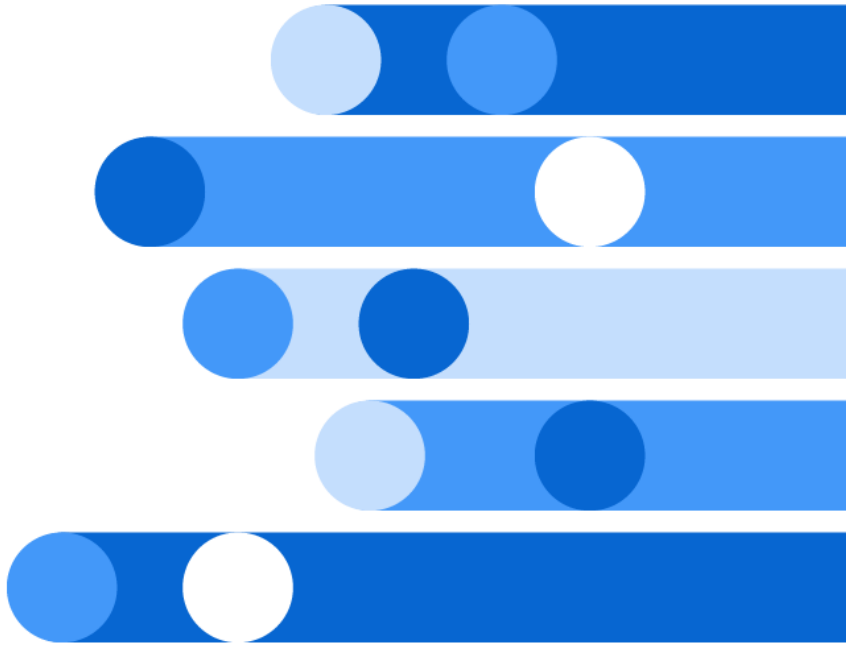
SAS® Econometrics
Econometrics Procedures
HMM Procedure

2024.07*



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

SAS® Documentation
July 18, 2024

This document is an individual chapter from *SAS® Econometrics: Econometrics Procedures*.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2024. *SAS® Econometrics: Econometrics Procedures*. Cary, NC: SAS Institute Inc.

SAS® Econometrics: Econometrics Procedures

July 2024

Chapter 20
HMM Procedure

Contents

Overview: HMM Procedure

The HMM procedure supports hidden Markov models (HMMs), which have been widely applied in economics, finance, science, and engineering. The hidden Markov models have many well-known aliases, such as the general state space model (GSSM), regime-switching model (RSM), Markov-switching model (MSM), and Markov regime-switching model (MRSM). PROC HMM supports the finite hidden Markov model (finite HMM), the Poisson hidden Markov model (Poisson HMM), the Gaussian hidden Markov model (Gaussian HMM), the Gaussian mixture hidden Markov model (GM HMM), the regime-switching regression model (RS REG, also known as the regression hidden Markov model or REG HMM), and the regime-switching autoregression model (RS AR, also known as the autoregressive hidden Markov model or AR HMM) in both the standard and mean-adjusted forms.

The HMM procedure supports the following:

- cross-sectional time series data
- the maximum likelihood (ML) method and the maximum a posteriori (MAP) method for parameter estimation, which is a nonlinear optimization problem
- the prior distribution for parameters when you use the MAP method

- four types of optimization algorithms: the expectation-maximization (EM) algorithm, the active set algorithm, the interior point algorithm, and the stochastic gradient descent (SGD) algorithm
- multistart mode for the active set and interior point optimization algorithms; in this mode, the local solver approaches the problem from multiple starting points, possibly finding a better local optimum as a result. However, the computing cost in multistart mode can be huge.
- specification of initial values for optimization
- state-independent constraints on parameters
- output of parameter estimates to a data table, which can be read back later in order to use those parameter estimates as the initial values for a new estimation
- output of the covariance matrix of parameter estimates
- specification of the order in which to label the states (and components in the case of GM HMMs) in order to solve the label switching problem, and output of the changes of state labels (and component labels in the case of GM HMMs)
- estimation of several models that have different numbers of states or different orders of autoregressive process (or both) in order to help you with the important and difficult task of model selection
- output of the filtering, smoothing, and decoding results
- output of the mean, standard error, covariance matrix, median, and confidence interval of forecasts of dependent variables
- output of the probabilities of states in the forecasts
- multistep forecasts after each observation
- output of the log likelihood up to any observation
- output of the following information criteria: Akaike's information criterion (AIC), the corrected AIC (AICC), the Bayesian information criterion (BIC), and the Hannan-Quinn criterion (HQC)
- saving the model information and using it later to score other new data
- the analytic store technology. You can even apply HMMs to new data on a platform that does not support the HMM procedure.

PROC HMM requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

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

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

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

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

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

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

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 54 in Chapter 4, “Shared Concepts.”

Getting Started: HMM Procedure

This section provides several examples of the types of models that the HMM procedure supports.

Gaussian Hidden Markov Model

Let $\mathbf{Y}_t = (y_{1t}, \ldots, y_{pt})', t = 1, \ldots, T$, denote a p-dimensional time series vector of random variables. Conditional on a latent variable $\mathbf{S}_t$, $\mathbf{Y}_t$ follows a normal distribution,

$$\mathbf{Y}_t|\mathbf{S}_t \sim N(\mu_{\mathbf{S}_t}, \Sigma_{\mathbf{S}_t})$$

where $\mu_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the variable $\mathbf{S}_t$. The variable $\mathbf{S}_t$, the so-called state, follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t|\mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \ldots, \mathbf{S}_1) = p(\mathbf{S}_t|\mathbf{S}_{t-1})$$

where $p(.|.)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j\,|\mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π, of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \ldots, K\}$$

Because the variable $\mathbf{S}_t$ is unobservable and follows a Markov chain and because $\mathbf{Y}_t$ follows a Gaussian distribution conditional on $\mathbf{S}_t$, the model is called the Gaussian hidden Markov model (Gaussian HMM). The Gaussian HMM is sometimes described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{\mu_i, \Sigma_i, i = 1, \ldots, K\}$, consisting of parameters that define the state-dependent distribution of observable variables.

Consider a bivariate Gaussian HMM that has the following parameter values:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.001 & 0.999 \\ 0.999 & 0.001 \end{pmatrix}$$

$$\mu_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The following statements simulate the bivariate vector time series from the previous model to provide test data for the HMM procedure:

```
%let pi1 = 0.5;
%let a11 = 0.001;
%let a22 = 0.001;
%let mu1_1 = 1;
%let mu1_2 = 1;
%let sigma1_11 = 1;
%let sigma1_21 = 0;
%let sigma1_22 = 1;
%let mu2_1 = -1;
%let mu2_2 = -1;
%let sigma2_11 = 1;
%let sigma2_21 = 0;
%let sigma2_22 = 1;
%let T = 10000;
%let seed = 1234;

data DGPone;
   retain cd1_11 cd1_21 cd1_22 cd2_11 cd2_21 cd2_22;
   do t = 1 to &T.;
      if(t=1) then do;
         /* initial probability distribution */
         p = &pi1.;
         /* Cholesky decomposition of sigma1 */
         cd1_11 = sqrt(&sigma1_11.);
         cd1_21 = &sigma1_21./sqrt(&sigma1_11.);
         cd1_22 = sqrt(&sigma1_22.-&sigma1_21.*&sigma1_21./&sigma1_11.);
         /* Cholesky decomposition of sigma2 */
         cd2_11 = sqrt(&sigma2_11.);
         cd2_21 = &sigma2_21./sqrt(&sigma2_11.);
         cd2_22 = sqrt(&sigma2_22.-&sigma2_21.*&sigma2_21./&sigma2_11.);
      end;
      else do;
         /* transition probability matrix */
         if(lags=1) then p = &a11.;
         else p = 1-&a22.;
      end;
      u = uniform(&seed.);
      if(u<=p) then s=1;
      else s = 2;
      e1 = normal(&seed.); e2 = normal(&seed.);
      if(s=1) then do;
         /* (x,y) ~ N(mu1, Sigma1) at state 1 */
         x = &mu1_1. + cd1_11*e1;
         y = &mu1_2. + cd1_21*e1+cd1_22*e2;
      end;
      else do;
         /* (x,y) ~ N(mu2, Sigma2) at state 2 */
         x = &mu2_1. + cd2_11*e1;
         y = &mu2_2. + cd2_21*e1+cd2_22*e2;
      end;
      output;
      lags = s;
   end;
```

```
run;

data One;
   set DGPone;
   keep t x y;
run;
```

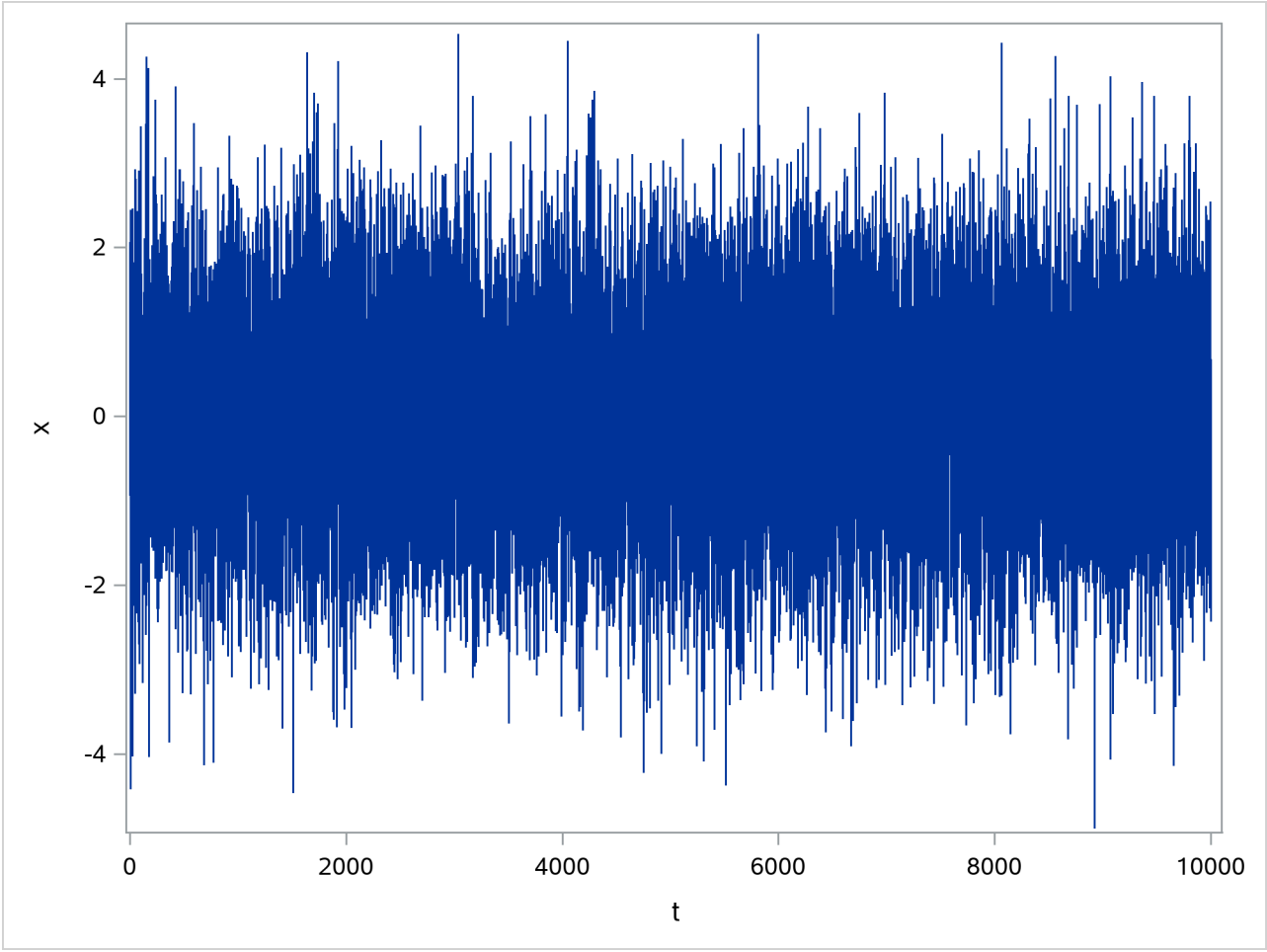
In general, the data generating process is unknown. What can be seen is the data table One, but not the data table DGPone.

The following statements plot the simulated series x:

```
proc sgplot data=One;
   series x=t y=x;
run;
```

Figure 20.1 shows the plot of x, whose points seem to fall randomly around 0.

Figure 20.1 Plot of Time Series x

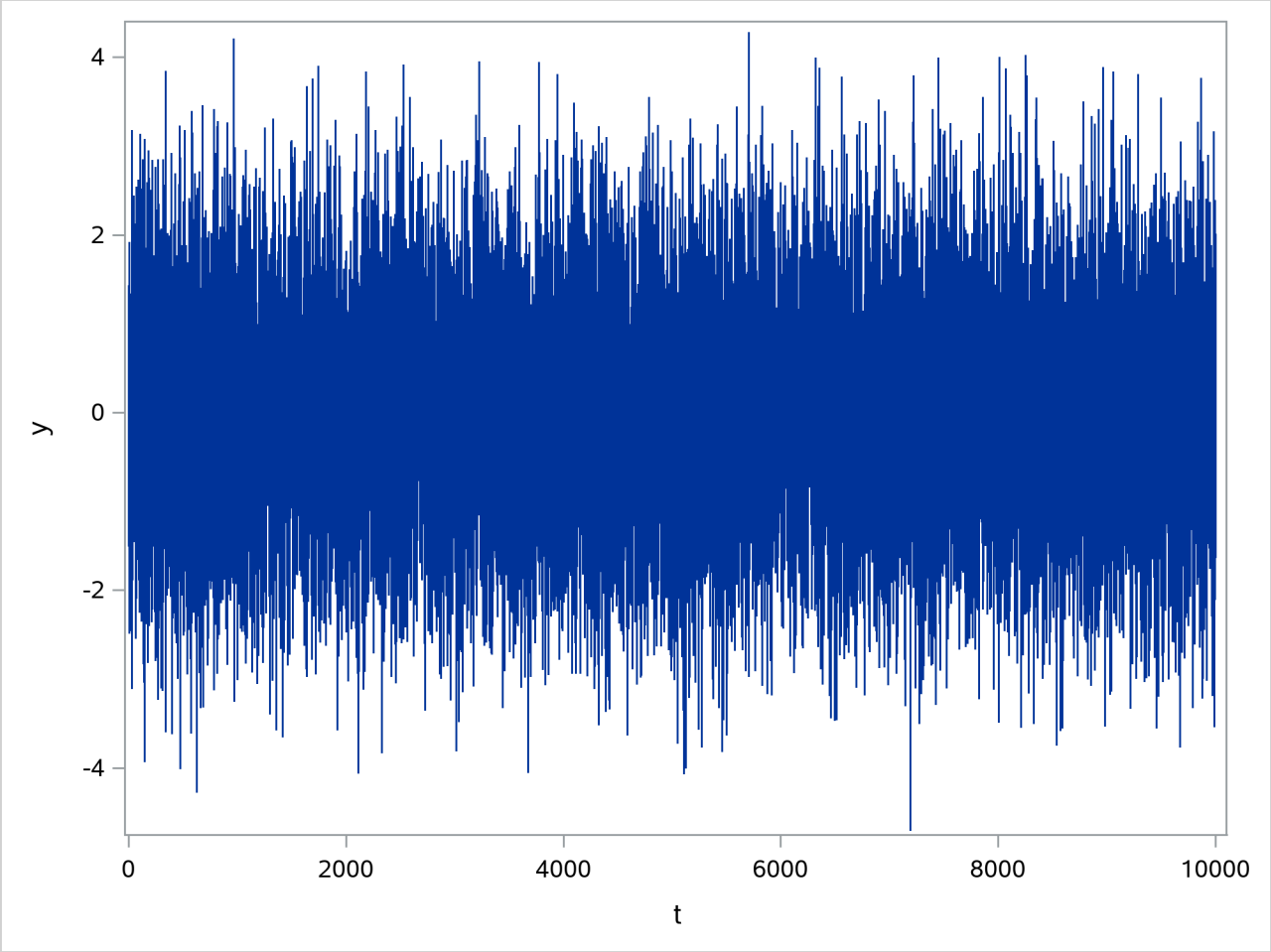


The following statements plot the simulated series y:

```
proc sgplot data=One;
   series x=t y=y;
run;
```

Figure 20.2 shows the plot of y, whose points also seem to fall randomly around 0.

Figure 20.2 Plot of Time Series y

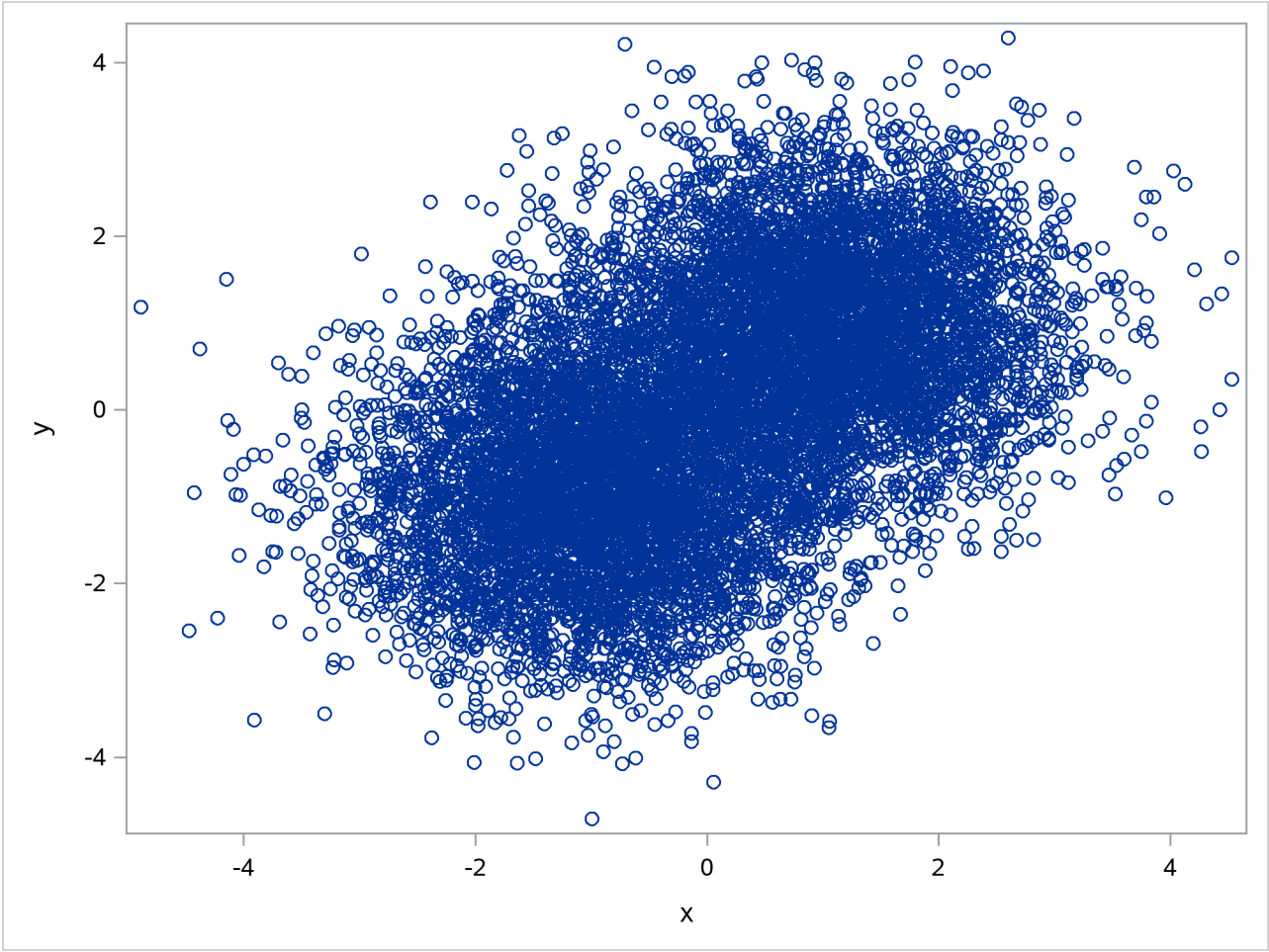


Ignoring the sequential information, the following statements plot the observed points (x, y):

```
proc sgplot data=One;
   scatter y=y x=x;
run;
```

As shown in Figure 20.3, it seems that the points come from two different cluster centers, and many of them lie between two cluster centers.

Figure 20.3 Unclassified Observations



It is very difficult for any cluster analysis methods that cannot use the sequential information to find the correct states for the points that lie between two cluster centers. The simplest cluster analysis method to classify the points, because of the symmetry shown in Figure 20.3, is to draw a line $y = -x$, classify all points above the line to one state, and classify all points below the line to another state.[1] The following statements implement this simplest cluster analysis method and calculate the accuracy, based on the data table DGPone:

```
/* classify the data points by the line y = -x */
data Cluster;
   set One;
   if(y>-x) then state = 1;
   else state = 2;
   keep t state;
run;

data clusterCheck;
   merge DGPone(in=a) Cluster(in=b);
```

[1] In theory, no cluster analysis method, without using the sequential information, could do a better job on this data table than this simplest cluster analysis method.

```
   by t;
   if(s=state) then correct = 1;
   else correct = 0;
   if(a=b);
   keep correct;
run;

data clusterAccuracy;
   set clusterCheck;
   retain count 0 correctCount 0;
   count = count + 1;
   correctCount = correctCount + correct;
   if(count>&T.-0.5) then do;
      accuracy = correctCount / count;
      if(accuracy<0.5) then accuracy = 1 - accuracy;
      output;
   end;
   keep accuracy;
run;

proc print data = clusterAccuracy noobs; run;
```

The accuracy of the cluster analysis, which is shown in Figure 20.4, is 92.02%. This implies that the cluster analysis cannot choose the correct states for about 800 data points!

Figure 20.4 Accuracy of Cluster Analysis

accuracy
0.9202

What is the big difference that you get by using the sequential information through HMMs? Before estimating any model, make a copy of the data in the library:

```
data mylib.One; set One; run;
```

The following statements estimate the two-state Gaussian HMM and then classify the data points through decoding; here decoding means to find the best possible state path for the observed data. The SORT=DESC(MU) option requests that the states be labeled in descending order of the mean parameter values; for more information about the label switching problem, see the section "Label Switching Problem" on page 1283.

```
proc hmm data=mylib.One labelSwitch=(sort=desc(mu));
   id time=t;
   model x y / type=gaussian nstate=2;
   optimize printLevel=3 printIterFreq=1;
   decode out=mylib.oneDecode;
run;
```

The number of observations and model information are shown in Figure 20.5.

Figure 20.5 Number of Observations and Model Information

Observation Information	
Number of Observations	10000
Number of Missing Observations	0

Model Information	
Type of Model	Gaussian HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	2

The default method, which is the maximum likelihood method, is applied; this is a nonlinear optimization problem. The initial parameter values and the objective function value at the start of the optimization are shown in Figure 20.6.

Figure 20.6 Initial Parameter Values and Objective Function Value

Optimization Start	
Parameter	**Estimate**
TPM1_1	0.500000
TPM1_2	0.500000
TPM2_1	0.500000
TPM2_2	0.500000
MU1_1	0.642059
MU1_2	0.513308
MU2_1	-1.609545
MU2_2	-1.329369
SIGMA1_1_1	1.152038
SIGMA1_2_1	0.320856
SIGMA1_2_2	1.478814
SIGMA2_1_1	0.525065
SIGMA2_2_1	-0.243910
SIGMA2_2_2	0.892018

Initial Value of Objective Function	-34510.54106

The details of the optimization algorithm are shown in Figure 20.7.

Figure 20.7 Optimization Algorithm Details

Algorithm Settings	
Maximum Number of Iterations	128
Tolerance for Optimality Error	1E-6
Tolerance for Infeasibility	1E-6
Maximum Allowed Time (seconds)	1.797693E308
Maximum Magnitude of Objective Function Value	1E20
Random Seed	1
Solution Type	1
Multi-Start Globalization Scheme	0
Multi-Start Globalization Bound Range	2

The iterations of the optimization process are shown in Figure 20.8.

Figure 20.8 Iterations of the Optimization Process

Iteration History						
Iteration	**Objective Function**	**Optimality Error**	**Infeasibility**	**Function Calls**	**Gradient Calls**	**Hessian Calls**
1	-34510.54106	610.24392	0.00000	2	1	0
2	-29343.35393	350.54614	0.00000	11	10	1
3	-28606.45004	78.05203	0.00000	19	18	2
4	-28467.87680	1.42097	0.00000	28	27	3
5	-28460.41282	0.34998	0.00000	37	36	4
6	-28459.08728	0.10030	0.00000	46	45	5
7	-28458.80039	0.02968	0.00000	55	54	6
8	-28458.73576	0.00812	0.00000	64	63	7
9	-28458.72552	0.00167	0.00000	73	72	8
10	-28458.72484	0.00018	0.00000	82	81	9
11	-28458.72483	0.00000	0.00000	91	90	10

The final parameter values and the objective function value at the end of the optimization are shown in Figure 20.9.

Figure 20.9 Final Parameter Values and Objective Function Value

Optimization Results	
Parameter	**Estimate**
TPM1_1	0.002071
TPM1_2	0.997929
TPM2_1	0.999863
TPM2_2	0.000137
MU1_1	1.006466
MU1_2	0.994184
MU2_1	-0.993036
MU2_2	-1.007869
SIGMA1_1_1	1.011152
SIGMA1_2_1	0.006387
SIGMA1_2_2	0.988329
SIGMA2_1_1	0.992333
SIGMA2_2_1	-0.005378
SIGMA2_2_2	1.008722

Final Value of Objective Function	-28458.72483

The estimates of the initial state probability vector (ISPV) are shown in Figure 20.10.

