAN-Style Analytical Solutions for Common Pharmacokinetic Models." *Journal of Pharmacological and Toxicological Methods* 73:42–48.

Aitkin, M., Anderson, D. A., Francis, B., and Hinde, J. (1989). *Statistical Modelling in GLIM*. Oxford: Oxford Science Publications.

Atkinson, A. C. (1979). "The Computer Generation of Poisson Random Variables." *Journal of the Royal Statistical Society, Series C* 28:29–35.

Atkinson, A. C., and Whittaker, J. (1976). "A Switching Algorithm for the Generation of Beta Random Variables with at Least One Parameter Less Than One." *Journal of the Royal Statistical Society, Series A* 139:462–467.

Bacon, D. W., and Watts, D. G. (1971). "Estimating the Transition between Two Intersecting Straight Lines." *Biometrika* 58:525–534.

Banerjee, S., Carlin, B. P., and Gelfand, A. E. (2004). *Hierarchical Modeling and Analysis for Spatial Data*. Boca Raton, FL: Chapman & Hall/CRC.

Beal, S. L., Sheiner, L. B., Boeckmann, A. J., and Bauer, R. J., eds. (2011). *NONMEM User's Guides (1989–2011)*. Hanover, MD: Icon Development Solutions.

Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis*. 2nd ed. New York: Springer-Verlag.

Box, G. E. P., and Cox, D. R. (1964). "An Analysis of Transformations." *Journal of the Royal Statistical Society, Series B* 26:211–234.

Byrne, G. D., and Hindmarsh, A. C. (1975). "A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations." *ACM Transactions on Mathematical Software* 1:71–96. http://portal.acm.org/citation.cfm?doid=355626.355636.

Carlin, B. P., Gelfand, A. E., and Smith, A. F. M. (1992). "Hierarchical Bayesian Analysis of Changepoint Problems." *Journal of the Royal Statistical Society, Series C* 41:389–405.

Chaloner, K. (1994). "Residual Analysis and Outliers in Bayesian Hierarchical Models." In *Aspects of Uncertainty: A Tribute to D. V. Lindley*, 149–157. New York: John Wiley & Sons.

Chaloner, K., and Brant, R. (1988). "A Bayesian Approach to Outlier Detection and Residual Analysis." *Biometrika* 75:651–659.

Cheng, R. C. H. (1978). "Generating Beta Variates with Non-integral Shape Parameters." *Communications ACM* 28:290–295.

Clayton, D. G. (1991). "A Monte Carlo Method for Bayesian Inference in Frailty Models." *Biometrics* 47:467–485.

Congdon, P. (2003). *Applied Bayesian Modeling*. New York: John Wiley & Sons.

Crowder, M. J. (1978). "Beta-Binomial Anova for Proportions." *Journal of the Royal Statistical Society, Series C* 27:34–37.

Draper, D. (1996). "Discussion of the Paper by Lee and Nelder." *Journal of the Royal Statistical Society, Series B* 58:662–663.

Eilers, P. H. C., and Marx, B. D. (1996). "Flexible Smoothing with *B*-Splines and Penalties." *Statistical Science* 11:89–121. With discussion.

Finney, D. J. (1947). "The Estimation from Individual Records of the Relationship between Dose and Quantal Response." *Biometrika* 34:320–334.

Fisher, D., and Shafer, S. (2007). "Pharmacokinetic and Pharmacodynamic Analysis with NONMEM: Basic Concepts." Course materials for Fisher/Shafer NONMEM Workshop, March 7–11, Ghent, Belgium.

Fisher, R. A. (1935). "The Fiducial Argument in Statistical Inference." *Annals of Eugenics* 6:391–398.

Fishman, G. S. (1996). *Monte Carlo: Concepts, Algorithms, and Applications*. New York: John Wiley & Sons.

Gabrielsson, J., and Weiner, D. (2006). *Pharmacokinetic and Pharmacodynamic Data Analysis: Concepts and Applications*. 4th ed. Stockholm: Swedish Pharmaceutical Press.

Gaver, D. P., and O'Muircheartaigh, I. G. (1987). "Robust Empirical Bayes Analysis of Event Rates." *Technometrics* 29:1–15.

Gelfand, A. E., Hills, S. E., Racine-Poon, A., and Smith, A. F. M. (1990). "Illustration of Bayesian Inference in Normal Data Models Using Gibbs Sampling." *Journal of the American Statistical Association* 85:972–985.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis*. 2nd ed. London: Chapman & Hall.

Gentleman, R., and Geyer, C. J. (1994). "Maximum Likelihood for Interval Censored Data: Consistency and Computation." *Biometrika* 81:618–623.

Gilks, W. R. (2003). "Adaptive Metropolis Rejection Sampling (ARMS)." Software from MRC Biostatistics Unit, Cambridge, UK. http://www.maths.leeds.ac.uk/~wally.gilks/adaptive.rejection/web_page/Welcome.html.

Gilks, W. R., and Wild, P. (1992). "Adaptive Rejection Sampling for Gibbs Sampling." *Journal of the Royal Statistical Society, Series C* 41:337–348.

Holmes, C. C., and Held, L. (2006). "Bayesian Auxiliary Variable Models for Binary and Multinomial Regression." *Bayesian Analysis* 1:145–168.

Ibrahim, J. G., Chen, M.-H., and Lipsitz, S. R. (2001). "Missing Responses in Generalised Linear Mixed Models When the Missing Data Mechanism Is Nonignorable." *Biometrika* 88:551–564.