Figure 20.10 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	**Estimation**
1	0.50048
2	0.49952

The estimates of the transition probability matrix (TPM) are shown in Figure 20.11.

Figure 20.11 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	**1**	**2**
1	0.00207	0.99793
2	0.99986	0.00014

The estimates of the mean vector of the Gaussian distribution for each state are shown in Figure 20.12.

Figure 20.12 Estimate of Mean Vector for Each State

	Mu	
State	**x**	**y**
1	1.00647	0.99418
2	-0.99304	-1.00787

The estimates of the covariance matrix of the Gaussian distribution for each state are shown in Figure 20.13.

Figure 20.13 Estimate of Covariance Matrix for Each State

Sigma			
State	Variable	x	y
1	x	1.01115	0.00639
	y	0.00639	0.98833
2	x	0.99233	-0.00538
	y	-0.00538	1.00872

The estimates of all parameters are shown in Figure 20.14, which displays columns for the parameter name, estimate value, standard error, *t*-value, and *p*-value.

Figure 20.14 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.002071	0.000711	2.91	0.0036
TPM1_2	0.997929	0.000711	1403.15	<.0001
TPM2_1	0.999863	0.000342	2926.28	<.0001
TPM2_2	0.000137	0.000342	0.40	0.6888
MU1_1	1.006466	0.014228	70.74	<.0001
MU1_2	0.994184	0.014054	70.74	<.0001
MU2_1	-0.993036	0.014096	-70.45	<.0001
MU2_2	-1.007869	0.014217	-70.89	<.0001
SIGMA1_1_1	1.011152	0.020252	49.93	<.0001
SIGMA1_2_1	0.006387	0.014139	0.45	0.6514
SIGMA1_2_2	0.988329	0.019757	50.02	<.0001
SIGMA2_1_1	0.992333	0.019859	49.97	<.0001
SIGMA2_2_1	-0.005378	0.014159	-0.38	0.7041
SIGMA2_2_2	1.008722	0.020194	49.95	<.0001

The log likelihood and information criteria are shown in Figure 20.15.

Figure 20.15 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-28458.72483
AIC	56941.449663
AICC	56941.480904
BIC	57027.973748
HQC	56970.737507

The accuracy of classification through the Gaussian HMM is calculated by the following statements:

```
data gHmmCheck;
   merge DGPone(in=a) mylib.oneDecode(in=b);
   by t;
   if(s=state) then do; correct=1; ds = s; end;
   else do; correct=0; ds = -s; end;
   if(a=b);
   keep t x y s correct ds;
run;

%let qFlipState = -1;
data gHmmAccuracy;
   set gHmmCheck;
   retain count 0 correctCount 0;
   count = count + 1;
   correctCount = correctCount + correct;
   if(count>&T.-0.5) then do;
      accuracy = correctCount / count;
      if(accuracy<0.5) then call symputx("qFlipState",1,'G');
      else call symputx("qFlipState",0,'G');
      if(accuracy<0.5) then accuracy = 1 - accuracy;
      output;
   end;
   keep accuracy;
run;

data gHmmCheck;
   set gHmmCheck;
   if(&qFlipState.=1) then do;
      ds = -ds;
      correct = 1 - correct;
   end;
run;

proc print data = gHmmAccuracy noobs; run;
```

The accuracy of the Gaussian HMM, which is shown in Figure 20.16, is 99.99%. This means that the Gaussian HMM successfully takes advantage of the sequential information: only 1 data point cannot be correctly classified, compared to about 800 points that cannot be correctly classified by the cluster analysis method without the sequential information.

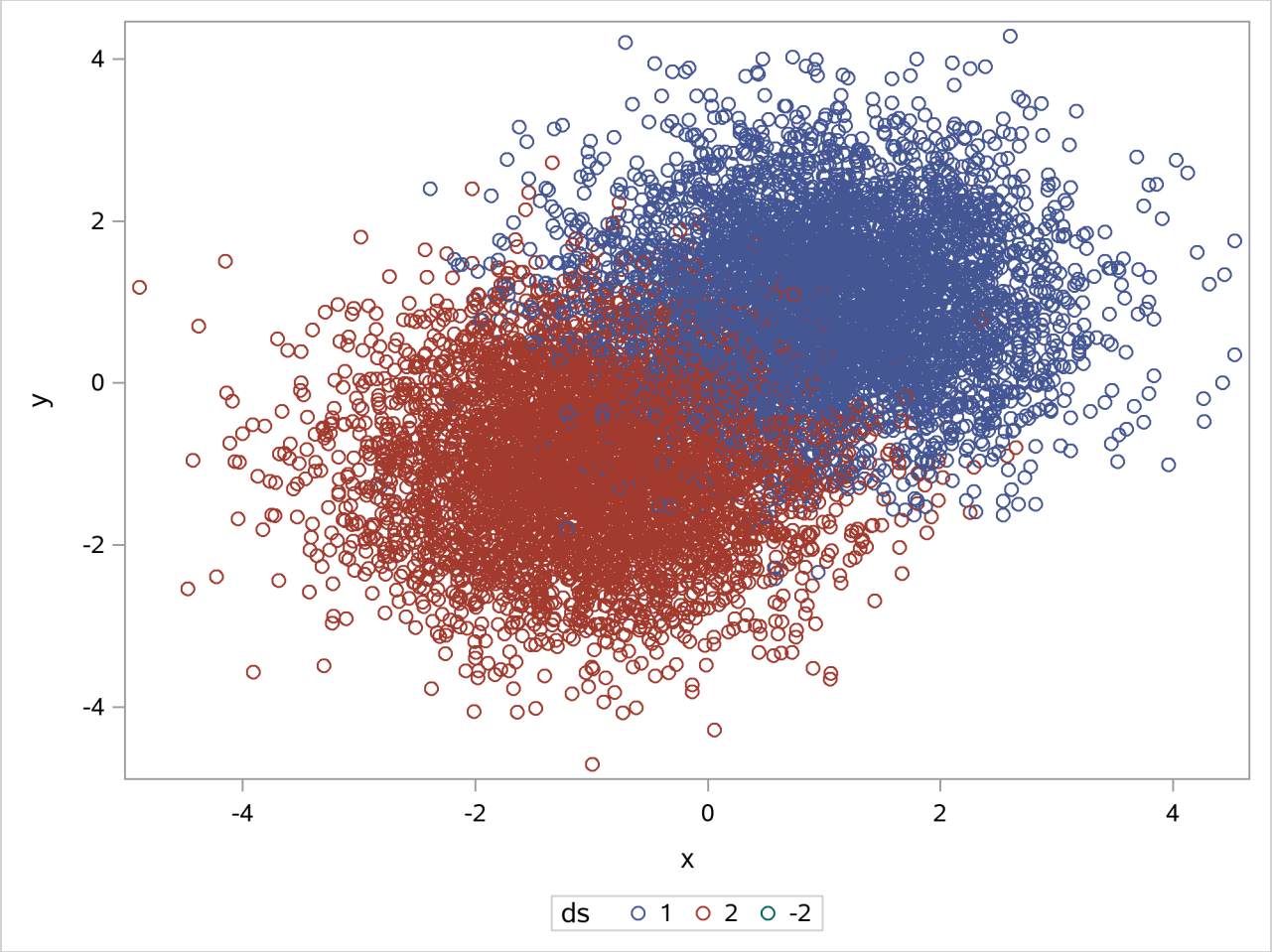
Figure 20.16 Accuracy of Decoding by the Gaussian HMM

accuracy
0.9999

To show how well the data are classified, the previous statements label any incorrectly classified data points with negative state values. In this example, only 1 point is not correctly classified. You plot the classified data points by using the following statements. The plot is shown in Figure 20.17.

```
proc sgplot data=gHmmCheck;
   scatter y=y x=x / group=ds;
run;
```

Figure 20.17 Classified Data Points by Gaussian HMM



Gaussian Hidden Markov Model for Cross-Sectional Time Series Data

Cross-sectional time series data consist of observations over both time and many subjects (such as individuals, objects, firms, or geographical areas). Let $\mathbf{Y}_{n,t} = (y_{n,1t}, \ldots, y_{n,pt})'$, where $n = 1, \ldots, N$ and $t = 1, \ldots, T_n$, denote a p-dimensional vector of random variables at the tth position in the nth section, where T_n is the sample size of the nth section. The $\mathbf{Y}_{n,t}$ follows the Gaussian HMM; that is,

$$\mathbf{Y}_{n,t}|\mathbf{S}_{n,t} \sim N(\mu_{\mathbf{S}_{n,t}}, \Sigma_{\mathbf{S}_{n,t}})$$

where $\mu_{\mathbf{S}_{n,t}}$ and $\Sigma_{\mathbf{S}_{n,t}}$ are mean and covariance parameters, respectively, whose values depend on the state variable $\mathbf{S}_{n,t}$. $\mathbf{S}_{n,t}$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_{n,t}|\mathbf{S}_{n,t-1}, \mathbf{S}_{n,t-2}, \ldots, \mathbf{S}_{n,1}) = p(\mathbf{S}_{n,t}|\mathbf{S}_{n,t-1})$$

where $p(.|.)$ denotes the conditional probability. $\mathbf{S}_{n,t}$ is independent of any $\mathbf{S}_{m,s}, m = 1, \ldots, N, s = 1, \ldots, T_m$, and $m \neq n$. The range of $\mathbf{S}_{n,t}$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_{n,t} = j | \mathbf{S}_{n,t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The initial state probability vector (ISPV), π, of the first state $\mathbf{S}_{n,1}$ is

$$\pi = \{\pi_i = p(\mathbf{S}_{n,1} = i), i = 1, \ldots, K\}$$

The Gaussian HMM for cross-sectional time series data, which is the same as the Gaussian HMM for time series data, can be described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{\mu_i, \Sigma_i, i = 1, \ldots, K\}$, which consists of parameters that define the state-dependent distribution of observable variables.

Consider a univariate six-state Gaussian HMM that has the following parameter values:

$$\pi = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.95 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.95 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.95 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.95 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.95 \end{pmatrix}$$

$$\mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \\ \mu_5 \\ \mu_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \\ \Sigma_3 \\ \Sigma_4 \\ \Sigma_5 \\ \Sigma_6 \end{pmatrix} = \begin{pmatrix} 0.0625 \\ 0.25 \\ 1 \\ 4 \\ 16 \\ 64 \end{pmatrix} \right\}$$

The following statements simulate the 10,000-section time series from the previous model to provide test data for the HMM procedure:

```
%let nSections = 10000;
%let T = 100;
%let seed = 1234;
%let nStates = 6;
%let mu1 = 0;
%let sigma1 = 0.0625;
%let mu2 = 0;
%let sigma2 = 0.25;
%let mu3 = 0;
%let sigma3 = 1;
%let mu4 = 0;
%let sigma4 = 4;
```

```
%let mu5 = 0;
%let sigma5 = 16;
%let mu6 = 0;
%let sigma6 = 64;
%let selfTransProb = 0.95;

%macro createCstsTable(tableName);
   data &tableName.;
      do sec = 1 to &nSections.;
         state = ceil(uniform(&seed.)*&nStates.);
         do t = 1 to &T.;
            %do i = 1 %to &nStates.;
               if(state=&i.) then do;
                  y = &&mu&i.. + sqrt(&&sigma&i..)*normal(&seed.);
                  output;
               end;
            %end;
            u = uniform(&seed.);
            if(u>&selfTransProb.) then do;
               u = (u-&selfTransProb.)/(1-&selfTransProb.)*(&nStates.-1);
               state=state + ceil(u);
               if(state>&nStates.) then state = state - &nStates.;
            end;
         end;
      end;
   run;
%mend;

%createCstsTable(cstsDGP);

data mylib.cstsDGP;
   set cstsDGP;
run;
```

The following statements estimate the Gaussian HMM:

```
proc hmm data=mylib.cstsDGP
   outstat=mylib.cstsStat;
   id section=sec time=t;
   model y  / method=ml type=gaussian nstate=6;
   optimize ALGORITHM=activeset printlevel=3 printIterFreq=1;
   estimate out=mylib.cstsEst;
   evaluate out=mylib.cstsEval;
   decode out=mylib.cstsDecode;
   filter out=mylib.cstsFilter;
   smooth out=mylib.cstsSmooth;
   forecast out=mylib.cstsForcast;
   score outmodel=mylib.cstsModel;
run;
```

The estimates of the ISPV, TPM, mean parameters, and covariance parameters are shown in Figure 20.18, Figure 20.19, Figure 20.20, and Figure 20.21, respectively. They are all very close to the true parameters that are used in the data generating process.

Figure 20.18 Estimates of the Initial State Probability Vector (ISPV)

Initial State Probabilities	
State	**Estimation**
1	0.16814
2	0.16340
3	0.16812
4	0.16424
5	0.16594
6	0.17016

Figure 20.19 Estimates of the Transition Probability Matrix (TPM)

Estimated Transition Probability Matrix						
State	**1**	**2**	**3**	**4**	**5**	**6**
1	0.94993	0.00963	0.01013	0.01034	0.00970	0.01028
2	0.00992	0.95071	0.01050	0.00906	0.00942	0.01037
3	0.01040	0.00973	0.94935	0.00950	0.01067	0.01035
4	0.01010	0.00976	0.00973	0.94994	0.01052	0.00995
5	0.01015	0.00939	0.01032	0.01017	0.95012	0.00985
6	0.01002	0.00963	0.01049	0.01009	0.00932	0.95045

Figure 20.20 Estimates of the Mean Parameters

Mu	
State	**y**
1	0.00025
2	-0.00200
3	-0.00254
4	0.00695
5	-0.02130
6	0.01899

Figure 20.21 Estimates of the Covariance Parameters

Sigma		
State	**Variable**	**y**
1	**y**	0.06257
2	**y**	0.24731
3	**y**	0.99708
4	**y**	3.97252
5	**y**	15.95629
6	**y**	64.01087

Gaussian Mixture Hidden Markov Model for Time Series Data and Cross-Sectional Time Series Data

Let $\mathbf{Y}_t = (y_{1t}, \ldots, y_{pt})', t = 1, \ldots, T$, denote a p-dimensional time series vector of random variables. Conditional on a latent variable $\mathbf{S}_t$, $\mathbf{Y}_t$ follows a Gaussian mixture distribution,

$$\mathbf{Y}_t|\mathbf{S}_t \sim \mathrm{GM}(\{w_{\mathbf{S}_t,1}, \ldots, w_{\mathbf{S}_t,M}\}, \{\mu_{\mathbf{S}_t,1}, \ldots, \mu_{\mathbf{S}_t,M}\}, \{\Sigma_{\mathbf{S}_t,1}, \ldots, \Sigma_{\mathbf{S}_t,M}\})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\mathrm{GM}(.,.,.)$ represents the Gaussian mixture distribution; M is the number of components at each state; and the $w_{\mathbf{S}_t,j}, j = 1, \ldots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_{\mathbf{S}_t,j} \geq 0, j = 1, \ldots, M$, and $\sum_{j=1}^{M} w_{\mathbf{S}_t,j} = 1$). The $\mu_{\mathbf{S}_t,j}$ and $\Sigma_{\mathbf{S}_t,j}, j = 1, \ldots, M$, are the mean and covariance parameters for the jth Gaussian component at state $\mathbf{S}_t$, and their values depend on the variable $\mathbf{S}_t$. The state variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t|\mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \ldots, \mathbf{S}_1) = p(\mathbf{S}_t|\mathbf{S}_{t-1})$$

where $p(.|.)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j|\mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π, of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \ldots, K\}$$

Because the variable $\mathbf{S}_t$ is unobservable and follows a Markov chain and because $\mathbf{Y}_t$ follows a Gaussian mixture distribution conditional on $\mathbf{S}_t$, the model is called the Gaussian mixture hidden Markov model (GM HMM). The GM HMM is sometimes described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{w_{ij}, \mu_{ij}, \Sigma_{ij}, i = 1, \ldots, K, j = 1, \ldots, M\}$, and it consists of parameters that define the state-dependent distribution of observable variables.

The GM HMM for cross-sectional time series data is similar to the extension from Gaussian HMM to Gaussian HMM for cross-sectional time series data; for more information, see the section "Gaussian Hidden Markov Model for Cross-Sectional Time Series Data" on page 1160.

Consider a bistate bi-component GM HMM that has the following parameter values. The ISPV and TPM are

$$\pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.05 \\ 0.15 & 0.85 \end{pmatrix}$$

At state 1, the two components have weights

$$c_{11} = 0.6, c_{12} = 0.4$$

and the mean and covariance parameters for two components are

$$\mu_{11} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma_{11} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_{12} = \begin{pmatrix} -3 \\ -3 \end{pmatrix}, \Sigma_{12} = \begin{pmatrix} 1.69 & 1.00 \\ 1.00 & 1.69 \end{pmatrix}$$

At state 2, the two components have weights

$$c_{21} = 0.7, c_{22} = 0.3$$

and the mean and covariance parameters for two components are

$$\mu_{21} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \Sigma_{22} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_{22} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \Sigma_{22} = \begin{pmatrix} 1.69 & 1.00 \\ 1.00 & 1.69 \end{pmatrix}$$

The following statements simulate the bivariate vector time series from the previous model to provide test data for the HMM procedure:

```
%let a11 = 0.98;
%let a22 = 0.95;
%let c1_1 = 0.6;
%let c1_2 = 0.4;
%let c2_1 = 0.7;
%let c2_2 = 0.3;
%let mu1_1_1 = 1;
%let mu1_1_2 = 1;
%let mu1_2_1 = -3;
%let mu1_2_2 = -3;
%let sigma1_1_11 = 1;
%let sigma1_1_21 = 0;
%let sigma1_1_22 = 1;
%let sigma1_2_11 = 1.69;
%let sigma1_2_21 = 1;
%let sigma1_2_22 = 1.69;
%let mu2_1_1 = -1;
%let mu2_1_2 = -1;
%let mu2_2_1 = 3;
%let mu2_2_2 = 3;
%let sigma2_1_11 = 1;
%let sigma2_1_21 = 0;
%let sigma2_1_22 = 1;
%let sigma2_2_11 = 1.69;
%let sigma2_2_21 = 1;
%let sigma2_2_22 = 1.69;
%let seed = 1234;
%let T = 200000;

data gmDGP;
   retain cd1_1_11 cd1_1_21 cd1_1_22
          cd1_2_11 cd1_2_21 cd1_2_22
          cd2_1_11 cd2_1_21 cd2_1_22
          cd2_2_11 cd2_2_21 cd2_2_22;
```

```
do t = 1 to &T.;
   if(t=1) then do;
      * initial probability distribution;
      p = (1-&a22.)/((1-&a11.)+(1-&a22.));;
      * Cholesky decomposition;
      cd1_1_11 = sqrt(&sigma1_1_11.);
      cd1_1_21 = &sigma1_1_21./cd1_1_11;
      cd1_1_22 = sqrt(&sigma1_1_22.-cd1_1_21*cd1_1_21);
      cd1_2_11 = sqrt(&sigma1_2_11.);
      cd1_2_21 = &sigma1_2_21./cd1_2_11;
      cd1_2_22 = sqrt(&sigma1_2_22.-cd1_2_21*cd1_2_21);
      cd2_1_11 = sqrt(&sigma2_1_11.);
      cd2_1_21 = &sigma2_1_21./cd2_1_11;
      cd2_1_22 = sqrt(&sigma2_1_22.-cd2_1_21*cd2_1_21);
      cd2_2_11 = sqrt(&sigma2_2_11.);
      cd2_2_21 = &sigma2_2_21./cd2_2_11;
      cd2_2_22 = sqrt(&sigma2_2_22.-cd2_2_21*cd2_2_21);
   end;
   else do;
      * transition probability matrix;
      if(lags=1) then p = &a11.;
      else p = 1-&a22.;
   end;
   u = uniform(&seed.);
   if(u<=p) then s=1;
   else s = 2;
   e1 = normal(&seed.); e2 = normal(&seed.);
   if(s=1) then do;
      * choose component;
      u = uniform(&seed.);
      if(u<=&c1_1.) then do;
         * (x,y) ~ N(mu, Sigma) at state 1, component 1;
         x = &mu1_1_1. + cd1_1_11*e1;
         y = &mu1_1_2. + cd1_1_21*e1+cd1_1_22*e2;
      end;
      else do;
         * (x,y) ~ N(mu, Sigma) at state 1, component 2;
         x = &mu1_2_1. + cd1_2_11*e1;
         y = &mu1_2_2. + cd1_2_21*e1+cd1_2_22*e2;
      end;
   end;
   else do;
      * choose component;
      u = uniform(&seed.);
      if(u<=&c2_1.) then do;
         * (x,y) ~ N(mu, Sigma) at state 2, component 1;
         x = &mu2_1_1. + cd2_1_11*e1;
         y = &mu2_1_2. + cd2_1_21*e1+cd2_1_22*e2;
      end;
      else do;
         * (x,y) ~ N(mu, Sigma) at state 2, component 2;
         x = &mu2_2_1. + cd2_2_11*e1;
         y = &mu2_2_2. + cd2_2_21*e1+cd2_2_22*e2;
      end;
```

```
      end;
      output;
      lags = s;
   end;
run;

data gmTrain;
   set gmDGP(where=(t<=&T./2));
   keep t x y;
run;

data gmTest;
   set gmDGP(where=(t>&T./2));
   keep t x y;
run;
```

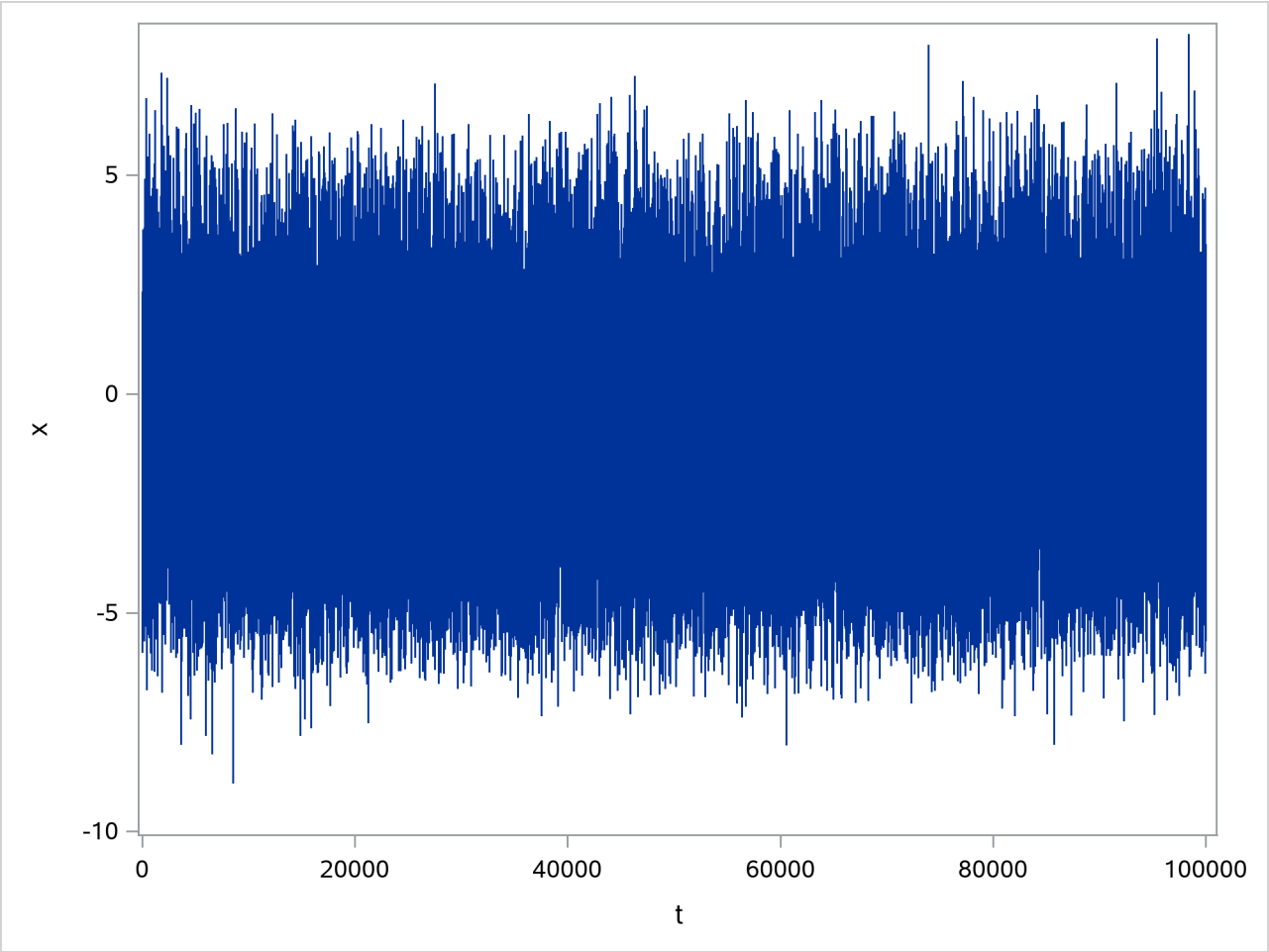
In general, the data generating process is unknown. The data tables gmTrain and gmTest can be seen, but the data table gmDGP cannot be seen.

First, assume that the following statements plot the simulated series x that is contained in the data table gmTrain:

```
proc sgplot data=gmTrain;
   series x=t y=x;
run;
```

Figure 20.22 shows the plot of x, whose points seem to fall randomly around 0.

Figure 20.22 Plot of Time Series x

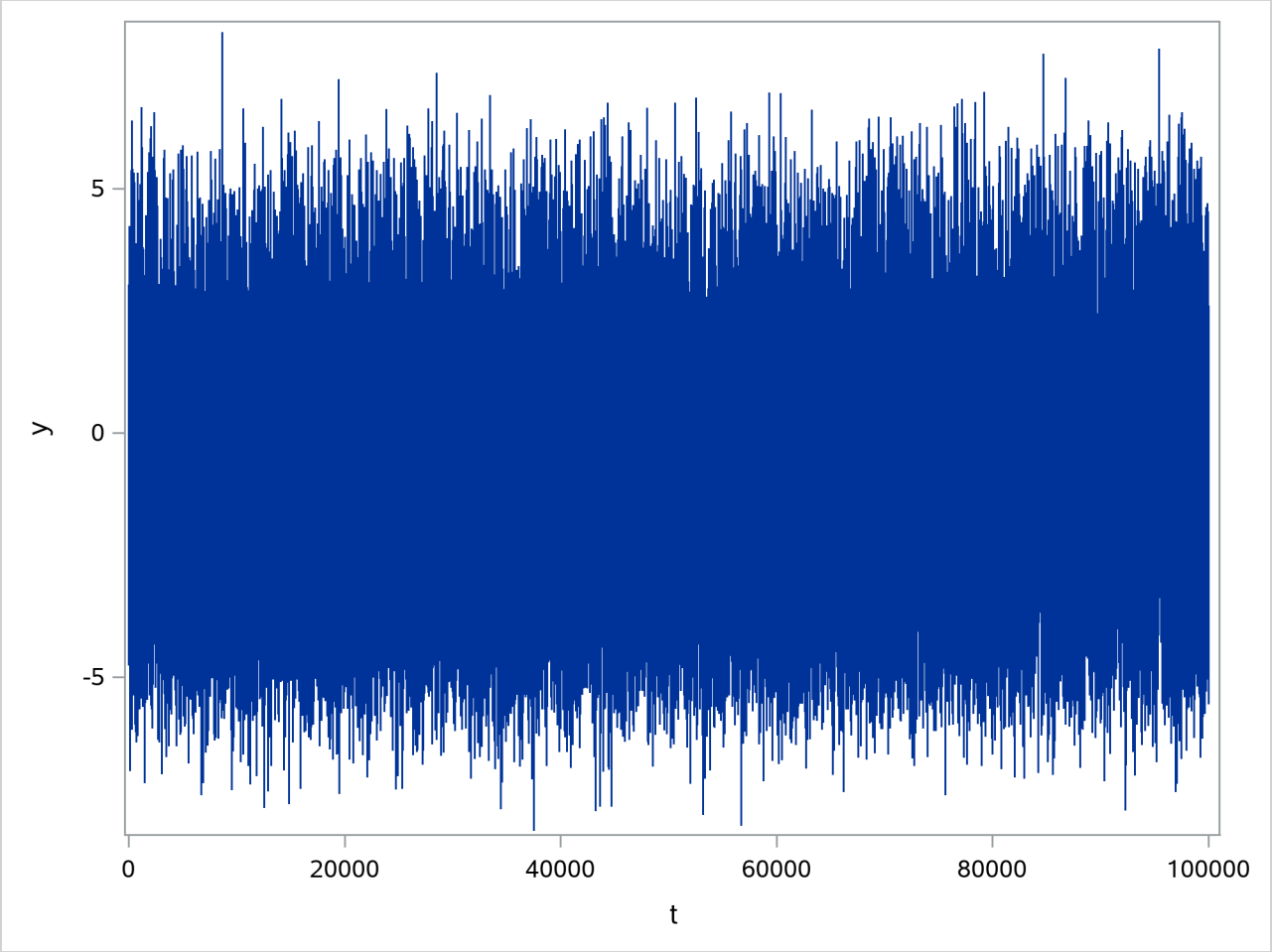


The following statements plot the simulated series y that is contained in the data set gmTrain:

```
proc sgplot data=gmTrain;
   series x=t y=y;
run;
```

Figure 20.23 shows the plot of y, whose points also seem to fall randomly around 0.

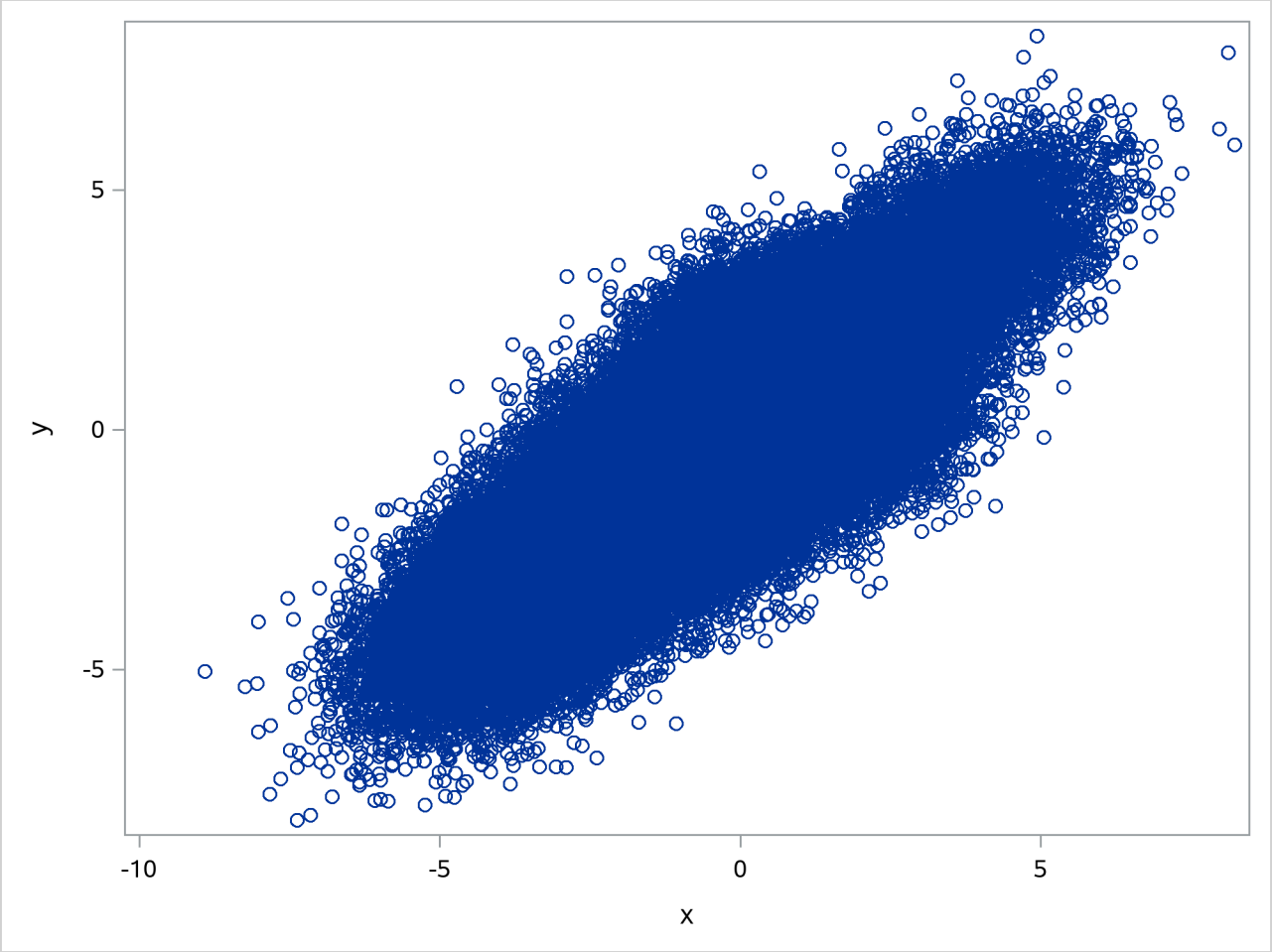
Figure 20.23 Plot of Time Series y



Ignoring the sequential information, the following statements plot the observed points (x, y) that are contained in the data table gmTrain:

```
proc sgplot data=gmTrain;
   scatter y=y x=x;
run;
```

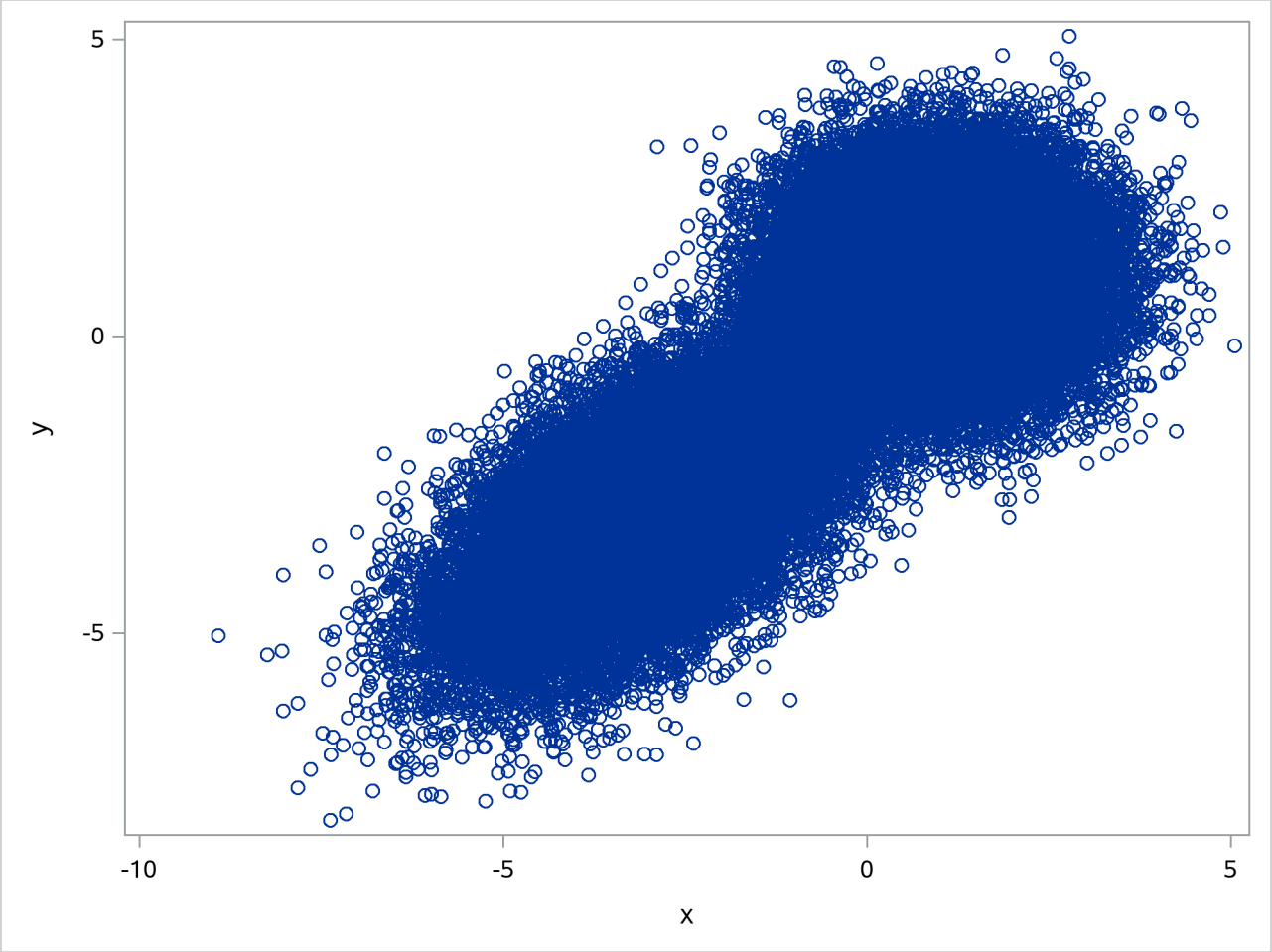
Figure 20.24 Unclassified Observations



As shown in Figure 20.24, it seems that all points come from one bivariate Gaussian distribution that has a high positive correlation. However, according to the true DGP, some points are from state 1, in which there are two clusters, as shown in Figure 20.25:

```
proc sgplot data=gmDGP(where=(t<=&T./2 and s=1));
   scatter y=y x=x;
run;
```

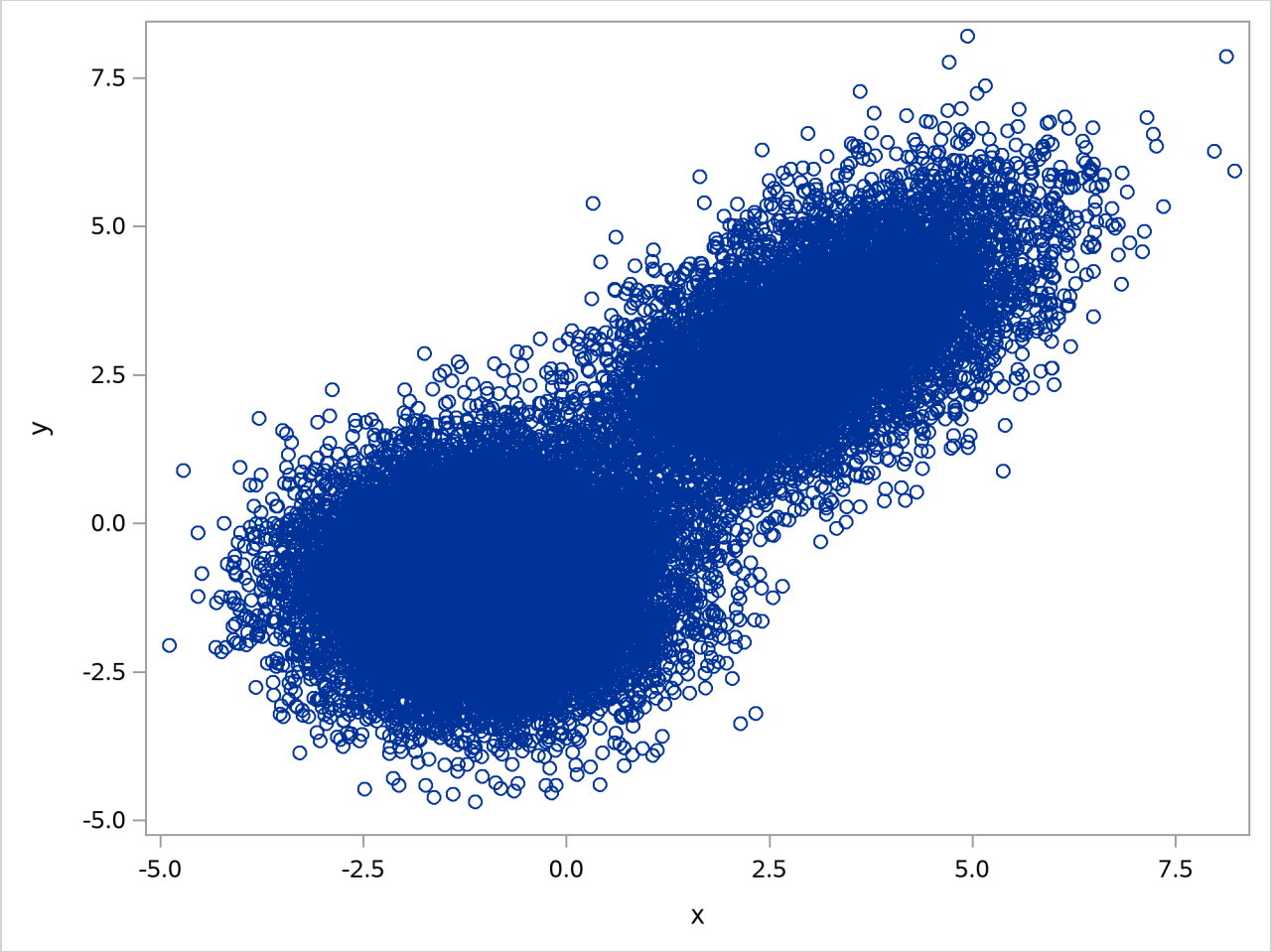
Figure 20.25 Observations from State 1



Other points are from state 2 in which there are two clusters, as shown in Figure 20.26:

```
proc sgplot data=gmDGP(where=(t<=&T./2 and s=2));
   scatter y=y x=x;
run;
```

Figure 20.26 Observations from State 2



Without considering the sequential order, it is impossible to distinguish the two states.

Before you estimate any model, you should make a copy of the data in the library, as in the following DATA steps:

```
data mylib.gmTrain; set gmTrain; run;
data mylib.gmTest; set gmTest; run;
```

The following statements estimate the bistate bi-component GM HMM. The type of model is specified in the TYPE= option in the MODEL statement. The number of states is specified in the NSTATE= option in the MODEL statement. The number of components for each state is specified in the NCOMPONENT= option in the MODEL statement. The OUTMODEL= option in the SCORE statement outputs the model information for future scoring on new data table. The LABELSWITCH=(SORT=NONE) option in the PROC HMM statement suppresses the relabeling of states and components; for more information about the label switching problem, see the section "Label Switching Problem" on page 1283.

```
proc hmm data=mylib.gmTrain labelSwitch=(sort=none);
   id time=t;
   model x y / type=gaussianMixture nstate=2 ncomponent=2;
   optimize printLevel=3 printIterFreq=1;
   score outmodel=mylib.gmModel;
run;
```

These statements assume that your engine libref is named mylib, but you can substitute any appropriately defined engine libref.

The number of observations and model information are shown in Figure 20.27.

Figure 20.27 Number of Observations and Model Information

Observation Information	
Number of Observations	100000
Number of Missing Observations	0

Model Information	
Type of Model	Gaussian Mixture HMM
Stationary	Yes
Number of States	2
Number of Components for a State	2
Number of Dependent Variables	2

The default method, which is the maximum likelihood method, is applied; this is a nonlinear optimization problem. The initial parameter values and the objective function value at the start of the optimization are shown in Figure 20.28.

Figure 20.28 Initial Parameter Values and Objective Function Value

Optimization Start	
Parameter	**Estimate**
TPM1_1	0.500000
TPM1_2	0.500000
TPM2_1	0.500000
TPM2_2	0.500000
MCP1_1	0.500000
MCP1_2	0.500000
MCP2_1	0.500000
MCP2_2	0.500000
MU1_1_1	0.416654
MU1_1_2	0.961871
MU1_2_1	-3.422684
MU1_2_2	-3.125473
MU2_1_1	2.963828
MU2_1_2	1.590644
MU2_2_1	-0.815362
MU2_2_2	-2.039325
SIGMA1_1_1_1	1.354409
SIGMA1_1_2_1	0.762768
SIGMA1_1_2_2	1.393576
SIGMA1_2_1_1	0.961355
SIGMA1_2_2_1	0.661717
SIGMA1_2_2_2	1.581451
SIGMA2_1_1_1	1.025889
SIGMA2_1_2_1	1.399630
SIGMA2_1_2_2	2.884377
SIGMA2_2_1_1	0.749535
SIGMA2_2_2_1	0.265184
SIGMA2_2_2_2	0.633097

Initial Value of Objective Function	-401529.3793

The details of the optimization algorithm are shown in Figure 20.29.

Figure 20.29 Optimization Algorithm Details

Algorithm Settings	
Maximum Number of Iterations	128
Tolerance for Optimality Error	1E-6
Tolerance for Infeasibility	1E-6
Maximum Allowed Time (seconds)	1.797693E308
Maximum Magnitude of Objective Function Value	1E20
Random Seed	1
Solution Type	1
Multi-Start Globalization Scheme	0
Multi-Start Globalization Bound Range	2

The iterations of the optimization process are shown in Figure 20.30.

Figure 20.30 Iterations of the Optimization Process

Iteration History

Iteration	Objective Function	Optimality Error	Infeasibility	Function Calls	Gradient Calls	Hessian Calls
1	-401529.3793	2887.16466	0.00000	2	1	0
2	-389554.1158	2133.36152	0.00000	10	9	1
3	-386179.6546	1199.52355	0.00000	19	18	2
4	-371985.2011	848.19186	0.00000	29	28	3
5	-367516.0600	249.88933	0.00000	38	37	4
6	-366322.8416	161.33937	0.00000	49	48	5
7	-365982.2352	46.74362	0.00000	59	58	6
8	-365945.5123	5.29758	0.00000	69	68	7
9	-365945.0948	0.06672	0.00000	79	78	8
10	-365945.0946	0.00000	0.00000	89	88	9

The final parameter values and the objective function value at the end of the optimization are shown in Figure 20.31.

Figure 20.31 Final Parameter Values and Objective Function Value

Optimization Results	
Parameter	**Estimate**
TPM1_1	0.979636
TPM1_2	0.020364
TPM2_1	0.049729
TPM2_2	0.950271
MCP1_1	0.600332
MCP1_2	0.399668
MCP2_1	0.297811
MCP2_2	0.702189
MU1_1_1	0.993524
MU1_1_2	0.996707
MU1_2_1	-3.003906
MU1_2_2	-3.005379
MU2_1_1	3.005306
MU2_1_2	3.009691
MU2_2_1	-0.997604
MU2_2_2	-1.005076
SIGMA1_1_1_1	1.000305
SIGMA1_1_2_1	0.007588
SIGMA1_1_2_2	1.003672
SIGMA1_2_1_1	1.704270
SIGMA1_2_2_1	1.003413
SIGMA1_2_2_2	1.688212
SIGMA2_1_1_1	1.712932
SIGMA2_1_2_1	1.014865
SIGMA2_1_2_2	1.704181
SIGMA2_2_1_1	1.015895
SIGMA2_2_2_1	0.012899
SIGMA2_2_2_2	1.006142

Final Value of Objective Function	-365945.0946

The estimates of the initial state probability vector (ISPV) are shown in Figure 20.32.

Figure 20.32 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	**Estimation**
1	0.70947
2	0.29053

The estimates of the transition probability matrix (TPM) are shown in Figure 20.33.

Figure 20.33 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	**1**	**2**
1	0.97964	0.02036
2	0.04973	0.95027

The estimates of the mixture component probabilities (MCP) are shown in Figure 20.34.

Figure 20.34 Estimates of Mixture Component Probabilities

Estimated Mixture Component Probabilities		
	Component	
State	**1**	**2**
1	0.60033	0.39967
2	0.29781	0.70219

The estimates of the mean vector of the Gaussian distribution for each component at each state are shown in Figure 20.35.

Figure 20.35 Estimate of Mean Vector for Each Component at Each State

		Mu	
State	**Component**	**x**	**y**
1	**1**	0.99352	0.99671
	2	-3.00391	-3.00538
2	**1**	3.00531	3.00969
	2	-0.99760	-1.00508

The estimates of the covariance matrix of the Gaussian distribution for each component at each state are shown in Figure 20.36.

Figure 20.36 Estimate of Covariance Matrix for Each Component at Each State

		Sigma		
State	**Component**	**Variable**	**x**	**y**
1	**1**	**x**	1.00031	0.00759
	1	**y**	0.00759	1.00367
	2	**x**	1.70427	1.00341
	2	**y**	1.00341	1.68821
2	**1**	**x**	1.71293	1.01486
	1	**y**	1.01486	1.70418
	2	**x**	1.01589	0.01290
	2	**y**	0.01290	1.00614

The estimates of all parameters are shown in Figure 20.37, which displays columns for the parameter name, estimate value, standard error, t value, and p-value. The state labels ("1" and "2" in this example) and the

component labels (“1” and “2” in this example) might exchange randomly, compared to the true DGP. In the preceding estimation, the state labels are the same as the labels in the DGP. The component labels for state 1 are also the same as the ones in the DGP, but the component labels for state 2 do exchange! Compared to the parameter values in the DGP, the estimates are very accurate.

Figure 20.37 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.979636	0.000600	1632.32	<.0001
TPM1_2	0.020364	0.000600	33.93	<.0001
TPM2_1	0.049729	0.001422	34.98	<.0001
TPM2_2	0.950271	0.001422	668.37	<.0001
MCP1_1	0.600332	0.001987	302.17	<.0001
MCP1_2	0.399668	0.001987	201.17	<.0001
MCP2_1	0.297811	0.002966	100.42	<.0001
MCP2_2	0.702189	0.002966	236.78	<.0001
MU1_1_1	0.993524	0.005202	191.00	<.0001
MU1_1_2	0.996707	0.005225	190.77	<.0001
MU1_2_1	-3.003906	0.009259	-324.43	<.0001
MU1_2_2	-3.005379	0.009199	-326.71	<.0001
MU2_1_1	3.005306	0.017586	170.89	<.0001
MU2_1_2	3.009691	0.017492	172.06	<.0001
MU2_2_1	-0.997604	0.007672	-130.04	<.0001
MU2_2_2	-1.005076	0.007645	-131.47	<.0001
SIGMA1_1_1_1	1.000305	0.007559	132.33	<.0001
SIGMA1_1_2_1	0.007588	0.005615	1.35	0.1765
SIGMA1_1_2_2	1.003672	0.007625	131.63	<.0001
SIGMA1_2_1_1	1.704270	0.018040	94.47	<.0001
SIGMA1_2_2_1	1.003413	0.015372	65.27	<.0001
SIGMA1_2_2_2	1.688212	0.017808	94.80	<.0001
SIGMA2_1_1_1	1.712932	0.033960	50.44	<.0001
SIGMA2_1_2_1	1.014865	0.028939	35.07	<.0001
SIGMA2_1_2_2	1.704181	0.033648	50.65	<.0001
SIGMA2_2_1_1	1.015895	0.011295	89.94	<.0001
SIGMA2_2_2_1	0.012899	0.008289	1.56	0.1197
SIGMA2_2_2_2	1.006142	0.011248	89.45	<.0001

The log likelihood and information criteria are shown in Figure 20.38.