Kass, R. E., Carlin, B. P., Gelman, A., and Neal, R. M. (1998). "Markov Chain Monte Carlo in Practice: A Roundtable Discussion." *American Statistician* 52:93–100.

Krall, J. M., Uthoff, V. A., and Harley, J. B. (1975). "A Step-Up Procedure for Selecting Variables Associated with Survival." *Biometrics* 31:49–57.

Kuhfeld, W. F. (2010). *Conjoint Analysis*. Technical report, SAS Institute Inc., Cary, NC. http://support.sas.com/resources/papers/tnote/tnote_marketresearch.html.

Kurada, R. R., and Chen, F. (2018). "Fitting Compartment Models Using PROC NLMIXED." In *Proceedings of the SAS Global Forum 2018 Conference*. Cary, NC: SAS Institute Inc. https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1883-2018.pdf.

Lin, D. Y. (1994). "Cox Regression Analysis of Multivariate Failure Time Data: The Marginal Approach." *Statistics in Medicine* 13:2233–2247.

Little, R. J. A., and Rubin, D. B. (2002). *Statistical Analysis with Missing Data*. 2nd ed. Hoboken, NJ: John Wiley & Sons.

Matsumoto, M., and Kurita, Y. (1992). "Twisted GFSR Generators." *ACM Transactions on Modeling and Computer Simulation* 2:179–194.

Matsumoto, M., and Kurita, Y. (1994). "Twisted GFSR Generators II." *ACM Transactions on Modeling and Computer Simulation* 4:254–266.

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.

McGrath, E. J., and Irving, D. C. (1973). *Techniques for Efficient Monte Carlo Simulation, Vol. 2: Random Number Generation for Selected Probability Distributions*. Technical report, Science Applications Inc., La Jolla, CA.

Michael, J. R., Schucany, W. R., and Haas, R. W. (1976). "Generating Random Variates Using Transformations with Multiple Roots." *American Statistician* 30:88–90.

Owen, J. S., and Fiedler-Kelly, J. (2014). *Introduction to Population Pharmacokinetic/Pharmacodynamic Analysis with Nonlinear Mixed Effects Models*. Hoboken, NJ: John Wiley & Sons.

Pinheiro, J. C., and Bates, D. M. (1995). "Approximations to the Log-Likelihood Function in the Nonlinear Mixed-Effects Model." *Journal of Computational and Graphical Statistics* 4:12–35.

Pregibon, D. (1981). "Logistic Regression Diagnostics." *Annals of Statistics* 9:705–724.

Ralston, A., and Rabinowitz, P. (1978). *A First Course in Numerical Analysis*. New York: McGraw-Hill.

Rice, S. O. (1973). "Efficient Evaluation of Integrals of Analytic Functions by the Trapezoidal Rule." *Bell System Technical Journal* 52:707–722.

Ripley, B. D. (1987). *Stochastic Simulation*. New York: John Wiley & Sons.

Robert, C. P. (1995). "Simulation of Truncated Normal Variables." *Statistics and Computing* 5:121–125.

Roberts, G. O., Gelman, A., and Gilks, W. R. (1997). "Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms." *Annals of Applied Probability* 7:110–120.

Roberts, G. O., and Rosenthal, J. S. (2001). "Optimal Scaling for Various Metropolis-Hastings Algorithms." *Statistical Science* 16:351–367.

Rubin, D. B. (1976). "Inference and Missing Data." *Biometrika* 63:581–592.

Rubin, D. B. (1981). "Estimation in Parallel Randomized Experiments." *Journal of Educational Statistics* 6:377–411.

Schervish, M. J. (1995). *Theory of Statistics*. New York: Springer-Verlag.

Sharples, L. (1990). "Identification and Accommodation of Outliers in General Hierarchical Models." *Biometrika* 77:445–453.

Sikorsky, K. (1982). "Optimal Quadrature Algorithms in H_P Spaces." *Numerische Mathematik* 39:405–410.

Sikorsky, K., and Stenger, F. (1984). "Optimal Quadratures in H_P Spaces." *ACM Transactions on Mathematical Software* 3:140–151.

Spiegelhalter, D. J., Thomas, A., Best, N. G., and Gilks, W. R. (1996a). "BUGS Examples, Volume 1." Version 0.5 (version ii).

Spiegelhalter, D. J., Thomas, A., Best, N. G., and Gilks, W. R. (1996b). "BUGS Examples, Volume 2." Version 0.5 (version ii).

Squire, W. (1987). "Comparison of Gauss-Hermite and Midpoint Quadrature with Application to the Voigt Function." In *Numerical Integration: Recent Developments*, edited by P. Keast and G. Fairweather, 111–112. Dordrecht, Netherlands: D. Reidel Publishing.

Stenger, F. (1973a). "Integration Formulas Based on the Trapezoidal Formula." *Journal of the Institute of Mathematics and Its Applications* 12:103–114.

Stenger, F. (1973b). "Remarks on Integration Formulas Based on the Trapezoidal Formula." *Journal of the Institute of Mathematics and Its Applications* 19:145–147.

Thomas, A., Best, N. G., Lunn, D., Arnold, R., and Spiegelhalter, D. (2004). "GeoBUGS User Manual." http://www.mrc-bsu.cam.ac.uk/wp-content/uploads/geobugs12manual.pdf.