Figure 20.38 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-365945.0946
AIC	731942.18929
AICC	731942.20333
BIC	732189.52535
HQC	732017.24975

The estimates from the data table gmTrain are used to score the testing data table gmTest to check the predictability in the following statements:

```
proc hmm data=mylib.gmTest;
   score inmodel=mylib.gmModel;
   decode out=mylib.gmDecode;
run;
```

The accuracy of classification through the GM HMM is calculated by the following statements:

```
data gmHmmCheck;
   merge gmDGP(in=a where=(t>&T./2)) mylib.gmDecode(in=b);
   by t;
   if(s=state) then do; correct=1; ds = s; end;
   else do; correct=0; ds = -s; end;
   if(a=b);
   keep t x y s correct ds;
run;

%let qFlipState = -1;
data gmHmmAccuracy;
   set gmHmmCheck;
   retain count 0 correctCount 0;
   count = count + 1;
   correctCount = correctCount + correct;
   if(count>&T./2-0.5) then do;
      accuracy = correctCount / count;
      if(accuracy<0.5) then call symputx("qFlipState",1,'G');
      else call symputx("qFlipState",0,'G');
      if(accuracy<0.5) then accuracy = 1 - accuracy;
      output;
   end;
   keep accuracy;
run;

data gmHmmCheck;
   set gmHmmCheck;
   if(&qFlipState.=1) then do;
      ds = -ds;
      correct = 1 - correct;
   end;
run;

proc print data = gmHmmAccuracy noobs; run;
```

The accuracy of the GM HMM, which is shown in Figure 20.39, is about 98%. This means that the GM HMM successfully takes advantage of the sequential information.

Figure 20.39 Accuracy of Decoding by the GM HMM

accuracy
0.97835

Now, consider the GM HMM for cross-sectional time series data. Each section's data follow the same GM HMM as described previously. The following statements generate the data table for analysis. There are 200 sections in the data table, and each section of time series consist of 1,000 observations.

```
%let a11 = 0.98;
%let a22 = 0.95;
%let c1_1 = 0.6;
%let c1_2 = 0.4;
%let c2_1 = 0.7;
%let c2_2 = 0.3;
%let mu1_1_1 = 1;
%let mu1_1_2 = 1;
%let mu1_2_1 = -3;
%let mu1_2_2 = -3;
%let sigma1_1_11 = 1;
%let sigma1_1_21 = 0;
%let sigma1_1_22 = 1;
%let sigma1_2_11 = 1.69;
%let sigma1_2_21 = 1;
%let sigma1_2_22 = 1.69;
%let mu2_1_1 = -1;
%let mu2_1_2 = -1;
%let mu2_2_1 = 3;
%let mu2_2_2 = 3;
%let sigma2_1_11 = 1;
%let sigma2_1_21 = 0;
%let sigma2_1_22 = 1;
%let sigma2_2_11 = 1.69;
%let sigma2_2_21 = 1;
%let sigma2_2_22 = 1.69;
%let seed = 1234;
%let T = 200000;
%let nSections = 200;

data gmDGP;
   retain cd1_1_11 cd1_1_21 cd1_1_22
          cd1_2_11 cd1_2_21 cd1_2_22
          cd2_1_11 cd2_1_21 cd2_1_22
          cd2_2_11 cd2_2_21 cd2_2_22;
   do t = 1 to &T.;
      sec = ceil(t*&nSections./&T.);
      if(t=1) then do;
         * initial probability distribution;
         p = (1-&a22.)/((1-&a11.)+(1-&a22.));;
         * Cholesky decomposition;
         cd1_1_11 = sqrt(&sigma1_1_11.);
         cd1_1_21 = &sigma1_1_21./cd1_1_11;
         cd1_1_22 = sqrt(&sigma1_1_22.-cd1_1_21*cd1_1_21);
         cd1_2_11 = sqrt(&sigma1_2_11.);
         cd1_2_21 = &sigma1_2_21./cd1_2_11;
         cd1_2_22 = sqrt(&sigma1_2_22.-cd1_2_21*cd1_2_21);
         cd2_1_11 = sqrt(&sigma2_1_11.);
         cd2_1_21 = &sigma2_1_21./cd2_1_11;
         cd2_1_22 = sqrt(&sigma2_1_22.-cd2_1_21*cd2_1_21);
         cd2_2_11 = sqrt(&sigma2_2_11.);
         cd2_2_21 = &sigma2_2_21./cd2_2_11;
```

```
         cd2_2_22 = sqrt(&sigma2_2_22.-cd2_2_21*cd2_2_21);
      end;
      else do;
         * transition probability matrix;
         if(lags=1) then p = &a11.;
         else p = 1-&a22.;
      end;
      u = uniform(&seed.);
      if(u<=p) then s=1;
      else s = 2;
      e1 = normal(&seed.); e2 = normal(&seed.);
      if(s=1) then do;
         * choose component;
         u = uniform(&seed.);
         if(u<=&c1_1.) then do;
            * (x,y) ~ N(mu, Sigma) at state 1, component 1;
            x = &mu1_1_1. + cd1_1_11*e1;
            y = &mu1_1_2. + cd1_1_21*e1+cd1_1_22*e2;
         end;
         else do;
            * (x,y) ~ N(mu, Sigma) at state 1, component 2;
            x = &mu1_2_1. + cd1_2_11*e1;
            y = &mu1_2_2. + cd1_2_21*e1+cd1_2_22*e2;
         end;
      end;
      else do;
         * choose component;
         u = uniform(&seed.);
         if(u<=&c2_1.) then do;
            * (x,y) ~ N(mu, Sigma) at state 2, component 1;
            x = &mu2_1_1. + cd2_1_11*e1;
            y = &mu2_1_2. + cd2_1_21*e1+cd2_1_22*e2;
         end;
         else do;
            * (x,y) ~ N(mu, Sigma) at state 2, component 2;
            x = &mu2_2_1. + cd2_2_11*e1;
            y = &mu2_2_2. + cd2_2_21*e1+cd2_2_22*e2;
         end;
      end;
      output;
      lags = s;
   end;
run;

data gmTrain;
   set gmDGP(where=(sec<=&nSections./2));
   keep sec t x y;
run;

data gmTest;
   set gmDGP(where=(sec>&nSections./2));
   keep sec t x y;
run;
```

The following statements estimate the GM HMM for the cross-sectional time series data after making the copies of the data tables in the library. The SECTION= option in the ID statement requests that the cross-sectional time series data be analyzed.

```
data mylib.gmTrain; set gmTrain; run;
data mylib.gmTest; set gmTest; run;

proc hmm data=mylib.gmTrain labelSwitch=(sort=none);
   id time=t section=sec;
   model x y / type=gaussianMixture nstate=2 ncomponent=2;
   optimize printLevel=3 printIterFreq=1;
   score outmodel=mylib.gmModel;
   filter out=mylib.gmFilter;
   evaluate out=mylib.gmEval;
run;
```

The model information is shown in Figure 20.40. There are 100 sections in the data.

Figure 20.40 Model Information

Model Information	
Type of Model	Gaussian Mixture HMM
Stationary	Yes
Number of States	2
Number of Components for a State	2
Number of Dependent Variables	2
Number of Sections	100

The estimates of all parameters are shown in Figure 20.41; the estimates are very close to the parameter values in the DGP.

Figure 20.41 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.979697	0.000598	1639.23	<.0001
TPM1_2	0.020303	0.000598	33.97	<.0001
TPM2_1	0.049991	0.001419	35.23	<.0001
TPM2_2	0.950009	0.001419	669.51	<.0001
MCP1_1	0.600330	0.001987	302.13	<.0001
MCP1_2	0.399670	0.001987	201.15	<.0001
MCP2_1	0.297864	0.002966	100.41	<.0001
MCP2_2	0.702136	0.002966	236.70	<.0001
MU1_1_1	0.993469	0.005202	190.97	<.0001
MU1_1_2	0.996750	0.005224	190.80	<.0001
MU1_2_1	-3.004280	0.009259	-324.45	<.0001
MU1_2_2	-3.005705	0.009202	-326.64	<.0001
MU2_1_1	3.004517	0.017596	170.75	<.0001
MU2_1_2	3.008912	0.017510	171.84	<.0001
MU2_2_1	-0.997409	0.007674	-129.97	<.0001
MU2_2_2	-1.005153	0.007644	-131.50	<.0001
SIGMA1_1_1_1	1.000332	0.007560	132.32	<.0001
SIGMA1_1_2_1	0.007321	0.005614	1.30	0.1922
SIGMA1_1_2_2	1.003283	0.007624	131.60	<.0001
SIGMA1_2_1_1	1.703717	0.018034	94.47	<.0001
SIGMA1_2_2_1	1.002882	0.015368	65.26	<.0001
SIGMA1_2_2_2	1.687759	0.017806	94.78	<.0001
SIGMA2_1_1_1	1.713186	0.033969	50.43	<.0001
SIGMA2_1_2_1	1.015588	0.028964	35.06	<.0001
SIGMA2_1_2_2	1.705412	0.033689	50.62	<.0001
SIGMA2_2_1_1	1.015835	0.011301	89.89	<.0001
SIGMA2_2_2_1	0.012931	0.008291	1.56	0.1188
SIGMA2_2_2_2	1.005816	0.011243	89.46	<.0001

You can apply the estimates from the data table gmTrain to the testing data table gmTest to check the predictability in the following statements:

```
proc hmm data=mylib.gmTest;
   score inmodel=mylib.gmModel;
   decode out=mylib.gmDecode;
   filter out=mylib.gmFilter2;
   evaluate out=mylib.gmEval2;
run;

data gmHmmCheck;
   merge gmDGP(in=a where=(sec>&nSections./2)) mylib.gmDecode(in=b);
   by t;
   if(s=state) then do; correct=1; ds = s; end;
   else do; correct=0; ds = -s; end;
   if(a=b);
   keep sec t x y s correct ds;
```

```
run;

%let qFlipState = -1;
data gmHmmAccuracy;
   set gmHmmCheck;
   retain count 0 correctCount 0;
   count = count + 1;
   correctCount = correctCount + correct;
   if(count>&T./2-0.5) then do;
      accuracy = correctCount / count;
      if(accuracy<0.5) then call symputx("qFlipState",1,'G');
      else call symputx("qFlipState",0,'G');
      if(accuracy<0.5) then accuracy = 1 - accuracy;
      output;
   end;
   keep accuracy;
run;

data gmHmmCheck;
   set gmHmmCheck;
   if(&qFlipState.=1) then do;
      ds = -ds;
      correct = 1 - correct;
   end;
run;

proc print data = gmHmmAccuracy noobs; run;
```

The accuracy of the GM HMM for cross-sectional time series data, which is shown in Figure 20.42, is also around 98%.

Figure 20.42 Accuracy of Decoding by the GM HMM for Cross-Sectional Time Series Data

accuracy
0.97827

Regime-Switching Regression Model

Let $\mathbf{Y}_t = (y_{1t}, \ldots, y_{pt})', t = 1, \ldots, T$, denote a p-dimensional time series vector of random variables. The $\mathbf{Y}_t$ can be modeled in the regression

$$\mathbf{Y}_t = \beta_{\mathbf{S}_t} z_t + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, the regressor z_t is observable, the latent variable $\mathbf{S}_t$ is the so-called state, and $\beta_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the state $\mathbf{S}_t$. The variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \ldots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(.|.)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j \,|\mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π, of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \ldots, K\}$$

Because the regression parameters depend on the state, the $\mathbf{Y}_t$ follows different regressions in different regimes; hence, this type of model is called the regime-switching regression model.

Consider a univariate regime-switching regression model that has two regimes and two regressors (an intercept and an exogenous variable):

$$y_t = \begin{cases} x_t + \varepsilon_t, \varepsilon_t \sim N(0, 2.56) & \text{if } \mathbf{S}_t = 1 \\ 4 + 1.5x_t + \varepsilon_t, \varepsilon_t \sim N(0, 4) & \text{if } \mathbf{S}_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$$

The following statements simulate the time series from the previous model to provide test data for the HMM procedure:

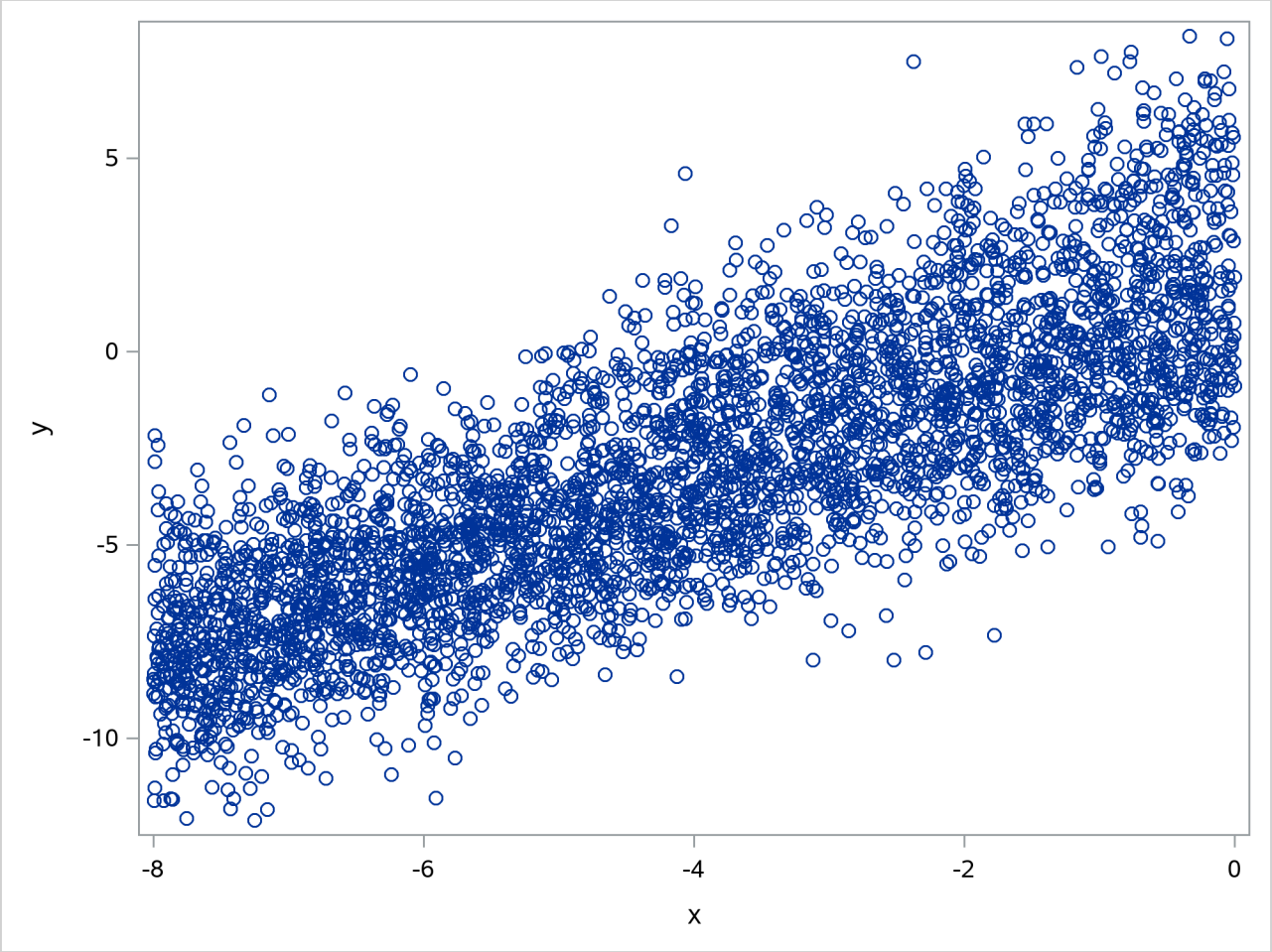
```
%let x_lb = -8;
%let x_ub = 0;
%let pi1 = 0.5;
%let a11 = 0.95;
%let a22 = 0.95;
%let const1_1 = 0;
%let xl1_0_1_1 = 1;
%let cov1_11 = 2.56;
%let const2_1 = 4;
%let xl2_0_1_1 = 1.5;
%let cov2_11 = 4;
%let T = 4000;
%let seed = 1234;

data rsregdgp;
   retain cd1_11 cd2_11;
   do t = 1 to &T.;
      if(t=1) then do;
         /* initial probability distribution */
         p = &pi1.;
         /* Cholesky decomposition of COV1 */
```

```
         cd1_11 = sqrt(&cov1_11.);
         /* Cholesky decomposition of COV2 */
         cd2_11 = sqrt(&cov2_11.);
      end;
      else do;
         /* transition probability matrix */
         if(lags=1) then p = &a11.;
         else p = 1-&a22.;
      end;
      u = uniform(&seed.);
      if(u<=p) then s=1;
      else s = 2;
      x = &x_lb. + (&x_ub. - &x_lb.)*uniform(&seed.);
      e = normal(&seed.);
      if(s=1) then do;
         /* y ~ N(beta1_0+beta1_1*x, Sigma1) at state 1 */
         y = &const1_1. + &xl1_0_1_1 * x + cd1_11*e;
      end;
      else do;
         /* y ~ N(beta2_0+beta2_1*x, Sigma2) at state 2 */
         y = &const2_1. + &xl2_0_1_1 * x + cd2_11*e;
      end;
      output;
      lags = s;
   end;
run;

data rsreg;
   set rsregdgp;
   keep t y x;
run;
```

The following statements create the scatter plot in Figure 20.43:

```
proc sgplot data=rsreg;
   scatter y=y x=x;
run;
```

Figure 20.43 Scatter Plot of Data Points



As shown in Figure 20.43, when the time dependence is ignored, it is difficult—if not impossible—to distinguish the two regimes.

The following code uploads the data table and estimates the regime-switching regression model:

```
data mylib.rsreg; set rsreg; run;
proc hmm data=mylib.rsreg;
   id time=t;
   model y = x / type=reg nstate=2 method=ml;
   optimize algorithm=activeset maxiter=256 printLevel=3 printIterFreq=1;
   learn out=mylib.mylearn;
   filter out=mylib.myfilter;
   smooth out=mylib.mysmooth;
   decode out=mylib.mydecode;
   evaluate out=mylib.myeval;
run;
```

These statements assume that your engine libref is named mylib, but you can substitute any appropriately defined engine libref.

The estimates of all parameters are shown in Figure 20.44, which displays columns for the parameter name, estimate value, standard error, t value, and p-value. The parameter estimates are very close to the true parameter values that are used in the data generating process.

Figure 20.44 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.950779	0.006259	151.91	<.0001
TPM1_2	0.049221	0.006259	7.86	<.0001
TPM2_1	0.057042	0.007335	7.78	<.0001
TPM2_2	0.942958	0.007335	128.56	<.0001
CONST1_1	-0.058860	0.079933	-0.74	0.4616
CONST2_1	4.105147	0.113280	36.24	<.0001
XL1_0_1_1	0.986513	0.016304	60.51	<.0001
XL2_0_1_1	1.513899	0.022254	68.03	<.0001
COV1_1_1	2.563899	0.094248	27.20	<.0001
COV2_1_1	3.839914	0.143172	26.82	<.0001

The accuracy of classification through the regime-switching regression model is calculated by the following statements:

```
data rsregCheck;
   merge rsregdgp(in=a) mylib.mydecode(in=b);
   by t;
   if(s=state) then do; correct=1; ds = s; end;
   else do; correct=0; ds = -s; end;
   if(a=b);
   keep t x y s correct ds;
run;

%let qFlipState = -1;
data rsregAccuracy;
   set rsregCheck;
   retain count 0 correctCount 0;
   count = count + 1;
   correctCount = correctCount + correct;
   if(count>&T.-0.5) then do;
      accuracy = correctCount / count;
      if(accuracy<0.5) then call symputx("qFlipState",1,'G');
      else call symputx("qFlipState",0,'G');
      if(accuracy<0.5) then accuracy = 1 - accuracy;
      output;
   end;
   keep accuracy;
run;

data rsregCheck;
   set rsregCheck;
   if(&qFlipState.=1) then do;
      ds = -ds;
      correct = 1 - correct;
   end;
run;
```

```
proc print data = rsregAccuracy noobs; run;
```

The accuracy of the regime-switching regression model, which is shown in Figure 20.45, is close to 90%. This means that the regime-switching regression model successfully distinguishes the two regimes.

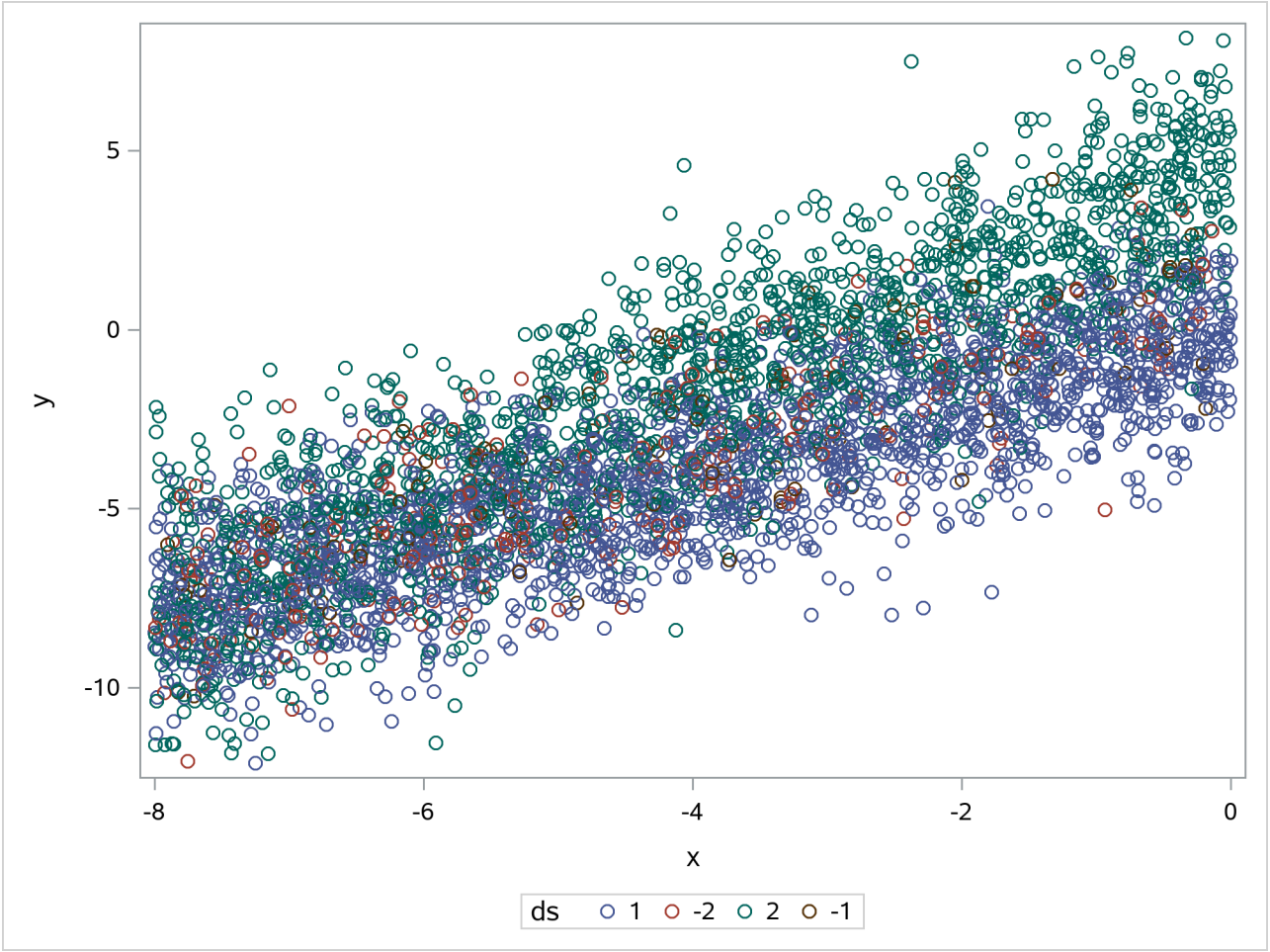
Figure 20.45 Accuracy of Decoding by the Regime-Switching Regression Model

accuracy
0.89925

The following statements create the scatter plot in Figure 20.46 to show how well the data points are classified:

```
proc sgplot data=rsregCheck;
   scatter y=y x=x / group=ds;
run;
```

Figure 20.46 Scatter Plot of Data Points



Regime-Switching Autoregression Model

Let $\mathbf{Y}_t = (y_{1t}, \ldots, y_{pt})', t = 1, \ldots, T$, denote a p-dimensional time series vector of random variables. The $\mathbf{Y}_t$ can be modeled in the regression

$$\mathbf{Y}_t = \beta_{\mathbf{S}_t} z_t + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, the regressor z_t is observable and contains the lagged dependent variables, the latent variable $\mathbf{S}_t$ is the so-called state, and $\beta_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the state $\mathbf{S}_t$. The variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t|\mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \ldots, \mathbf{S}_1) = p(\mathbf{S}_t|\mathbf{S}_{t-1})$$

where $p(.|.)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j|\mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π, of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \ldots, K\}$$

Because the lagged dependent variables are included in the regressors and the autoregression parameters depend on the state, the $\mathbf{Y}_t$ follows different autoregressions in different regimes; hence, this type of model is called the regime-switching autoregression model.

Consider a univariate regime-switching autoregression model that has two regimes:

$$y_t = \begin{cases} 0.8 * y_{t-1} + \varepsilon_t, \varepsilon_t \sim N(0, 2.56) & \text{if } \mathbf{S}_t = 1 \\ -.7 * y_{t-1} + \varepsilon_t, \varepsilon_t \sim N(0, 4) & \text{if } \mathbf{S}_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix}$$

The following statements simulate the time series from the previous model to provide test data for the HMM procedure:

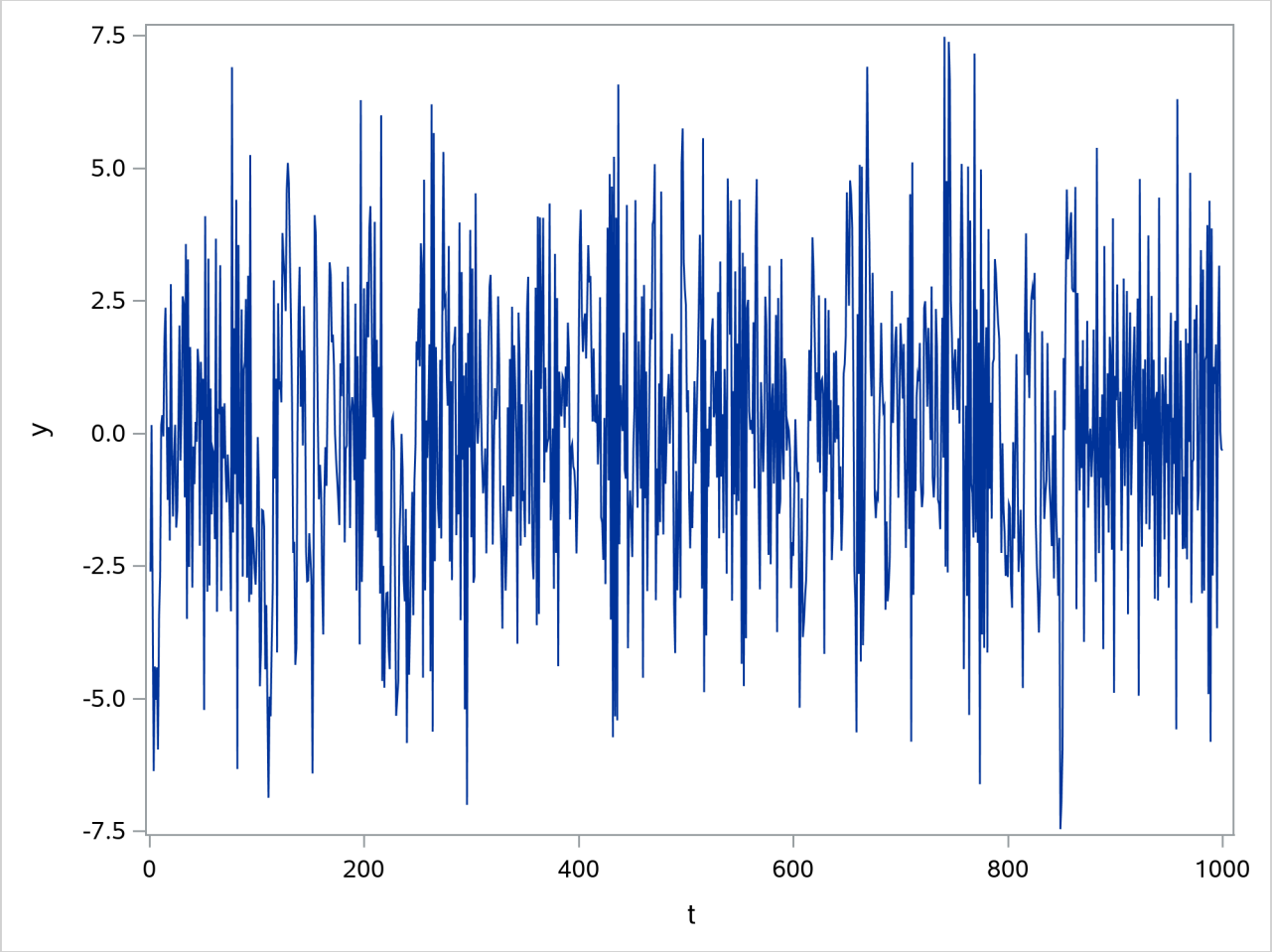
```
%let pi1 = 0.5;
%let a11 = 0.95;
%let a22 = 0.95;
%let ar1_1_1_1 = 0.8;
%let cov1_11 = 2.56;
%let ar2_1_1_1 = -0.7;
```

```
%let cov2_11 = 4;
%let T = 1000;
%let seed = 1234;

data rsardgp;
   retain cd1_11 cd2_11;
   ylag = 0;
   do t = 1 to &T.;
      if(t=1) then do;
         /* initial probability distribution */
         p = &pi1.;
         /* Cholesky decomposition of COV1 */
         cd1_11 = sqrt(&cov1_11.);
         /* Cholesky decomposition of COV2 */
         cd2_11 = sqrt(&cov2_11.);
      end;
      else do;
         /* transition probability matrix */
         if(lags=1) then p = &a11.;
         else p = 1-&a22.;
      end;
      u = uniform(&seed.);
      if(u<=p) then s=1;
      else s = 2;
      e = normal(&seed.);
      if(s=1) then do;
         /* y ~ N(beta1*ylag, Sigma1) at state 1 */
         y = &ar1_1_1_1 * ylag + cd1_11*e;
      end;
      else do;
         /* y ~ N(beta2*ylag, Sigma2) at state 2 */
         y = &ar2_1_1_1 * ylag + cd2_11*e;
      end;
      output;
      lags = s;
      ylag = y;
   end;
run;

data rsar;
   set rsardgp;
   keep t y;
run;
```

The following statements create the series plot in Figure 20.47. It is difficult to tell directly from the plot that the series are from two different regimes.

```
proc sgplot data=rsar;
   series y=y x=t;
run;
```

Figure 20.47 Series Plot of Data Points



The following code uploads the data table and estimates the regime-switching autoregression model:

```
data mylib.rsar; set rsar; run;
proc hmm data=mylib.rsar;
   id time=t;
   model y / type=ar noint ylag=1 nstate=2 method=ml;
   optimize algorithm=activeset printLevel=3 printIterFreq=1;
   learn out=mylib.mylearn;
   filter out=mylib.myfilter;
   smooth out=mylib.mysmooth;
   decode out=mylib.mydecode;
   evaluate out=mylib.myeval;
run;
```

These statements assume that your engine libref is named mylib, but you can substitute any appropriately defined engine libref.

The estimates of all parameters are shown in Figure 20.48, which displays columns for parameter name, estimate value, standard error, t value, and p-value. The parameter estimates are very close to the true parameter values that are used in the data generating process.

Figure 20.48 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.937990	0.013985	67.07	<.0001
TPM1_2	0.062010	0.013985	4.43	<.0001
TPM2_1	0.056595	0.013425	4.22	<.0001
TPM2_2	0.943405	0.013425	70.27	<.0001
AR1_1_1_1	0.759087	0.032724	23.20	<.0001
AR2_1_1_1	-0.630710	0.037261	-16.93	<.0001
COV1_1_1	2.687805	0.198060	13.57	<.0001
COV2_1_1	4.225692	0.280319	15.07	<.0001

The accuracy of classification through the regime-switching autoregression model is calculated by the following statements:

```
data rsarCheck;
   merge rsardgp(in=a) mylib.mydecode(in=b);
   by t;
   if(s=state) then do; correct=1; ds = s; end;
   else do; correct=0; ds = -s; end;
   if(a=b);
   keep t y s correct ds;
run;

%let qFlipState = -1;
data rsarAccuracy;
   set rsarCheck;
   retain count 0 correctCount 0;
   count = count + 1;
   correctCount = correctCount + correct;
   if(count>&T.-0.5) then do;
      accuracy = correctCount / count;
      if(accuracy<0.5) then call symputx("qFlipState",1,'G');
      else call symputx("qFlipState",0,'G');
      if(accuracy<0.5) then accuracy = 1 - accuracy;
      output;
   end;
   keep accuracy;
run;

data rsarCheck;
   set rsarCheck;
   if(&qFlipState.=1) then do;
      ds = -ds;
      correct = 1 - correct;
   end;
run;

proc print data = rsarAccuracy noobs; run;
```

The accuracy of the regime-switching autoregression model, which is shown in Figure 20.49, is more than 92%. This means that the regime-switching autoregression model successfully distinguishes the two regimes.

Figure 20.49 Accuracy of Decoding by the Regime-Switching Autoregression Model

accuracy
0.927

The following statements create the scatter plot in Figure 20.50 to show how well the data points are classified:

```
proc sgplot data=rsarCheck;
   scatter y=y x=t / group=ds;
run;
```

Figure 20.50 Scatter Plot of Data Points

Finite Hidden Markov Model

Let $\mathbf{Y}_t, t = 1, \ldots, T$, denote a random discrete variable whose sample space is a finite set of nominal classes. These values from this sample space can be represented by the finite integers $1, \ldots, M$. $\mathbf{Y}_t$ can be modeled in a categorical distribution,

$$\mathbf{Y}_t | \mathbf{S}_t = i \sim C(b_{i1}, \ldots, b_{iM})$$

where the category probability b_{ij} is the probability of observing $Y_t = j, j = 1, \ldots, M$, conditional on the latent variable $\mathbf{S}_t = i, i = 1, \ldots, K$. The $K \times M$ category probability matrix (CPM) $\mathbf{B} \equiv (b_{ij})$ is used to define the state-dependent categorical distribution of observable variables. The variable $\mathbf{S}_t$ is the so-called state and follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \ldots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(.|.)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π, of the first state, $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \ldots, K\}$$

Because the variable $\mathbf{S}_t$ is unobservable and follows a Markov chain and $\mathbf{Y}_t$ is from a finite support, the model is called the finite hidden Markov model (finite HMM). The finite HMM is sometimes described as $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Consider a standard die with six faces, numbered 1 through 6. A fair die has an equal probability of displaying 1 to 6; that is, $\mathbf{Y}_t \sim C(1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$. In order to tell whether a die is fair, checking whether the opportunity of displaying each number is equal might not be enough.

The following data record the results from the roll of a die 1,000 times:

```
data mylib.finite1;
   input t  y ;
datalines;
1   6
2   3
3   6
4   5
5   1
6   2
7   1
8   6

   ... more lines ...
```

```
993 2
994 3
995 6
996 6
997 5
998 5
999 2
1000    6
;
```

The following statements estimate the unistate finite HMM:

```
proc hmm data=mylib.finite1;
   id time=t;
   model y / type=finite nstate=1 nCategory=6;
   estimate outall=mylib.myEstimate;
run;
```

The number of observations and model information are shown in Figure 20.51.

Figure 20.51 Number of Observations and Model Information

Observation Information	
Number of Observations	1000
Number of Missing Observations	0

Model Information	
Type of Model	Finite HMM
Stationary	Yes
Number of States	1
Number of Dependent Variables	1

The estimates of the initial state probability vector (ISPV) and the transition probability matrix are trivial. The category probability matrix is shown in Figure 20.52, which shows the probabilities of occurrence of each number.

Figure 20.52 Estimates of Category Probability Matrix

Estimated Category Probability Matrix						
	Category					
State	1	2	3	4	5	6
1	0.16300	0.15400	0.16600	0.16700	0.17400	0.17600

The estimates of all parameters are shown in Figure 20.53, which displays columns for parameter name, estimated value, standard error, t value, and p-value.

Figure 20.53 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	1.000000	0		
CPM1_1	0.163000	0.011680	13.96	<.0001
CPM1_2	0.154000	0.011414	13.49	<.0001
CPM1_3	0.166000	0.011766	14.11	<.0001
CPM1_4	0.167000	0.011795	14.16	<.0001
CPM1_5	0.174000	0.011988	14.51	<.0001
CPM1_6	0.176000	0.012043	14.61	<.0001

The following statements run a Wald test under the null hypothesis that all category probabilities are equal to 1/6:

```
proc iml;
  use mylib.myEstimate;
  read all var {Estimate cov_2 cov_3 cov_4 cov_5 cov_6} into m;
  close mylib.myEstimate;

cov = m[2:6,2:6];
invcov = inv(cov);
v = m[2:6,1];
c = 1/6 // 1/6 // 1/6 // 1/6 // 1/6 ;
v1 = v-c;

stat = v1`*invcov *v1;
df = 5;
pvalue = 1- cdf('CHISQ',stat,df);
varname={"stat" "df" "pvalue"};
create table var varname;
append;
close table;
quit;

proc print data=table noobs label;
   label stat="Stat" df = "DF" pvalue = "Pr > ChiSq";
run;
```

The *p*-value shown in Figure 20.54 is very large, which means that the test cannot reject the null hypothesis that the category probabilities are equal to 1/6 at the 10% significance level. The die looks fair.

Figure 20.54 The Wald Test

Stat	DF	Pr > ChiSq
1.93169	5	0.85851

The log likelihood and information criteria are shown in Figure 20.55.

Figure 20.55 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-1790.807551
AIC	3593.6151015
AICC	3593.6996936
BIC	3623.0616332
HQC	3604.8068383

The following code estimates a bistate finite HMM:

```
proc hmm data=mylib.finite1;
   id time=t;
   model y / type=finite nstate=2 nCategory=6;
   score outmodel=mylib.myScore;
run;
```

The number of observations and model information are shown in Figure 20.56.

Figure 20.56 Number of Observations and Model Information

Observation Information	
Number of Observations	1000
Number of Missing Observations	0

Model Information	
Type of Model	Finite HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	1

The estimates of the ISPV are shown in Figure 20.57.

Figure 20.57 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	**Estimation**
1	0.52660
2	0.47340

The estimates of the TPM are shown in Figure 20.58.

Figure 20.58 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	**1**	**2**
1	0.74650	0.25350
2	0.28200	0.71800

The estimates of the CPM are shown in Figure 20.59.

Figure 20.59 Estimates of Category Probability Matrix

Estimated Category Probability Matrix

	Category					
State	1	2	3	4	5	6
1	0.00000	0.01525	0.01435	0.31762	0.31804	0.33474
2	0.34372	0.30784	0.33414	0.00000	0.01430	0.00000

The estimates of all parameters are shown in Figure 20.60, which displays columns for parameter name, estimated value, standard error, t value, and p-value.

Figure 20.60 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.746496	0.027822	26.83	<.0001
TPM1_2	0.253504	0.027822	9.11	<.0001
TPM2_1	0.281997	0.025626	11.00	<.0001
TPM2_2	0.718003	0.025626	28.02	<.0001
CPM1_1	9.9114962E-9	0.000017508	0.00	0.9995
CPM1_2	0.015249	0.017745	0.86	0.3904
CPM1_3	0.014352	0.017406	0.82	0.4098
CPM1_4	0.317621	0.022761	13.95	<.0001
CPM1_5	0.318039	0.024580	12.94	<.0001
CPM1_6	0.334739	0.023261	14.39	<.0001
CPM2_1	0.343724	0.025061	13.72	<.0001
CPM2_2	0.307839	0.025495	12.07	<.0001
CPM2_3	0.334138	0.025467	13.12	<.0001
CPM2_4	1.799124E-10	0.000002004	0.00	0.9999
CPM2_5	0.014298	0.018616	0.77	0.4426
CPM2_6	4.390824E-10	0.000004237	0.00	0.9999

In Figure 20.60, you see that the category probabilities for small numbers (1, 2, 3) in the first state and the category probabilities for large numbers (4, 5, 6) in the second state are not significant at the 10% significance level. This indicates that the small numbers appear only in one state and the large numbers appear only in the other state.

The log likelihood and information criteria are shown in Figure 20.61.

Figure 20.61 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-1697.866801
AIC	3423.7336013
AICC	3424.1599972
BIC	3492.4421752
HQC	3449.8476538

All the information criteria from the bistate finite HMM are smaller than those from the unistate finite HMM. This indicates that the bistate finite HMM is a better model. The tristate finite HMM, which is not shown here, is also beaten by the bistate finite HMM for this case.

The casino game sic bo involves whether you will get a small or big number when you roll a die. A small bet is when you predict that the die will display a 1, 2, or 3. A big bet is when you predict that the die will display a 4, 5, or 6. If your guess is correct, you get a reward; otherwise you lose the bet.

You can use PROC HMM to test the bistate finite HMM's ability to forecast the result of this game. The following data record the results from the roll of a die 100 times:

```
data mylib.finite2;
   input t  y ;
datalines;
1   6
2   4
3   6
4   1
5   1
6   1
7   1
8   4

   ... more lines ...

93  6
94  6
95  5
96  2
97  1
98  6
99  4
100 6
;
```

The following statements perform one-step online (continuous) forecasting of betting big with the estimated parameters from the preceding PROC HMM statements and compare the prediction with the finite2 data set. If the forecasting is correct, you gain one unit. Otherwise you lose one unit.

```
proc hmm data=mylib.finite2;
   id time=t;
   score inmodel=mylib.myScore;
   forecast out=mylib.myforecast online;
run;
data mylib.forecast;
   set mylib.myforecast;
   if (Category1+Category2+Category3>0.5) then bet = 0;
   else bet = 1;
run;
data mylib.forecast;
   set mylib.forecast;
   t=t+step; *so t equals the time when forecast is on;
 run;
```

```
data mylib.finite2;
   set mylib.finite2;
 if(y<=3) then big=0;
 else big=1;
run;
data fnHmmCheck;
   merge mylib.finite2(in=a) mylib.forecast(in=b);
   by t;
run;
data fnHmmCheck;
   set fnHmmCheck;
   *for first round bet according to inital probability;
   if(t=1) then bet=1;
   if(big=bet) then gain=1;
   else gain=-1;
   keep t y gain;
run;
data totalgain;
   set fnHmmCheck;
   Totalgain + gain;
   if (t=100) then do
      Zscore = Totalgain/10;
      pvalue = 2*(1- cdf('NORMAL',Zscore));
      output totalgain;
   end;
   keep Totalgain Zscore pvalue;
run;
proc print data = totalgain noobs label;
   label totalgain = "Total gain" Zscore="Z-score" pvalue="Pr > |z|";
run;
```

The total gain from rolling the die 100 times is shown in Figure 20.62. Under the null hypothesis that the die is fair, according to the central limit theorem the expected return of betting 100 times is 0 with a standard error of 10 units. The values shown in Figure 20.62 indicate that the null hypothesis is rejected even at the 1% significance level. This means that the finite HMM successfully detects the loaded die.

Figure 20.62 Gain of Forecasting by the Finite HMM

Total gain	Z-score	Pr > |z|
48	4.8	.000001587

The data generating process behind the observed data sets is as follows:

$$\mathbf{Y}_t \sim \begin{cases} C(1/3, 1/3, 1/3, 0, 0, 0) & \text{if } \mathbf{S}_t = 1 \\ C(0, 0, 0, 1/3, 1/3, 1/3) & \text{if } \mathbf{S}_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{pmatrix}$$

The following statements simulate the original time series data for this example:

```
%let pi1 = 0.5;*initial probabilty in state 1;
%let a11 = 0.75;
%let a22 = 0.75;

%macro simData(name, T, seed);
data mylib.&name.(keep = t y);
  p = &pi1.;
  call streaminit(&seed);
  do t = 1 to &T.;
   if(t=1) then
      /* initial probability distribution */
     p = &pi1.;
   else
      /* transition probability matrix */
     if(lagx=1) then p = &a11.;
     else p = 1-&a22.;
     u = rand("Uniform");
     if(u<=p) then x=1;
     *Using the random value u to control regimes, set x=number of regimes;
     else x = 2;
     if(x=1) then
     do;
      /* at state 1 */
     y = rand("Table", 1/3, 1/3, 1/3, 0 , 0 , 0);
   end;
   else do;
      /* at state 2 */
     y = rand("Table", 0 , 0 , 0 , 1/3, 1/3, 1/3);
   end;
   output;
   lagx=x;
end;
run;
%mend;

%simData(finite1,1000, 3);
%simData(finite2,100, 4);
```

PROC HMM assumes that values of the response variables are consecutive natural numbers that start from 1. If this is not the case, you might need to preprocess the input data set before running PROC HMM. You can use the SAS macro %FiniteHMM for automatic preprocessing and postprocessing. For more information about %FiniteHMM, see the section "The %FiniteHMM Macro" on page 1240.

Poisson Hidden Markov Model

Let $y_t, t = 1, \ldots, T$, denote a discrete time-series random variable whose sample space is a set of nonnegative integers.

The variable y_t is said to follow a Poisson HMM distribution, with the parameter $\lambda_{s_t} > 0$, if it follows a Poisson distribution in each state S_t. The conditional probability distribution is denoted as $y_t|\mathbf{S}_t = i \sim \text{Pois}(\lambda_{\mathbf{S}_t})$, where i is a positive integer.

The variable $\mathbf{S}_t$ is the so-called state and follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t|\mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \ldots, \mathbf{S}_1) = p(\mathbf{S}_t|\mathbf{S}_{t-1})$$

where $p(.|.)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j|\mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π, of the first state, $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \ldots, K\}$$

Because the state variable $\mathbf{S}_t$ is unobservable and follows a Markov chain, and because y_t is from a Poisson distribution that depends on a different state, the model is called the Poisson hidden Markov model (Poisson HMM).

Consider a univariate Poisson HMM model $y_t \sim \text{Pois}(\lambda_{s_t})$ that has two regimes, $\lambda_1 = 5$ and $\lambda_2 = 10$.

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

```
%let pi1 = 0.7;
%let a11 = 0.9;
%let a22 = 0.8;
%let lambda1 = 5;
%let lambda2 = 10;

%macro dgpPsHMM(table,T,nSections,seed);
data &table;
   call streaminit(&seed.);
   secLen = ceil(&T./&nSections.);
   do t = 1 to &T.;
      sec = floor((t-1)/secLen)+1;
      if(t=1) then do;
```

```
      /* initial probability distribution */
         p = &pi1.;
      end;
      else do;
      /* transition probability matrix */
      if(lags=1) then p = &a11.;
      else p = 1-&a22.;
      end;
      u = rand("Uniform");
      if(u<=p) then s=1;
      else s = 2;
      if(s=1) then do;
      /* at state 1 */
         y = rand("Poisson",&lambda1.);
      end;
      else do;
      /* at state 2 */
         y = rand("Poisson",&lambda2.);
      end;
      output;
      lags = s;
      end;
   run;
   proc sort data=&table.; by sec t; run;
   %mend;

   %let T = 1000 ;
   %let nSections = 2;
   %let seed = 1234;
   %dgpPsHMM(hmmpsdt,&T.,&nSections.,&seed.);

   data mylib.hmmpsdt;
      set hmmpsdt;
   run;
```

The following code estimates a bistate Poisson HMM:

```
   proc hmm data=mylib.hmmpsdt;
      id time=t;
      model y / type=Poisson nstate=2;
   run;
```

The number of observations and model information are shown in Figure 20.63.

Figure 20.63 Number of Observations and Model Information

Observation Information	
Number of Observations	1000
Number of Missing Observations	0

Model Information	
Type of Model	Poisson HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	1

The estimates of the ISPV are shown in Figure 20.64.

Figure 20.64 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	**Estimation**
1	0.71828
2	0.28172

The estimates of the TPM are shown in Figure 20.65.

Figure 20.65 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	**1**	**2**
1	0.93328	0.06672
2	0.17010	0.82990

The estimates of the lambda are shown in Figure 20.66.

Figure 20.66 Estimates of Lambda of Poisson Distribution

Lambda	
State	**Estimation**
1	5.15141
2	9.84789

The estimates of all parameters are shown in Figure 20.67, which displays columns for parameter name, estimated value, standard error, t value, and p-value.

Figure 20.67 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.933281	0.014084	66.26	<.0001
TPM1_2	0.066719	0.014084	4.74	<.0001
TPM2_1	0.170104	0.031678	5.37	<.0001
TPM2_2	0.829896	0.031678	26.20	<.0001
LAMBDA1	5.151408	0.108184	47.62	<.0001
LAMBDA2	9.847886	0.254877	38.64	<.0001

Syntax: HMM Procedure

PROC HMM DATA=*libref.data-table* < *options* > ;
  DECODE < *option* > ;
  DISPLAY < *table-list* > < / *options* > ;
  DISPLAYOUT *table-spec-list* < / *options* > ;
  ESTIMATE < *options* > ;
  EVALUATE < *option* > ;
  FILTER < *option* > ;
  FORECAST < *options* > ;
  ID TIME= *variable* < **SECTION=** *variable* > ;
  INITIAL *equation* < , . . . , *equation* > ;
  MODEL *dependents* < = *regressors* > < / *options* > ;
  OPTIMIZE < *options* > ;
  PRIOR *distribution* < , . . . , *distribution* > ;
  SCORE < *options* > ;
  SMOOTH < *option* > ;
  STORE < **OUT=** > *libref.data-table* ;

Functional Summary

The statements and options available in the HMM procedure are summarized in Table 20.1.

Table 20.1 Functional Summary

Description	Statement	Option
Input Data Table Options		
Specifies the input data table	PROC HMM	DATA=
Specifies the input data table for parameter estimation	ESTIMATE	IN=
Specifies the input data table for model information	SCORE	INMODEL=

Table 20.1 *continued*

Description	Statement	Option
Output Data Table Options		
Writes the log likelihood and information criteria to the specified output data table	PROC HMM	OUTSTAT=
Writes the parameter estimates to the specified output data table	ESTIMATE	OUT=
Writes the parameter estimates and their standard errors and covariance matrix to the specified output data table	ESTIMATE	OUTALL=
Writes the evaluation results to the specified output data table	EVALUATE	OUT=
Writes the decoding results to the specified output data table	DECODE	OUT=
Writes the filtering results to the specified output data table	FILTER	OUT=
Writes the forecasting results to the specified output data table	FORECAST	OUT=
Writes the state distributions, means, standard errors, confidence intervals, medians, and covariance matrices of forecasts for dependent variables to the specified output data table	FORECAST	OUTALL=
Writes the model information to the specified output data table	SCORE	OUTMODEL=
Writes the smoothing results to the specified output data table	SMOOTH	OUT=
Writes the model information to the specified output data table that can be processed by the ASTORE procedure	STORE	OUT=
ID Variable		
Specifies the variable that identifies the time or sequence	ID	TIME=
Specifies the variable that identifies the section	ID	SECTION=
Model Options		
Specifies the mean form in a regime-switching autoregression model	MODEL	ARMEAN=
Estimates the initial state probability vector (ISPV)	MODEL	ESTISPV
Specifies the estimation method	MODEL	METHOD=
Specifies the number of categories in a state for a finite HMM	MODEL	NCATEGORY
Specifies the number of components in a state in a Gaussian mixture HMM (GM HMM)	MODEL	NCOMPONENT
Suppresses the present values of the exogenous variables	MODEL	NOCURRENTX

Table 20.1 *continued*

Description	Statement	Option
Suppresses the constant (intercept) parameters	MODEL	NOINT
Specifies the number of seasonal periods	MODEL	NSEASON=
Specifies the number or the range of numbers of hidden states	MODEL	NSTATE=
Centers seasonal dummies	MODEL	SCENTER
Specifies the state-independent parameters	MODEL	STATEINDEPENDENT=
Specifies the degree of deterministic time trend	MODEL	TREND=
Specifies the type of model of interest	MODEL	TYPE=
Specifies the lags of exogenous (independent) variables	MODEL	XLAG=
Specifies the order or the range of orders of the autoregressive process	MODEL	YLAG=
Optimization Options		
Specifies a nonnegative integer to use as the seed for generating random number sequences	PROC HMM	SEED=
Specifies the initial parameter values for optimization	INITIAL	
Specifies the optimization algorithm	OPTIMIZE	ALGORITHM=
Specifies whether to calculate the covariance of parameters	OPTIMIZE	COVOFPARM=
Specifies the maximum number of iterations in the optimization process	OPTIMIZE	MAXITER=
Specifies the maximum time (in seconds) allowed for optimization	OPTIMIZE	MAXTIME=
Specifies the print iteration frequency	OPTIMIZE	PRINTITERFREQ=
Specifies the print level	OPTIMIZE	PRINTLEVEL=
Specifies the priors for parameters	PRIOR	
Optimization Suboptions for the Active-Set or Interior Point Algorithm		
Specifies the range from which each variable can take values during the sampling process in multistart mode	OPTIMIZE	MSRANGE=
Specifies whether to enable multistart mode	OPTIMIZE	MULTISTART=
Specifies the upper limit on the magnitude of the objective value	OPTIMIZE	OBJLIMIT=
Specifies the tolerance for infeasibility	OPTIMIZE	FEASTOL=
Specifies the tolerance for the optimality error	OPTIMIZE	OPTTOL=
Specifies whether the optimizer should return only a solution that is locally optimal	OPTIMIZE	SOLTYPE=
Optimization Suboption for the Expectation-Maximization (EM) Algorithm		
Specifies the minimum number of iterations in the optimization process	OPTIMIZE	MINITER=

Table 20.1 *continued*

Description	Statement	Option
Executes the EM algorithm in a single thread on a single node	OPTIMIZE	SINGLE
Optimization Suboptions for the Stochastic Gradient Descent (SGD) Algorithm		
Specifies the annealing rate	OPTIMIZE	ANNEALINGRATE=
Specifies the learning rate	OPTIMIZE	LEARNINGRATE=
Specifies the size of the minibatches to use	OPTIMIZE	MINIBATCHSIZE=
Specifies the value for momentum	OPTIMIZE	MOMENTUM=
Suppresses the adaptive gradient descent algorithm	OPTIMIZE	NONADUPDATE
Specifies the value of adaptive rate	OPTIMIZE	ADAPTIVEDECAY=
Requests that computational threads share a common weight vector and that they update the weight vector without race conditions	OPTIMIZE	USELOCKING=
Specifies the length of a chain in a minibatch	OPTIMIZE	CHAIN=(LENGTH=)
Specifies the buffer length on the left side of a chain in a minibatch	OPTIMIZE	CHAIN=(BUFFER=)
Optimization Suboptions for the Distributed Multistart Option		
Specifies the number of CAS subsessions that are dedicated to the function evaluations	OPTIMIZE	FESESSIONS=
Specifies the number of computing nodes in each CAS subsession that are dedicated to the function evaluations	OPTIMIZE	FESESSNODES=
Specifies the maximum number of starting points to be used by the multistart algorithm	OPTIMIZE	MAXSTARTS=
Specifies the ratio of the total number of sample points to be generated to the value of the MAXSTARTS= option	OPTIMIZE	SAMPSELRATIO=
Specifies the number of computing nodes in the CAS session dedicated to the multistart algorithm	OPTIMIZE	SESSNODES=
Forecasting Options		
Specifies a nonnegative integer to use as the seed for generating random number sequences	PROC HMM	SEED=
Specifies the size of confidence limits for forecasting	FORECAST	ALPHA=
Starts forecasting before the end of the input data	FORECAST	BACK=
Specifies how many periods to forecast	FORECAST	LEAD=
Specifies the number of paths to simulate in the forecasting of a regime-switching autoregression model	FORECAST	NSIM=
Specifies whether to perform forecasts after each observation	FORECAST	ONLINE

Table 20.1 *continued*

Description	Statement	Option
Output Control Options		
Specifies the order in which to label the states (and components)	PROC HMM	LABELSWITCH=
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as output tables	DISPLAYOUT	

PROC HMM Statement

PROC HMM DATA=*libref.data-table* < *options* > **;**

The PROC HMM statement invokes the HMM procedure. You must specify the following *option*:

DATA=*libref.data-table*
names the input data table for PROC HMM to use. *libref.data-table* is a two-level name, where

libref refers to a collection of information that is defined in the LIBNAME statement and includes the library, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *libref*, see the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

data-table specifies the name of the input data table.

You can also specify the following *options*:

LABELSWITCH=(*options***)**
specifies the order in which to label the states (and components). The label switching problem is a common issue for HMMs because any two HMMs are mathematically equivalent if the only difference between two models is that the states are labeled in different ways. For example, consider the following two univariate bistate Gaussian HMMs, which are described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, μ_1 and Σ_1 are mean and covariance parameters for state 1, and μ_2 and Σ_2 are mean and covariance parameters for state 2:

$$\text{model 1: } \pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{pmatrix}, \mu_1 = 1, \Sigma_1 = 1, \mu_2 = -1, \Sigma_2 = 4$$

$$\text{model 2: } \pi = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{pmatrix}, \mu_1 = -1, \Sigma_1 = 4, \mu_2 = 1, \Sigma_2 = 1$$

In theory, these two models are equivalent, and just the state labels "1" and "2" are switched. When the labels are switched, it is inconvenient to compare any state-related results, such as filtered or smoothed probabilities or decoded path. The LABELSWITCH= option helps you define the order of the state (and component) labels. You can specify the following *options*:

OUT=*libref.data-table*

writes the changes of state labels (and component labels in the case of GM HMMs) to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

REORDER=(*number-list***)**

specifies the new order of the state labels, where the *number-list* directly specifies the new order of the states. The *number-list* must be a permutation of numbers 1 to K, where K is the number of states; that is, the size of the *number-list* must be K, and each element in the *number-list* must be unique and between 1 and K. For example, REORDER=(3 1 2) implies that the original state 1 becomes state 3, state 2 becomes state 1, and state 3 becomes state 2. If you specify this option, the SORT= option is ignored.

REORDERCOMPONENT=(*number-list***)**

specifies the new order of the component labels in the case of GM HMMs, where the *number-list* directly specifies the new order of the components for each state. For a K-state M-component GM HMM, the size of the *number-list*, $\{n_{11}, \ldots, n_{1M}, \ldots, n_{K1}, \ldots, n_{KM}\}$, must be KM, and each sublist $\{n_{i1}, \ldots, n_{iM}\}, i = 1, \ldots, K$, must be a permutation of numbers 1 to M—that is, each element in the sublist $\{n_{i1}, \ldots, n_{iM}\}$ must be unique and between 1 and M. For example, REORDERCOMPONENT=(3 2 1 2 1 3) for a bistate tricomponent GM HMM implies that the original components 1, 2, and 3 for the original state 1 become components 3, 2, and 1, respectively, for the original state 1, and the original components 1, 2, and 3 for the original state 2 become components 2, 1, and 3, respectively, for the original state 2. If you specify this option, the labels for components in each state are not sorted.

SORT=*type* **<(***name-list***)>**

specifies whether to sort the labels of states (and components) and in which type of order. You can specify the following *type*s:

ASC <(*name-list*)> sorts the labels in ascending order. You can add a *name-list* in parentheses in order to sort the labels of states (and components). The possible names that can appear in the *name-list* are ISPV, TPM, MCP, CPM, LAMBDA, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV; the meanings of those names are defined in Table 20.4. Any parameters that are not in the model are ignored.

DESC <(*name-list*)> sorts the labels in descending order. You can add a *name-list* in parentheses in order to sort the labels of states (and components). The possible names that can appear in the *name-list* are ISPV, TPM, MCP, CPM, LAMBDA, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV; the meanings of those names are defined in Table 20.4. Any parameters that are not in the model are ignored.

NONE does not sort the labels.

By default, if the INMODEL= option is not specified in the SCORE statement, SORT=ASC(SIGMA); otherwise, SORT=NONE. For example, when the ESTISPV option is not specified, the ISPV is not in the stationary HMM; if you include ISPV in the *name-list*,

it is ignored. For a GM HMM, the labels of components for each state are first sorted by the parameters state by state if the REORDERCOMPONENT= option is not specified; then the labels of states are sorted by the parameters that are stacked in the new order of components.

For more details and examples, see the section "Label Switching Problem" on page 1283.

OUTSTAT=*libref.data-table*
writes the log likelihood and information criteria to the specified output data table. The information criteria include Akaike's information criterion (AIC), the corrected Akaike's information criterion (AICC), the Bayesian information criterion (BIC, also referred to as the Schwarz Bayesian criterion, SBC), and the Hannan-Quinn criterion (HQC). *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

SEED=*number*
specifies a nonnegative integer to use as the seed for generating random number sequences. When the seed value is 0, random number sequences are generated by using the time of day from the computer's clock. Seed values greater than 0 generate reproducible random number sequences. By default, SEED=1.

DECODE Statement

DECODE < *option* > **;**

The DECODE statement solves the decoding problem by finding the best possible state path for the observations. You can specify the following *option*:

OUT=*libref.data-table*
writes the decoding results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

DISPLAY Statement

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

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

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE

statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

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

The table names that you can specify are listed in the section "ODS Table Names" on page 1304. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

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

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

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

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

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

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

TRACE
displays the display table names, labels, and paths.

DISPLAYOUT Statement

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

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

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

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

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

The ODS table names that you can specify are listed in the section "ODS Table Names" on page 1304. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the DISPLAY statement. The DISPLAYOUT statement does not support regular expressions.

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

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

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

REPEATED
replicates all output tables on all nodes.

ESTIMATE Statement

ESTIMATE < *options* > **;**

LEARN < *options* > **;**

The ESTIMATE statement estimates the parameters. You can specify the following *options*:

IN=*libref.data-table*
specifies the input data table for parameter estimation. In general, the data table is an output data table that is produced by the OUT= option in the ESTIMATE statement in a previous call of the HMM procedure. To estimate each model that is specified in the MODEL statement, PROC HMM first checks whether an INITIAL statement is specified for the model. If the INITIAL statement exists for the model, the initial values that are specified in the INITIAL statement are used; otherwise, PROC HMM searches the IN= data table to see whether it contains a model that has the same model specification. If the IN= data table contains a matched model, its values are used as the initial values for the model that is specified in the MODEL statement. **NOTE:** When PROC HMM checks for matched models, the ESTISPV model option, which calculates the ISPV parameter, presents a special case. If the current model specification does not include the ESTISPV option, and the IN= table is a parameter estimate output table from a previous model specification that does include the ESTISPV option, then values in the IN= table are not used as initial values, even if all other specifications of the two models are the same.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

OUT=*libref.data-table*
writes the parameter estimates to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

OUTALL=*libref.data-table*
writes the parameter estimates and their standard errors and covariance matrix to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

EVALUATE Statement

EVALUATE < *option* > **;**

The EVALUATE statement evaluates the model by calculating the log likelihood. You can specify the following *option*:

OUT=*libref.data-table*
writes the evaluation results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

FILTER Statement

FILTER < *option* > **;**

The FILTER statement solves the filtering problem, which is to find the probability distribution of the current state, conditional on the observations that precede and include the current observation. You can specify the following *option*:

OUT=*libref.data-table*
writes the filtering results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

FORECAST Statement

FORECAST < *options* > **;**

The FORECAST statement finds the probability distribution of the future states and the dependent variables. You can specify the following *options*:

ALPHA=*number*
specifies the forecast confidence limit size, where *number* is between 0 and 1. When you specify this option, the upper and lower confidence limits define the $100\alpha\%$ confidence interval. By default, ALPHA=0.95, which produces 95% confidence intervals.

BACK=*number*
specifies the number of observations before the end of the data at which the multistep forecasts begin, where *number* must be less than or equal to the number of observations minus the number of lagged regressors in the model. By default, BACK=0, which means that the forecasts start at the end of the available data.

LEAD=*number*
specifies the number of multistep forecast values to compute, where *number* must be a nonnegative integer. By default, LEAD=1, which means that a one-step-ahead forecast is performed.

METHOD=ANALYTIC | SIMULATION
specifies the forecasting method. The SIMULATION option works only when you specify the TYPE=AR option in the MODEL statement. You can specify the following values:

ANALYTIC specifies the analytical method of forecasting.

SIMULATION specifies the simulation method of forecasting for autoregressive HMM.

By default, METHOD=SIMULATION for autoregressive HMM and METHOD=ANALYTIC for other types of HMM.

NSIM=*number*
specifies the number of paths to be simulated in the forecasting of a regime-switching autoregression model, where *number* must be a positive integer. By default, NSIM=100.

ONLINE
requests that the (multistep) forecasts be done after each observation. When you specify this option, the BACK= option is ignored.

OUT=*libref.data-table*
writes the forecasting results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

OUTALL=*libref.data-table*
writes the state distributions, means, standard errors, confidence intervals, medians, and covariance matrices of forecasts for dependent variables to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

ID Statement

ID TIME=*variable* < **SECTION=***variable* > **;**

The ID statement identifies observations in the input data table by specifying a variable for the time series data or two variables for the cross-sectional time series data.

You must specify the following option:

TIME=*variable*
specifies the temporal or sequential order of the observations. The *variable* cannot have missing values. If you omit the SECTION= option, the *variable* cannot have duplicate values. If you specify the SECTION= option, the *variable* cannot have duplicate values within a section.

You can also specify the following option:

SECTION=*variable*
identifies the section of each observation for modeling cross-sectional time series data. The *variable* cannot have missing values.

INITIAL Statement

INITIAL *equation* < *, . . . , equation* > **;**

The INITIAL statement specifies the initial parameter values for nonlinear optimization when the maximum likelihood method or the maximum a posteriori method is applied to the estimation of the HMMs (that is, METHOD=ML or METHOD=MAP, respectively, in the MODEL statement). Only one INITIAL statement is allowed. If you specify more than one *equation*, separate them with commas. The INITIAL statement supports only equations; hence, inequality comparison operators (<, <=, >, and >=) and the distribution operator (∼) are not supported in the INITIAL statement. If you omit this statement, the initial values of parameters are determined by the input data.

If you specify the NSTATE=$n_1 : n_2$ option or the YLAG=$m_1 : m_2$ option (or both) in the MODEL statement to estimate multiple models, the INITIAL statement is applied to the first model that corresponds to the specification of NSTATE=n_1 or YLAG=m_1 (or that corresponds to both NSTATE=n_1 and YLAG=m_1 if both options are specified).

To use the INITIAL statement, you need to know the form of the model: different sets of parameters are available for different types of HMMs. Nonlinear equations on parameters are not supported.

The *equation* is in the form of a matrix expression. For more information about the matrix expression, see the section "Matrix Expression" on page 1273.

The initial parameter values are values that solve the specified linear equations. If you do not specify initial values for all parameters, the default initial value for any parameter that is not specified in the INITIAL statement is 0, except for the following:

- The diagonal elements of the **SIGMA** or **COV** parameter matrix are set to ones if the **SIGMA** or **COV** parameter matrix is to be estimated.
- If a row of the **TPM** parameter matrix contains all zeros, each element in the row is set to $1/K$, where K is the number of states; if a row of the **TPM** parameter matrix does not add up to one, the row is normalized by dividing each element in the row by the sum of the row.
- If all elements of the **ISPV** parameter vector are zeros, each element of the **ISPV** parameter vector is set to $1/K$, where K is the number of states; if the sum of the **ISPV** parameter vector is not equal to one, it is normalized by dividing each element of the **ISPV** parameter vector by the sum of the **ISPV** parameter vector.
- The value of the **LAMBDA** parameter vector for each state is set to one if the **LAMBDA** parameter vector is to be estimated.

The following example uses the INITIAL statement for a bivariate three-state Gaussian HMM, which PROC HMM estimates by the maximum likelihood method by default:

```
proc hmm data=One;
   id time = t;
   model y1 y2 / type=gaussian nstate=3;
   initial TPM={0.8 0.1 0.1, 0.2 0.7 0.1, 0.9 0.05 0.05},
           MU={-1 -1, 0 0, 1 1}`,
           SIGMA(1)=4*I(2),
           SIGMA(2)=I(2),
           SIGMA(3)=4*I(2)+2;
run;
```

MODEL Statement

MODEL *dependents* < = *regressors* > < / *options* > **;**

The MODEL statement specifies dependent (endogenous) variables and regressors (the independent or exogenous variables) for the HMM model. Only one MODEL statement is allowed.

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

ARMEAN=ADJUSTED | STANDARD

specifies the mean form in a regime-switching autoregression model. The ARMEAN= option applies only when you specify the TYPE=AR option in the MODEL statement. You can specify the following values:

ADJUSTED specifies the mean-adjusted form of regime-switching autoregression model.

STANDARD specifies the standard regime-switching autoregression model.

By default, METHOD=STANDARD.

ESTISPV
estimates the initial state probability vector (ISPV). By default, the ISPV is fixed to the stationary distribution of the Markov chain. For the finite-state-space and homogeneous HMM, the stationary distribution of the Markov chain is the π such that $\pi' A = \pi'$, where A is the transition probability matrix (TPM); that is, π is the leading left eigenvector of A. If you do not specify this option but you do specify the ISPV function in the INITIAL or PRIOR statement, the HMM procedure issues errors and then stops.

METHOD=MAP | ML
requests the type of estimates to be computed. You can specify the following values:

MAP specifies the maximum a posteriori method.

ML specifies the maximum likelihood method.

By default, METHOD=ML.

NCATEGORY=*number*
specifies the number of categories in a state for a finite HMM, where *number* must be a positive integer. This option applies only when you specify TYPE=FINITE. By default, NCATEGORY=2.

NCOMPONENT=*number*
specifies the number of components in a state for a Gaussian mixture HMM (GM HMM), where *number* must be a positive integer. This option applies only when you specify TYPE=GAUSSIANMIXTURE. By default, NCOMPONENT=1.

NOCURRENTX
suppresses the present values of the exogenous variables. This option applies only when you specify TYPE=AR or TYPE=REG.

NOINT
suppresses the constant (intercept) parameters. This option applies only when you specify TYPE=AR or TYPE=REG.

NSEASON=*s*
specifies the number of seasonal periods (where *s* must be a positive integer) and adds $(s-1)$ seasonal dummies to the regressors. This option applies only when you specify TYPE=AR or TYPE=REG and do not specify the NOINT option.

NSTATE=*number1* **<** **:** *number2* **>**
specifies the number or the range of numbers of hidden states, where *number1* and *number2* must be positive integers. If you specify NSTATE=*number1*, the number of hidden states is *number1*; if you specify NSTATE=*number1*:*number2*, multiple models whose number of hidden states is inclusively between *number1* and *number2* are estimated and inferred. If you specify the same number for both *number1* and *number2*, *number2* is ignored. By default, NSTATE=2.

SCENTER
centers seasonal dummies that are implied by the NSEASON= option. The centered seasonal dummies are generated by $c - \frac{1}{s}$, where c is a seasonal dummy that the NSEASON=*s* option generates. This option applies only when you specify the NSEASON= option and TYPE=AR or TYPE=REG.

STATEINDEPENDENT=(*name-list***)**
specifies the parameters to be estimated independent of state. The names of the independent parameters are entered in the *name-list* in parentheses. The names that can appear in the *name-list* are ISPV, TPM, MCP, CPM, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV. Table 20.4 gives the meanings of these names. This option does not support inputting the component index after the parameter names. For example, it supports COV but not COV(1,1,1). The name ISPV is ignored unless ESTISPV is also specified. Any parameters that are not in the model are ignored. Currently this option does not support ALGORITHM=SGD in the OPTIMIZE statement. The estimates are ordered by the names specified in the *name-list*.

To see how this option works, consider the following univariate bistate Gaussian HMM, which is described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, μ_1 and Σ_1 are the mean and covariance parameters for state 1, and μ_2 and Σ_2 are the mean and covariance parameters for state 2. STATEINDEPENDENT=(SIGMA) estimates the model with $\Sigma_1 = \Sigma_2$. The following statement specifies this option:

```
model y / stateindependent=(sigma);
```

For more details and an example, see "Example 20.3: Analysis of the Business Cycle by Using the Regime-Switching Mean-Adjusted Autoregression Model" on page 1357.

TREND=LINEAR | QUAD
specifies the degree of deterministic time trend that the model includes. You can specify the following values:

LINEAR includes a linear time trend as a regressor.

QUAD includes linear and quadratic time trends as regressors.

This option applies only when you specify TYPE=AR or TYPE=REG.

TYPE=AR | FINITE | GAUSSIAN | GAUSSIANMIXTURE | POISSON | REG
specifies the type of the model of interest to use. You can specify the following values:

AR uses the regime-switching autoregression model (RS-AR, also known as the autoregressive hidden Markov model, AR HMM).

FINITE uses the finite hidden Markov model (finite HMM).

GAUSSIAN uses the Gaussian hidden Markov model (Gaussian HMM).

GAUSSIANMIXTURE uses the Gaussian mixture hidden Markov model (GM HMM).

POISSON uses the Poisson hidden Markov model (Poisson HMM).

REG uses the regime-switching regression model (RS-REG, also known as the regression hidden Markov model, REG HMM).

By default, TYPE=GAUSSIAN.

XLAG=*number*
specifies the lags of exogenous (independent) variables. To exclude the present values of exogenous variables from the model, you must specify the NOCURRENTX option. This option applies only when you specify TYPE=AR or TYPE=REG. By default, XLAG=0.

YLAG=*number1* **<** **:** *number2* **>**
specifies the order or the range of orders of the autoregressive process, where *number1* and *number2* must be nonnegative integers. If you specify YLAG=*number1*, the order of the autoregressive process is *number1*; if you specify YLAG=*number1*:*number2*, multiple models whose order of the autoregressive process is inclusively between *number1* and *number2* are estimated and inferred. If you specify the same number for both *number1* and *number2*, *number2* is ignored. This option applies only when you specify TYPE=AR. By default, YLAG=0.

OPTIMIZE Statement

OPTIMIZE < *options* > **;**

The OPTIMIZE statement optimizes the parameter estimation.

You can specify the following *options*:

ALGORITHM=*type* **<** **(***suboptions***)** **>**
specifies the optimization algorithm to use. You can specify the following *types*:

ACTIVESET **<** **(***suboptions***)** **>**
uses the active set algorithm. You can specify one or more of the following *suboption*s in parentheses:

FEASTOL=*number*
defines the feasible tolerance. The optimizer exits if the constraint violation is less than *number* and the scaled optimality conditions are less than the value of the OPTTOL= suboption. By default, FEASTOL=10^{-6}.

MSRANGE=*number*
defines the range from which each variable can take values during the sampling process in multistart mode, where *number* is a positive number. This option is ignored if MULTISTART=0. This option affects only the sampling process that determines starting points for the local solver. It does not affect the bounds of the original nonlinear optimization problem. By default, MSRANGE=2.

MULTISTART=0 | **1** | **(***suboptions***)**
specifies whether to enable multistart mode or specifies *suboptions* to enable the distributed multistart mode. You can specify the following values:

0 disables multistart mode.

1 solves the problem from multiple starting points, possibly finding a better local optimum as a result. The computing cost in multistart mode might be huge.

By default, MULTISTART=0.

The following *suboptions*, which are specified in parentheses, are supported only in distributed (MPP) mode. If you specify empty parentheses, these *suboptions* are assigned their default values.

FESESSIONS=*number*
specifies the number of CAS subsessions that are dedicated to the function evaluations, where *number* is a positive integer. The default value is based on the setting of the CAS session.

FESESSNODES=*number*
specifies the number of computing nodes in each CAS subsession that are dedicated to the function evaluations, where *number* is a positive integer. The default value is based on the setting of the CAS session.

MAXSTARTS=*number*
specifies the maximum number of starting points to be used by the multistart algorithm, where *number* is a positive integer. By default, MAXSTARTS=100.

SAMPSELRATIO=*number*
specifies the ratio of the total number of sample points to be generated to the value of the MAXSTARTS= option, where *number* is a positive integer. By default, SAMPSELRATIO=20.

SESSNODES=*number*
specifies the number of computing nodes in the CAS session that are dedicated to the multistart algorithm, where *number* is a positive integer. The default value is based on the setting of the CAS session.

OBJLIMIT=*number*
specifies an upper limit on the magnitude of the objective value, where *number* is greater than or equal to 10^8. If OBJLIMIT=M, the algorithm terminates when the objective value becomes less than $-M$ for a minimization problem or when the objective value exceeds M for a maximization problem. The termination of the algorithm implies that either the problem is unbounded or the algorithm diverges. If optimization were allowed to continue, numerical difficulty might be encountered. By default, OBJLIMIT=10^{20}. If *number* is less than 10^8, it is reset to the default value of 10^{20}.

OPTTOL=ϵ
specifies the measure by which the current iterate is determined to be an acceptable approximation of a local optimum, where ϵ is a positive real number. The optimizer determines that the current iterate is a local optimum when the norm of the scaled vector of the optimality conditions is less than ϵ and the true constraint violation is less than the value of the FEASTOL= option. By default, OPTTOL=10^{-6}.

SOLTYPE=0 | 1
specifies the type of solution that the optimizer should return. You can specify the following values:

0 requests that the optimizer return a locally optimal solution, provided that it locates one.

1 requests that the optimizer return the best feasible solution that was found, provided that its objective value is better than that of the locally optimal solution that was found.

By default, SOLTYPE=1.

EM <(*suboption*)>
uses the expectation-maximization (EM) algorithm. This algorithm is supported only for the Gaussian HMM and the maximum likelihood method. You can specify the following *suboption* in parentheses:

MINITER=*number*
specifies the minimum number of iterations in the optimization process, where *number* must be a nonnegative integer. By default, MINITER=0.

SINGLE
executes the EM algorithm in a single thread on a single node. By default, the EM algorithm runs in parallel on multiple threads and nodes if you divide the data into sections and specify the SECTION= option in the ID statement.

INTERIORPOINT <(*suboptions*)>
uses the interior point algorithm. You can specify the same *suboptions* in parentheses that you can specify for ALGORITHM=ACTIVESET.

IPDIRECT <(*suboptions*)>
uses the interior point direct algorithm, in which the use of direct factorizations and other enhancements can reduce both the number of iterations and the CPU time for many problem types. You can specify the same *suboptions* in parentheses that you can specify for ALGORITHM=INTERIORPOINT.

SGD <(*suboptions*)>
uses the stochastic gradient descent algorithm. You can specify one or more of the following *suboptions* in parentheses:

ADAPTIVEDECAY=*number*
specifies the adaptive rate for the gradient descent algorithm when it is used, where the algorithm becomes ADAGRAD if ADAPTIVEDECAY=0 and becomes ADADELTA otherwise. By default, ADAPTIVEDECAY=0.95.

ANNEALINGRATE=*number*

specifies the annealing parameter, β, where *number* must be a nonnegative double. Annealing is a way to automatically reduce the learning rate as SGD progresses, causing smaller steps as SGD approaches a solution. Effectively, this option replaces the learning rate parameter, η, with

$$\eta' = \frac{\eta}{1 + \beta t}$$

where t is the number of iterations that SGD has performed. By default, ANNEALINGRATE=10^{-5}.

CHAIN=*(options)*

specifies the options for the sequence data (which constitute a chain). You can specify the following *options* within parentheses:

LENGTH=*number*

specifies the length of a chain in a minibatch in optimization process, where *number* must be a positive integer. By default, LENGTH=1000.

BUFFER=*number*

specifies the buffer length on the left side of a chain in a minibatch in optimization process, where *number* must be a nonnegative integer. By default, BUFFER=20.

LEARNINGRATE=*number*

specifies the learning rate parameter, η, for SGD, where *number* must be a positive double. New iterates for SGD are found by using

$$\theta_{k+1} = \theta_k - \frac{\eta}{\|I_k\|} \sum_{(x_i, y_i) \in I_k} \nabla L(\theta_k; x_i, y_i)$$

where θ_k is the current weight vector, θ_{k+1} is the new weight vector, I_k is the minibatch that is used during iteration k, and $L(\theta_k; x_i, y_i)$ is the negative log likelihood that is associated with the ith chain of observation. If you see a huge objective value with SGD (especially with a small data table), it is likely that too large a value was specified for this option. By default, LEARNINGRATE=10.

MINIBATCHSIZE=*number*

specifies the size of the minibatches used in SGD, where *number* must be a positive integer. By default, MINIBATCHSIZE=1.

MOMENTUM=*number*

specifies the value for momentum parameter α, where *number* must be greater than or equal to 0 and less than 1. New iterates for SGD that uses the momentum method are found by using

$$\theta_{k+1} = \theta_k + \nabla\theta_k$$
$$\nabla\theta_k = \alpha\nabla\theta_k - \frac{\eta}{\|I_k\|} \sum_{(x_i, y_i) \in I_k} \nabla L(\theta_k; x_i, y_i)$$

where θ_k is the current weight vector, θ_{k+1} is the new weight vector, I_k is the minibatch that is used during iteration k, and $L(\theta_k; x_i, y_i)$ is the negative log likelihood that is associated with the ith chain of observation. By default, MOMENTUM=0.

NONADUPDATE
suppresses the adaptive gradient descent algorithm.

USELOCKING
requests that computational threads share a common weight vector and that they update the weight vector without race conditions. By default, computational threads update a single weight vector simultaneously. This causes intentional race conditions and nondeterministic behavior, but it increases performance significantly.

By default, ALGORITHM=ACTIVESET.

COVOFPARM=YES | NO
specifies whether to calculate the covariance of parameters. You can specify the following values:

YES calculates the covariance of parameters.

NO suppresses the calculation of the covariance of parameters.

By default, COVOFPARM=YES. This option supports only the EM algorithm.

MAXITER=*number*
specifies the maximum number of iterations in the optimization process, where *number* must be a nonnegative integer. By default, MAXITER=128.

MAXTIME=*t*
specifies an upper limit of *t* seconds for the optimization process, including problem generation time and solution time, where *t* can be any positive number. The default value of *t* is the positive number that has the largest absolute value that your operating environment can represent. If you omit this option, the optimizer does not stop on the basis of the amount of time elapsed. This option is not supported for the EM algorithm.

PRINTITERFREQ=*number*
specifies how frequently to output the optimization process history, where *number* must be a nonnegative integer. If PRINTITERFREQ=0, no iterations are output. If *number*=*n* and *n* is positive, the optimization process history is output every *n* iterations. By default, PRINTITERFREQ=0.

PRINTLEVEL=0 | 1 | 2 | 3
specifies the print level. You can specify the following values:

0 suppresses output of the initial parameter values, the initial objective value, the algorithm information, the final parameter values, and the final objective value.

1 outputs the initial parameter values and the initial objective value.

2 outputs the initial parameter values, the initial objective value, and the algorithm information.

3 outputs the initial parameter values, the initial objective value, the algorithm information, the final parameter values, and the final objective value.

By default, PRINTLEVEL=0.

PRIOR Statement

PRIOR *distribution* < *, . . . , distribution* > **;**

The PRIOR statement specifies the initial hyperparameter values in the prior distributions of the parameters when the maximum a posteriori method is applied to the estimation of the HMM (that is, when METHOD=MAP in the MODEL statement). If you specify the maximum likelihood (ML) method (METHOD=ML), the PRIOR statement is ignored. You can specify only one PRIOR statement. If you specify more than one *distribution*, separate them with commas.

If you specify the NSTATE=$n_1 : n_2$ option or the YLAG=$m_1 : m_2$ option (or both) in the MODEL statement to estimate multiple models, the PRIOR statement is applied to the first model that corresponds to the specification of NSTATE=n_1 or YLAG=m_1 (or that corresponds to both NSTATE=n_1 and YLAG=m_1 if both options are specified).

To use the PRIOR statement, you need to know the type of the model: different sets of prior distributions are available for different types of HMMs.

The *distribution* is in the following format:

parameter-name $\sim$ *distribution-function(arguments)* **;**

You can specify the following *parameter-names*:

CPM indicates the category probability matrix in the finite HMM. The prior distribution must be the Dirichlet distribution.

ISPV indicates the initial state probability vector. The prior distribution must be the Dirichlet distribution.

LAMBDA indicates the mean vector in the Poisson HMM. The prior distribution must be the gamma distribution.

MCP indicates the mixture component probabilities in the Gaussian mixture HMM (GM HMM). The prior distribution must be the Dirichlet distribution.

MUSIGMA indicates the mean and covariance parameters for the observation distributions. For the Gaussian HMM, regime-switching regression model, or regime-switching autoregression model, the prior distribution of the mean and covariance parameters for each state must be the normal-inverse-Wishart (NIW) distribution. For the GM HMM, the prior distribution of the mean and covariance parameters for each component at each state must be the normal-inverse-Wishart (NIW) distribution.

TPM indicates the transition probability matrix. The prior distribution must be the Dirichlet distribution.

You can specify the following *distribution-functions*:

DIR specifies the Dirichlet distribution. There is one *argument* for the Dirichlet distribution, and it must be a valid matrix expression of the proper size. For example, if the Dirichlet distribution is the prior distribution of the ISPV, MCP, CPM or TPM, the size of the *argument* must be the same as the size of the ISPV, MCP, CPM or TPM. All elements of the *argument* should be positive. If you specify any nonpositive element in the *argument*,

the nonpositive element is changed to the machine precision. For information about how to define the matrix expressions, see the section "Matrix Expression" on page 1273. For more information about the Dirichlet distribution, see the section "Maximum a Posteriori (MAP) Method" on page 1266.

GAMMA specifies the gamma distribution. There are two *arguments* (α, β) for the gamma distribution, and they must be a valid matrix expression of the proper size. The size of the *arguments* must be a $K \times 2$ matrix. All elements of the *arguments* should be positive. If you specify any nonpositive element in the *argument*, the nonpositive element is changed to the machine precision. For information about how to define the matrix expressions, see the section "Matrix Expression" on page 1273. For more information about the gamma distribution, see the section "Maximum a Posteriori (MAP) Method" on page 1266.

NIW specifies the normal-inverse-Wishart (NIW) distribution. There are four *argument*s for the NIW distribution ($\tilde{\mu}$, $\tilde{\Sigma}$, $\tilde{\kappa}$, and $\tilde{\nu}$), and they must be valid matrix expressions of the proper size. If the NIW distribution is the prior distribution of the MUSIGMA for Gaussian HMM, then the following statements are true:

- $\tilde{\mu}$ is a $K \times k_y$ matrix, where K is the number of states, and k_y is the number of dependent variables.
- $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix.
- $\tilde{\kappa}$ is a $K \times 1$ vector.
- $\tilde{\nu}$ is a $K \times 1$ vector.

If the NIW distribution is the prior distribution of the MUSIGMA for GM HMM, then the following statements are true:

- $\tilde{\mu}$ is a $KM \times k_y$ matrix, where K is the number of states, M is the number of components at a state, and k_y is the number of dependent variables.
- $\tilde{\Sigma}$ is a $k_y KM \times k_y$ matrix.
- $\tilde{\kappa}$ is a $KM \times 1$ vector.
- $\tilde{\nu}$ is a $KM \times 1$ vector.

If the NIW distribution is the prior distribution of the MUSIGMA for regime-switching regression or regime-switching autoregression, then the following statements are true:

- $\tilde{\mu}$ is a $K \times k_y k_z$ matrix, where K is the number of states, k_y is the number of dependent variables, and k_z is the number of regressors in each equation. (For more information about how to calculate k_z, see the sections "Regime-Switching Regression Model" on page 1235 and "Regime-Switching Autoregression Model" on page 1236.)
- $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix.
- $\tilde{\kappa}$ is a $k_z K \times k_z$ matrix.
- $\tilde{\nu}$ is a $K \times 1$ vector.

For information about how to define the matrix expressions, see the section "Matrix Expression" on page 1273. For more information about the normal-inverse-Wishart (NIW) distribution, see the section "Maximum a Posteriori (MAP) Method" on page 1266.

The following statements specify the PRIOR statement for the ISPV, TPM, and MUSIGMA in a bivariate three-state Gaussian HMM:

```
proc hmm data=mylib.One;
   id time=t;
   model x y / type=gaussian nstate=3 estispv method=map;
   prior ISPV ~ dir(J(3,1,1)),
         TPM ~ dir(I(3)*100+J(3,3,1)),
         MUSIGMA ~ niw(J(1,2,-1) // J(1,2,0) // J(1,2,1),
                       J(3,1,1)@(I(2)*10),
                       5*J(3,1,1),
                       10*J(3,1,1));
run;
```

SCORE Statement

SCORE *options* **;**

The SCORE statement specifies options that are related to scoring data. You can specify the following *options*:

INMODEL=*libref.data-table*

specifies the input data table for model information. The data table must be an output data table that is produced by the OUTMODEL= option in a previous call of the HMM procedure. When you specify this option, the IN= option in the ESTIMATE statement and the ID, INITIAL, MODEL, OPTIMIZE, and PRIOR statements are ignored. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

OUTMODEL=*libref.data-table*

writes the model information to the specified output data table. The output data table consists of binary large object columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters because the data type of each column is binary. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

SMOOTH Statement

SMOOTH *option* **;**

The SMOOTH statement specifies options that are related to the smoothing problem, which is to find the probability distribution of the state, conditional on all available observations. You can specify the following *option*:

OUT=*libref.data-table*
writes the smoothing results to the specified output data table. *libref.data-table* is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

STORE Statement

STORE < **OUT=** >*libref.data-table* **;**

The STORE statement saves the model information to the specified output data table that can be processed by the ASTORE procedure. An example of using analytic store technology is to perform a time-consuming model learning (that is, parameter estimation) and to store the learned model by using the STORE statement. Later you can perform the inference (namely, evaluating, filtering, smoothing, decoding, and forecasting) based on the saved model from the previous analysis, without having to learn the model again. For more information about the ASTORE procedure, see the chapter "The ASTORE Procedure" in *SAS Viya: Machine Learning Procedures*. The output data table consists of binary large object columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary.

libref.data-table is a two-level name, where *libref* refers to the library, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section "Using CAS Sessions and CAS Engine Librefs" on page 1148.

Details: HMM Procedure

Hidden Markov Model

This section is a brief review of the hidden Markov model. For more information, see Rabiner (1989), Krolzig (1997), Frühwirth-Schnatter (2006), and Cappé, Moulines, and Rydén (2010).

The hidden Markov model (HMM) is a bivariate discrete time process $\{\mathbf{S}_t, \mathbf{Y}_t\}$, where $\{\mathbf{S}_t\}$ is a (hidden) Markov chain; conditional on $\{\mathbf{S}_t\}$, the observable process $\{\mathbf{Y}_t\}$ is a sequence of independent random variables such that the conditional distribution of $\mathbf{Y}_t$ depends only on $\mathbf{S}_t$. The $\mathbf{S}_t$ is often called the state. In general, the HMM can be expressed in three equations:

1. Initialization equation: $\mathbf{S}_1 \sim h_\theta(\cdot)$
2. Transition equation: $\mathbf{S}_t \sim f_\theta(\cdot|\mathbf{S}_{t-1})$
3. Observation equation: $\mathbf{Y}_t \sim g_\theta(\cdot|\mathbf{S}_t)$

Here the initial probability distribution function $h_\theta(\cdot)$ is the probability distribution function for initial state $\mathbf{S}_1$; the transition probability distribution function $f_\theta(\cdot|\mathbf{S}_{t-1})$ is the probability distribution function of current state $\mathbf{S}_t$ conditional on the past state $\mathbf{S}_{t-1}$; the observation probability distribution function $g_\theta(\cdot|\mathbf{S}_t)$ is the probability distribution function of current observation $\mathbf{Y}_t$ conditional on the current state $\mathbf{S}_t$; and all probability distribution functions might depend on the parameter θ.

There is no restriction on the dimensionality of $\mathbf{S}_t$ and $\mathbf{Y}_t$; that is, they might be scalars, vectors, or matrices.

If the transition probability distribution function $f_\theta(\cdot|\cdot)$ is time-independent (that is, the Markov chain is time-homogeneous), then the HMM is called the (time-)homogeneous HMM; otherwise, the HMM is called the (time-)inhomogeneous HMM.

The state $\mathbf{S}_t$ could take discrete or continuous values or even a mix of discrete and continuous values.[2] In the simplest and most popular case, $\mathbf{S}_t$ is a scalar and takes finite discrete values; that is, $\mathbf{S}_t \in \{1, \ldots, K\}$. In this case, the HMM could be called the finite-state-space HMM.

For the finite-state-space HMM, the initial state distribution function $h_\theta(\cdot)$ can be expressed as a $K \times 1$ vector π, which is called the initial state probability vector (ISPV); that is,

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \ldots, K\}$$

For the finite-state-space and homogeneous HMM, the transition probability distribution function $f_\theta(\cdot|\cdot)$ can be expressed as a $K \times K$ matrix $\mathbf{A}$, which is called the transition probability matrix (TPM), where an element a_{ij} of $\mathbf{A}$ denotes the probability transitioning from past state i to current state j; that is,

$$a_{ij} = p(\mathbf{S}_t = j \,|\mathbf{S}_{t-1} = i)$$

There are six common problems to solve for an HMM:

[2] In some literature, the HMM is referred to only as the discrete-state-space HMM (that is, the state can take only discrete values). The name "state space model" is used for the continuous-state-space HMM (that is, the state can take only continuous values).

- Filtering problem: What is $p(\mathbf{S}_t|\mathbf{Y}_1, \ldots, \mathbf{Y}_t)$, the probability distribution of state $\mathbf{S}_t$ given observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_t$?
- Smoothing problem: What is $p(\mathbf{S}_t|\mathbf{Y}_1, \ldots, \mathbf{Y}_T)$, the probability distribution of state $\mathbf{S}_t$ given observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_T$, where T is the sample size?
- Forecasting problem: What is $p(\mathbf{S}_{t+h}|\mathbf{Y}_1, \ldots, \mathbf{Y}_t), h > 0$, the probability distribution of state $\mathbf{S}_{t+h}$ given observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_t$?
- Evaluating problem: What is $p(\mathbf{Y}_1, \ldots, \mathbf{Y}_t)$, the probability (or the likelihood) of observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_t$?
- Decoding problem: What is $\arg\max_{\mathbf{S}_1,\ldots,\mathbf{S}_T} p(\mathbf{S}_1, \ldots, \mathbf{S}_T, \mathbf{Y}_1, \ldots, \mathbf{Y}_T)$, the most likely sequence of hidden states $\mathbf{S}_1, \ldots, \mathbf{S}_T$ given observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_T$, where T is the sample size?
- Learning problem: How do you estimate the unknown parameter θ given observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_T$, where T is the sample size?

In most cases, the learning problem is the most difficult one. You might apply the maximum likelihood (ML) method or the maximum a posteriori (MAP) method to get the point estimate of parameter θ. For more information, see the section "Parameter Estimation Methods" on page 1266.

Several algorithms are helpful in solving the problems for the HMMs. Three of them are the forward algorithm, the backward algorithm, and the Viterbi algorithm. They might be generalized to support many types of HMMs. The following sections focus on applying them to the finite-state-space and homogeneous HMM.

Forward Algorithm

The forward algorithm recursively calculates the joint probability distribution of a state at time t, $\mathbf{S}_t$, and all observations up to time t, $\mathbf{Y}_1, \ldots, \mathbf{Y}_t$. First, define $\alpha_t(i), i = 1, \ldots, K$, as the joint probability of $(\mathbf{S}_t = i)$ and $\mathbf{Y}_1, \ldots, \mathbf{Y}_t$:

$$\alpha_t(i) \equiv p(\mathbf{S}_t = i, \mathbf{Y}_1, \ldots, \mathbf{Y}_t)$$

Then, $\alpha_t(i)$ can be recursively calculated by the following steps:

1. Calculate $\alpha_1(i), i = 1, \ldots, K$, by

$$\alpha_1(i) = \pi_i g_\theta(\mathbf{Y}_1|\mathbf{S}_1 = i)$$

2. Calculate $\alpha_t(i), i = 1, \ldots, K, t = 2, \ldots, T$, by

$$\alpha_t(i) = \left(\sum_{j=1}^{K} \alpha_{t-1}(j) a_{ji} \right) g_\theta(\mathbf{Y}_t|\mathbf{S}_t = i)$$

You can solve the evaluating problem by

$$L_t \equiv p(\mathbf{Y}_1, \ldots, \mathbf{Y}_t) = \sum_{j=1}^{K} \alpha_t(j)$$

Because the likelihood of all observations, L_T, can be calculated, you can apply the maximum likelihood method to estimate the unknown parameter θ; if the prior distribution of the parameter θ is given, you can also apply the MAP method. That is, you might use the forward algorithm in the learning problem.

You can solve the filtering problem by

$$p_t^{(f)}(i) \equiv p(\mathbf{S}_t = i | \mathbf{Y}_1, \ldots, \mathbf{Y}_t) = \alpha_t(i)/L_t$$

When the filtering problem is solved, you can solve the forecast problem, because the h-step-ahead prediction of the state probability can be calculated by

$$p_{t+h|t}^{(p)} = (\mathbf{A}')^h p_t^{(f)}$$

where $p_t^{(f)}$ is the $K \times 1$ vector with the ith element as $p_t^{(f)}(i)$, and $p_{t+h|t}^{(p)}$ is the $K \times 1$ vector with the ith element as $p_{t+h|t}^{(p)}(i)$:

$$p_{t+h|t}^{(p)}(i) \equiv p(\mathbf{S}_{t+h} = i | \mathbf{Y}_1, \ldots, \mathbf{Y}_t)$$

The distribution of $\mathbf{Y}_{t+h}$ conditional on observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_t$ is the following mixture distribution:

$$\mathbf{Y}_{t+h} | \mathbf{Y}_1, \ldots, \mathbf{Y}_t \sim \sum_{i=1}^{K} p_{t+h|t}^{(p)}(i) g_\theta(\cdot | \mathbf{S}_{t+h} = i)$$

Backward Algorithm

To solve the smoothing problem, in addition to the forward algorithm, you also need the backward algorithm. Define $\beta_t(i)$ as the probability of observing the future observations $\mathbf{Y}_{t+1}, \ldots, \mathbf{Y}_T$ conditional on the current state $\mathbf{S}_t = i$:

$$\beta_t(i) \equiv p(\mathbf{Y}_{t+1}, \ldots, \mathbf{Y}_T | \mathbf{S}_t = i)$$

The backward algorithm recursively calculates $\beta_t(i)$ as follows:

1. Set $\beta_T(i) = 1, i = 1, \ldots, K$.
2. Calculate $\beta_t(i), i = 1, \ldots, K, t = T-1, \ldots, 1$, by

$$\beta_t(i) = \sum_{j=1}^{K} a_{ij} g_\theta(\mathbf{Y}_{t+1} | \mathbf{S}_{t+1} = j) \beta_{t+1}(j)$$

When you calculate both $\alpha_t(i)$ and $\beta_t(i), i = 1, \ldots, K, t = 1, \ldots, T$, by using the forward algorithm and backward algorithm, the smoothing problem is solved, because you can calculate the probability of state $\mathbf{S}_t$ taking value i given all observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_T$ as follows:

$$\gamma_t(i) \equiv p(\mathbf{S}_t = i | \mathbf{Y}_1, \ldots, \mathbf{Y}_T) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{K} \alpha_t(j)\beta_t(j)}$$

Viterbi Algorithm

You use the Viterbi algorithm to solve the decoding problem. First, define the highest probability of a single state path ending in state $\mathbf{S}_t = i$ and observations $\mathbf{Y}_1, \ldots, \mathbf{Y}_t$ as follows:

$$V_t(i) = \max_{\mathbf{S}_1, \ldots, \mathbf{S}_{t-1}} p(\mathbf{S}_1, \ldots, \mathbf{S}_{t-1}, \mathbf{S}_t = i, \mathbf{Y}_1, \ldots, \mathbf{Y}_t)$$

To keep track of the best path, for $t = 2, \ldots, T$, define

$$v_t(i) = \arg\max_j V_{t-1}(j) a_{ji}$$

Then, $V_t(i)$ and $v_t(i)$ can be recursively calculated as follows:

1. Calculate $V_1(i), i = 1, \ldots, K$, by

$$V_1(i) = \pi_i g_\theta(\mathbf{Y}_1 | \mathbf{S}_1 = i)$$

2. Calculate $V_t(i)$ and $v_t(i), i = 1, \ldots, K, t = 2, \ldots, T$, by

$$\begin{aligned} V_t(i) &= \left(\max_j V_{t-1}(j) a_{ji} \right) g_\theta(\mathbf{Y}_t | \mathbf{S}_t = i) \\ v_t(i) &= \arg\max_j V_{t-1}(j) a_{ji} \end{aligned}$$

Then, the probability of the best path is

$$V^* = \max_i V_T(i)$$

Define $v_T^* = \arg\max_i V_T(i)$, and the best state path can be backtracked by

$$v_t^* = v_{t+1}(v_{t+1}^*), t = T - 1, \ldots, 1$$

That is, $\{v_t^*\}_{t=1,\ldots,T}$ is the best state path. The decoding problem is solved.

Gaussian Hidden Markov Model

The Gaussian hidden Markov model (Gaussian HMM) is a type of finite-state-space and homogeneous HMM where the observation probability distribution is the normal distribution,

$$\mathbf{Y}_t|\mathbf{S}_t \sim N(\mu_{\mathbf{S}_t}, \Sigma_{\mathbf{S}_t})$$

where $\mu_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters at state $\mathbf{S}_t, \mathbf{S}_t = 1, \ldots, K$. Hence, the initial state probability vector (ISPV) π, the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{\mu_i, \Sigma_i\}_{i=1,\ldots,K}$, which consists of mean and covariance parameters) together specify the Gaussian HMM; that is, the parameter θ of the Gaussian HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the Gaussian HMM is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section "Hidden Markov Model" on page 1230. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Gaussian Mixture Hidden Markov Model

The Gaussian mixture hidden Markov model (GM HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the Gaussian mixture distribution,

$$\mathbf{Y}_t|\mathbf{S}_t \sim \mathrm{GM}(\{w_{\mathbf{S}_t,1}, \ldots, w_{\mathbf{S}_t,M}\}, \{\mu_{\mathbf{S}_t,1}, \ldots, \mu_{\mathbf{S}_t,M}\}, \{\Sigma_{\mathbf{S}_t,1}, \ldots, \Sigma_{\mathbf{S}_t,M}\})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\mathrm{GM}(.,.,.)$ represents the Gaussian mixture distribution; M is the number of components at each state; and the $w_{\mathbf{S}_t,j}, j = 1, \ldots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_{\mathbf{S}_t,j} \geq 0, j = 1, \ldots, M$, and $\sum_{j=1}^{M} w_{\mathbf{S}_t,j} = 1$). The $\mu_{\mathbf{S}_t,j}$ and $\Sigma_{\mathbf{S}_t,j}, j = 1, \ldots, M$, are the mean and covariance parameters for the jth Gaussian component at state $\mathbf{S}_t$, and their values depend on the variable $\mathbf{S}_t$. Hence, the initial state probability vector (ISPV) π, the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{w_{ij}, \mu_{ij}, \Sigma_{ij}, i = 1, \ldots, K, j = 1, \ldots, M\}$, which consists of the mixture component probabilities, the mean and covariance parameters for all components) together specify the GM HMM; that is, the parameter θ of the GM HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the GM HMM is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms that are introduced in the section "Hidden Markov Model" on page 1230. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Regression Model

The regime-switching regression model (RS REG, also known as the regression hidden Markov model or REG HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the normal distribution conditional on the specified regressors,

$$\mathbf{Y}_t | z_t, \mathbf{S}_t \sim N(\mathbf{B}_{\mathbf{S}_t} z_t, \Sigma_{\mathbf{S}_t})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\mathbf{B}_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are the mean and covariance parameters, respectively, at state $\mathbf{S}_t, \mathbf{S}_t = 1, \ldots, K$. Hence, the initial state probability vector (ISPV) π, the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{\mathbf{B}_i, \Sigma_i\}_{i=1,\ldots,K}$, which consists of mean and covariance parameters) together specify the RS REG; that is, the parameter θ of the RS REG is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t, might contain the constant, seasonal dummies, linear time trend, quadratic time trend, and exogenous variables and their lagged values. That is,

$$z_t = \left(\left(z_t^{(\mathrm{CONST})}\right)' \left(z_t^{(\mathrm{SD})}\right)' \left(z_t^{(\mathrm{LTREND})}\right)' \left(z_t^{(\mathrm{QTREND})}\right)' \left(z_t^{(\mathrm{XL})}\right)' \right)'$$

where

- $z_t^{(\mathrm{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\mathrm{CONST})} = (1)$.
- $z_t^{(\mathrm{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\mathrm{SD})} = (sd_t^{(1)} \cdots sd_t^{(n_s-1)})'$, where $sd_t^{(i)}, i = 1, \ldots, n_s - 1$, denote the values of seasonal dummies at time t and where n_s is the value specified in the NSEASON=n_s option.
- $z_t^{(\mathrm{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\mathrm{LTREND})} = (t)$.
- $z_t^{(\mathrm{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\mathrm{QTREND})} = (t^2)$.
- $z_t^{(\mathrm{XL})}$ is empty if no exogenous variables are specified in the model, or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\mathrm{XL})}$ is determined as follows, where XLAG=s and k_x is the number of exogenous variables:
  - If the NOCURRENTX option is specified,
    $$z_t^{(\mathrm{XL})} = (x_{1,t-1} \cdots x_{k_x,t-1} \cdots x_{1,t-s} \cdots x_{k_x,t-s})'$$
  - If the NOCURRENTX option is not specified,
    $$z_t^{(\mathrm{XL})} = (x_{1,t} \cdots x_{k_x,t} \; x_{1,t-1} \cdots x_{k_x,t-1} \cdots x_{1,t-s} \cdots x_{k_x,t-s})'$$

The statement

```
model y = x1 x2 x3 / type=reg nstate=2;
```

defines the regression for state $i, i = 1, \ldots, K$ ($K = 2$ in this case), as

$$\begin{aligned} y_t &= b_{i,1} + b_{i,2}x1_t + b_{i,3}x2_t + b_{i,4}x3_t + \varepsilon_t \\ &= (b_{i,1}\ b_{i,2}\ b_{i,3}\ b_{i,4})z_t + \varepsilon_t \end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_i)$, $z_t = (1\ x1_t\ x2_t\ x3_t)'$, and $k_z = 4$.

In the following statement, $z_t = (1\ sd_t^{(1)}\ sd_t^{(2)}\ sd_t^{(3)}\ t\ t^2\ x1_t\ x2_t\ x3_t\ x1_{t-1}\ x2_{t-1}\ x3_{t-1})'$, where $sd_t^{(i)}, i = 1, 2, 3$, denote the values of seasonal dummies at time t and where $k_z = 12$:

```
model y = x1 x2 x3 / trend=quad nseason=4 xlag=1 type=reg nstate=2;
```

In the following statement, $z_t = (x1_{t-1}\ x2_{t-1}\ x3_{t-1})'$ and $k_z = 3$:

```
model y = x1 x2 x3 / noint xlag=1 nocurrentx type=reg nstate=2;
```

The number of dependent variables, k_y, can be greater than 1; in that case, $\mathbf{B}_i, i = 1, \ldots, K$, is the $k_y \times k_z$ matrix, and $\Sigma_i, i = 1, \ldots, K$, is the $k_y \times k_y$ symmetric positive definite matrix.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, and **XL** functions refer to the mean parameters that correspond to $z_t^{(\mathrm{CONST})}$, $z_t^{(\mathrm{SD})}$, $z_t^{(\mathrm{LTREND})}$, $z_t^{(\mathrm{QTREND})}$, and $z_t^{(\mathrm{XL})}$, respectively; the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1\ \cdots\ \Sigma_K)'$, where K is the number of states.

The Gaussian HMM can be regarded as a special case of the RS REG when there is only one regressor constant.

Because the RS REG is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section "Hidden Markov Model" on page 1230. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Autoregression Model

The regime-switching autoregression model (RS AR, also known as the autoregressive hidden Markov model or AR HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the normal distribution conditional on the specified regressors. For a standard form of RS AR,

$$\mathbf{Y}_t | z_t, \mathbf{S}_t \sim N(\mathbf{B}_{\mathbf{S}_t} z_t, \Sigma_{\mathbf{S}_t})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\mathbf{B}_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters at state $\mathbf{S}_t, \mathbf{S}_t = 1, \ldots, K$. Hence, the initial state probability vector (ISPV) π, the transition probability matrix (TPM) $\mathbf{A}$, and the parameter $\mathbf{B}$ ($\equiv \{\mathbf{B}_i, \Sigma_i\}_{i=1,\ldots,K}$, which consists of mean and covariance parameters) together specify the RS AR; that is, the parameter θ of the RS AR is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t, might contain the constant, seasonal dummies, linear time trend, quadratic time trend, exogenous variables and their lagged values, and the lagged values of the dependent variables. That is,

$$z_t = \left(\left(z_t^{(\text{CONST})}\right)' \left(z_t^{(\text{SD})}\right)' \left(z_t^{(\text{LTREND})}\right)' \left(z_t^{(\text{QTREND})}\right)' \left(z_t^{(\text{XL})}\right)' \left(z_t^{(\text{AR})}\right)' \right)'$$

where

- $z_t^{(\text{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\text{CONST})} = (1)$.
- $z_t^{(\text{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\text{SD})} = (sd_t^{(1)} \cdots sd_t^{(n_s-1)})'$, where $sd_t^{(i)}, i = 1, \ldots, n_s - 1$, denote the values of seasonal dummies at time t and where n_s is the value specified in the NSEASON=n_s option.
- $z_t^{(\text{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\text{LTREND})} = (t)$.
- $z_t^{(\text{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\text{QTREND})} = (t^2)$.
- $z_t^{(\text{XL})}$ is empty if no exogenous variables are specified in the model, or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\text{XL})}$ is determined as follows, where XLAG=s and k_x is the number of exogenous variables:
  - If the NOCURRENTX option is specified,
    $$z_t^{(\text{XL})} = (x_{1,t-1} \cdots x_{k_x,t-1} \cdots x_{1,t-s} \cdots x_{k_x,t-s})'$$
  - If the NOCURRENTX option is not specified,
    $$z_t^{(\text{XL})} = (x_{1,t} \cdots x_{k_x,t} \; x_{1,t-1} \cdots x_{k_x,t-1} \cdots x_{1,t-s} \cdots x_{k_x,t-s})'$$
- $z_t^{(\text{AR})}$ is empty if the YLAG= option is not specified or YLAG=0 is specified; otherwise, $z_t^{(\text{AR})}$ is determined as follows, where YLAG=p and k_y is the number of dependent variables:

$$z_t^{(\text{AR})} = (y_{1,t-1} \cdots y_{k_y,t-1} \cdots y_{1,t-p} \cdots y_{k_y,t-p})'$$

The statement

```
model y = x1 x2 x3 / ylag=2 type=ar nstate=2;
```

defines the autoregression for state $i, i = 1, \ldots, K (K = 2$ in this case), as

$$\begin{aligned} y_t &= b_{i,1} + b_{i,2}x1_t + b_{i,3}x2_t + b_{i,4}x3_t + b_{i,5}y_{t-1} + b_{i,6}y_{t-2} + \varepsilon_t \\ &= (b_{i,1}\ b_{i,2}\ b_{i,3}\ b_{i,4}\ b_{i,5}\ b_{i,6})z_t + \varepsilon_t \end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_i)$, $z_t = (1\ x1_t\ x2_t\ x3_t\ y_{t-1}\ y_{t-2})'$, and $k_z = 6$.

In the following statement, $z_t = (1\ sd_t^{(1)}\ sd_t^{(2)}\ sd_t^{(3)}\ t\ t^2\ x1_t\ x2_t\ x3_t\ x1_{t-1}\ x2_{t-1}\ x3_{t-1}\ y_{t-1}\ y_{t-2})'$, where $sd_t^{(i)}, i = 1, 2, 3$, denote the values of seasonal dummies at time t, and $k_z = 14$:

```
model y = x1 x2 x3 / ylag=2 trend=quad nseason=4 xlag=1 type=reg nstate=2;
```

In the following statement, $z_t = (x1_{t-1}\ x2_{t-1}\ x3_{t-1}\ y1_{t-1}\ y2_{t-1}\ y1_{t-2}\ y2_{t-2})'$ and $k_z = 7$:

```
model y1 y2 = x1 x2 x3 / ylag=2 noint xlag=1 nocurrentx type=reg nstate=2;
```

The number of dependent variables, k_y, can be greater than 1 (for example, $k_y = 2$ in the previous statement); in that case, $\mathbf{B}_i, i = 1, \ldots, K$, is the $k_y \times k_z$ matrix, and $\Sigma_i, i = 1, \ldots, K$, is the $k_y \times k_y$ symmetric positive definite matrix. When $k_y > 1$, the model is called the vector autoregressive hidden Markov model (VAR HMM) in some literature.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, **XL**, and **AR** functions refer to the mean parameters that correspond to $z_t^{(\mathrm{CONST})}$, $z_t^{(\mathrm{SD})}$, $z_t^{(\mathrm{LTREND})}$, $z_t^{(\mathrm{QTREND})}$, $z_t^{(\mathrm{XL})}$, and $z_t^{(\mathrm{AR})}$, respectively; the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1\ \cdots\ \Sigma_K)'$, where K is the number of states.

The RS REG can be regarded as a special case of the RS AR when YLAG=0 is specified.

Because the RS AR is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section "Hidden Markov Model" on page 1230. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Mean-Adjusted Autoregression Model

The regime-switching mean-adjusted autoregression model (also known as the mean-adjusted autoregressive hidden Markov model) is a type of finite-state-space and homogeneous HMM. For a regime-switching mean-adjusted autoregression model, the observation $\mathbf{Y}_t$ follows the normal distribution as

$$\mathbf{Y}_t|\mathbf{Y}_t, \ldots, \mathbf{Y}_{t-p}, z_t, \ldots z_{t-p}, \mathbf{S}_t, \mathbf{S}_{t-1} \ldots, \mathbf{S}_{t-p} \sim N(\mathbf{B}^{(R)}_{\mathbf{S}_t} z_t + \sum_{\tau=1}^{p} \mathbf{B}^{(A)}_{\mathbf{S}_{t-\tau}} (\mathbf{Y}_{t-\tau} - \mathbf{B}^{(R)}_{\mathbf{S}_{t-\tau}} z_{t-p}), \Sigma_{\mathbf{S}_t})$$

where p is the autoregressive order, the variable $\mathbf{S}_t$ is the (hidden) state, $\mathbf{B}^{(A)}_{\mathbf{S}_{t-\tau}}$ is the autoregressive part of mean parameters at state $\mathbf{S}_t$, $\mathbf{B}^{(R)}_{\mathbf{S}_t}$ is the remaining mean parameters at state $\mathbf{S}_t$, and $\Sigma_{\mathbf{S}_t}$ is the covariance parameters at state $\mathbf{S}_t$ for $\mathbf{S}_t = 1, \ldots, K$. Hence, the initial state probability vector (ISPV) π, the transition probability matrix (TPM) $\mathbf{A}$, and the parameter $\mathbf{B}$ ($\equiv \{\mathbf{B}^{(A)}_i, \mathbf{B}^{(R)}_i, \Sigma_i\}_{i=1,\ldots,K}$) together specify the regime-switching mean-adjusted autoregression model; that is, the parameter θ of the regime-switching mean-adjusted autoregression model is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t, might contain the constant, seasonal dummies, linear time trend, quadratic time trend, exogenous variables. That is,

$$z_t = \left(\left(z_t^{(\mathrm{CONST})}\right)' \left(z_t^{(\mathrm{SD})}\right)' \left(z_t^{(\mathrm{LTREND})}\right)' \left(z_t^{(\mathrm{QTREND})}\right)' \left(z_t^{(\mathrm{XL})}\right)' \right)'$$

where

- $z_t^{(\mathrm{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\mathrm{CONST})} = (1)$.
- $z_t^{(\mathrm{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\mathrm{SD})} = (sd_t^{(1)} \ \cdots \ sd_t^{(n_s-1)})'$, where $sd_t^{(i)}, i = 1, \ldots, n_s - 1$, denote the values of the seasonal dummies at time t and where n_s is the value specified in the NSEASON=n_s option.
- $z_t^{(\mathrm{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\mathrm{LTREND})} = (t)$.
- $z_t^{(\mathrm{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\mathrm{QTREND})} = (t^2)$.
- $z_t^{(\mathrm{XL})}$ is empty if no exogenous variables are specified in the model or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\mathrm{XL})}$ is determined as follows, where XLAG=s and k_x is the number of exogenous variables:
  - If the NOCURRENTX option is specified,
    $$z_t^{(\mathrm{XL})} = (x_{1,t-1} \ \cdots x_{k_x,t-1} \ \cdots \ x_{1,t-s} \ \cdots x_{k_x,t-s})'$$
  - If the NOCURRENTX option is not specified,
    $$z_t^{(\mathrm{XL})} = (x_{1,t} \ \cdots x_{k_x,t} \ x_{1,t-1} \ \cdots x_{k_x,t-1} \ \cdots \ x_{1,t-s} \ \cdots x_{k_x,t-s})'$$

The ARMEAN=ADJUSTED option in the following statement specifies the following regime-switching mean-adjusted autoregression model:

```
model y = x / armean=adjusted ylag=1 type=ar nstate=2;
```

$$\begin{aligned}
y_t &= c_{\mathbf{S}_t} + b_{\mathbf{S}_t} x_t + a_{\mathbf{S}_t}(y_{t-1} - c_{\mathbf{S}_{t-1}} - b_{\mathbf{S}_{t-1}} x_{t-1}) + \varepsilon_t \\
&= (c_{\mathbf{S}_t} \ b_{\mathbf{S}_t}) z_t + a_{\mathbf{S}_t}(y_{t-1} - (c_{\mathbf{S}_{t-1}} \ b_{\mathbf{S}_{t-1}}) z_{t-1}) + \varepsilon_t \\
&= \begin{cases}
c_1 + b_1 x_t + a_1(y_{t-1} - c_1 - b_1 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 1, \mathbf{S}_{t-1} = 1, \\
c_2 + b_2 x_t + a_2(y_{t-1} - c_1 - b_1 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 2, \mathbf{S}_{t-1} = 1, \\
c_1 + b_1 x_t + a_1(y_{t-1} - c_2 - b_2 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 1, \mathbf{S}_{t-1} = 2, \\
c_2 + b_2 x_t + a_2(y_{t-1} - c_2 - b_2 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 2, \mathbf{S}_{t-1} = 2.
\end{cases}
\end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, $z_t = (1 \ x_t)'$, and $k_z = 2$.

The number of dependent variables, k_y, can be greater than 1; in that case, $\Sigma_i, i = 1, \ldots, K$, is the $k_y \times k_y$ symmetric positive definite matrix. When $k_y > 1$, the model is called the vector autoregressive hidden Markov model (VAR HMM) in some literature.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, and **XL** functions refer to the mean parameters that correspond to $z_t^{(\mathrm{CONST})}$, $z_t^{(\mathrm{SD})}$, $z_t^{(\mathrm{LTREND})}$, $z_t^{(\mathrm{QTREND})}$, $z_t^{(\mathrm{XL})}$, respectively; the **AR** function refers to the mean parameters that correspond to demeaned lagged dependent variables $(\mathbf{Y}_{t-\tau} - \mathbf{B}^{(R)}_{\mathbf{S}_{t-\tau}} z_{t-\tau}), \tau = 1, \ldots, p$; and the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1 \ \cdots \ \Sigma_K)'$, where K is the number of states.

Because the regime-switching mean-adjusted autoregression model is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms that are introduced in the section "Hidden Markov Model" on page 1230. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Finite Hidden Markov Model

The finite hidden Markov model (finite HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the categorical distribution whose support is finite,

$$Y_t|\mathbf{S}_t \sim C(b_{\mathbf{S}_t,1}, \ldots, b_{\mathbf{S}_t,M})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $C(\ldots)$ represents the categorical distribution; M is the number of categories at each state; and the $b_{\mathbf{S}_t,j}$, $j = 1, \ldots, M$, are category probabilities, satisfying ($b_{\mathbf{S}_t,j} \geq 0$, $j = 1, \ldots, M$, and $\sum_{j=1}^{M} b_{\mathbf{S}_t,j} = 1$).

Hence, the initial state probability vector (ISPV) π, the transition probability matrix (TPM) $\mathbf{A}$, and the probability of observational outcomes $\mathbf{B}$ ($\equiv \{b_{ij}, i = 1, \ldots, K, j = 1, \ldots, M\}$) together specify the finite HMM; that is, the parameter θ of the finite HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the finite HMM is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms that are introduced in the section "Hidden Markov Model" on page 1230. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood. The dependent variable for the finite HMM must be an integer greater than or equal to 1 and less than or equal to the number of the category. The number of dependent variables for finite HMM must be 1. It is a trivial case when NSTATE=1 and NCATEGORY=1 for the finite HMM, and the program will stop processing.

The %FiniteHMM Macro

Although in theory, finite HMM assumes that the response variable is discrete (whether it is numeric or categorical), PROC HMM assumes that the response variable is composed of consecutive natural numbers that start with 1. However, for most real-world business data, the variable of interest could be recorded in other formats, such as categorical levels, decimals, negative numbers, and so on. In other cases, a response variable could be recorded as nonconsecutive natural numbers. Therefore it is likely that you will need to preprocess the data before using PROC HMM to perform finite HMM.

For example, if you are interested in modeling the dynamic feature of receiving promotion channels for a buyer, the response variable y_1 could have values such as 'aa email', 'bb mail', 'cc phone', 'dd flyer', 'ee

seminar', and 'ff forum'. To use PROC HMM for finite HMM on y_1, you would need to preprocess the data by creating a natural-number-valued variable y and model on y instead. If you assume that y_1 is in the data set finite1, the following SAS program does the preprocessing:

```
data finite1;
   set finite1;
   if y1='aa email  ' then y=1;
   if y1='bb mail   ' then y=2;
   if y1='cc phone  ' then y=3;
   if y1='dd flyer  ' then y=4;
   if y1='ee seminar' then y=5;
   if y1='ff forum  ' then y=6;
run;
```

Then you can use PROC HMM for finite HMM on y as follows after you upload the data set to a SAS data library named mylib. Here the SAS data library mylib must already be defined.

```
ods output CPM=my_cpm_0
ParameterEstimates=my_est_0;
proc hmm data=mylib.finite1;
   id time=t section=section;
   model y / type=finite nstate=3 nCategory=6 method=MAP;
   estimate out=mylib.est_0 outall=mylib.estall_0;
   forecast out=mylib.fcst_0 lead=4;
run;
```

Because y has values 1 to 6, the output tables have information only about the levels of y, not the levels of y_1. To understand and analyze the results for the original y_1, you need to do postprocessing by manually matching the y levels to the levels of y_1. This can be tedious if y_1 has too many levels. The SAS macro %FiniteHMM helps you do the preprocessing and postprocessing automatically in such cases. The %FiniteHMM macro is a wrapper for finite HMM with PROC HMM. It treats the response variable as a categorical variable, and it automatically preprocesses the data and postprocesses the results. This macro is useful in the following three cases:

- The response variable contains values other than natural numbers.
- The response variable is composed of nonconsecutive natural numbers.
- The response variable is composed of consecutive natural numbers but does not start with 1.

Be aware that there are limits to using the %FiniteHMM macro. In particular, it does not currently support n-literal response variable names and does not support interior quotation marks in response variable values.

Specifically, the %FiniteHMM macro does the following:

- treats the response variable for finite HMM as a categorical variable
- creates the internal response variable _mody_ with consecutive natural numbers as its values, and links each natural number to the corresponding categorical level of the original response variable
- updates the NCATEGORY= option value to the actual number of levels of the original response variable

- applies PROC HMM to the internally created response variable _mody_ for finite HMM
- for the ODS CPM output table and the output forecast table, updates the label of categorical columns with their corresponding levels in the original response variable
- for both the ODS table and the output parameter estimates table, adds a column named RespVarLevel that indicates the corresponding levels in the original response variable

These updated tables from the %FiniteHMM macro are ready for analysis on the original response variable because the parameters are linked to the original response variable levels through labels or through the additional column RespVarLevel. The ODS CPM output table and output forecast table will have labels of the variables Category1 to Categoryk that are linked to the levels of the original variable. The parameter estimates table will have an additional column called RespVarLevel to indicate the levels of the original variable.

The parameters of the %FiniteHMM macro are defined to correspond to the statements in PROC HMM. The options in each statement of PROC HMM become the parameter values in the macro. Two additional parameters are added to the macro:

- DATA=%str(), which contains the name of the two-level input data set for PROC HMM
- ODSOUT=%str(), which contains the ODS table specification in the ODS OUTPUT statement of PROC HMM

Because the value of the DATA= option in PROC HMM is defined as a separate parameter in the %FiniteHMM macro, the PROC parameter in the macro contains only options other than the DATA= option of PROC HMM. To prevent ambiguity with the special symbols, it is suggested that the value of each parameter be quoted in %str( ). The following parameters are defined in the %FiniteHMM macro:

DATA
specifies the two-level input table name.

DECODE
specifies options for the DECODE statement.

ESTIMATE
specifies options for the ESTIMATE statement.

EVALUATE
specifies options for the EVALUATE statement.

FILTER
specifies options for the FILTER statement.

FORECAST
specifies options for the FORECAST statement.

ID
specifies options for the ID statement.

INITIAL
specifies options for the INITIAL statement.

MODEL
specifies options for the MODEL statement.

ODSOUT
specifies options for the ODS OUTPUT statement.

OPTIMIZE
specifies options for the OPTIMIZE statement.

PRIOR
specifies options for the PRIOR statement.

PROC
specifies options in the PROC HMM statement, excluding the DATA= option.

SCORE
specifies options for the SCORE statement.

SMOOTH
specifies options for the SMOOTH statement.

Among the macro parameters, only DATA, ID, and MODEL are required. All others are optional. For example, say that you run finite HMM by using the following PROC HMM specification:

```
ods output CPM=my_cpm_1
    ParameterEstimates=my_est_1;
proc hmm data=mylib.finite1;
   id time=t section=section;
   model y / type=finite nstate=3 nCategory=6 method=MAP;
   estimate out=mylib.est_1 outall=mylib.estall_1;
   forecast out=mylib.fcst_1 lead=4;
run;
```

The corresponding %FiniteHMM macro syntax would be as follows:

```
%FiniteHMM(
   data = %str(mylib.finite1),
   id = %str(time=t section=section),
   model = %str(y / type=finite nstate=3 nCategory=6 method=MAP),
   estimate = %str(out=mylib.est_1 outall=mylib.estall_1),
   forecast = %str(out=mylib.fcst_1 lead=4),
   odsout = %str(CPM=my_cpm_1
                 ParameterEstimates=my_est_1));
```

Note that in the preceding example, because DATA=MYLIB.FINITE1 is the only option in the PROC HMM statement, the PROC parameter is not needed in the %FiniteHMM macro syntax. The NCATEGORY=6 option is not required in this macro because this value is determined by input data; hence, even if you specify it as shown here, it will be overwritten according to the data.

Now say that you have a more complicated specification of PROC HMM, such as the following:

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite1 labelSwitch=(sort=asc(tpm));
   id time=t section=section;
   model y1 / type=finite nstate=3 nCategory=6 method=MAP;
   score outmodel=mylib.myScore_0;
   estimate out=mylib.est_0 outall=mylib.estall_0;
   forecast out=mylib.fcst_0 lead=4;
   decode out=mylib.decode_0;
   evaluate out=mylib.eval_0;
   filter out=mylib.flt_0;
   smooth out=mylib.smth_0;
   optimize printlevel=3 printiterfreq=2 maxiter=10;
   prior CPM ~ dir({1 1 1 2 2 2,
                   2 2 2 3 3 3,
                   2 2 2 4 4 4});
   initial TPM =   {0.97 0.02 0.01,
                    0.015 0.98 0.005,
                    0.01 0.03 0.96},
           CPM =   {0.09 0.32 0.57 0.01 0.01 0.00,
                    0.01 0.08 0.28 0.61 0.01 0.01,
                    0.01 0.02 0.01 0.08 0.29 0.59};
run;
```

The corresponding %FiniteHMM macro syntax would look like this:

```
%FiniteHMM(
   data = %str(mylib.finite1),
   proc = %str(labelSwitch=(sort=asc(tpm))),
   id = %str(time=t section=section),
   model = %str(y1 / type=finite nstate=3 nCategory=6 method=MAP),
   score = %str(outmodel=mylib.myScore_0),
   estimate = %str( out=mylib.est_0 outall=mylib.estall_0),
   forecast = %str(out=mylib.fcst_0 lead=4),
   decode = %str(out=mylib.decode_0),
   evaluate = %str(out=mylib.eval_0),
   filter = %str(out=mylib.flt_0),
   smooth = %str(out=mylib.smth_0),
   optimize= %str(printlevel=3 printiterfreq=2 maxiter=10),
   prior = %str(CPM ~ dir({1 1 1 2 2 2,
                           2 2 2 3 3 3,
                           2 2 2 4 4 4}) ),
   initial = %str(TPM =   {0.97 0.02 0.01,
                           0.01 0.98 0.01,
                           0.01 0.03 0.96},
                  CPM =   {0.09 0.32 0.57 0.01 0.01 0.00,
                           0.01 0.08 0.28 0.61 0.01 0.01,
                           0.01 0.02 0.01 0.08 0.29 0.59}),
   odsout = %str(CPM=my_cpm_0
                 ParameterEstimates=my_est_0 ));
```

Note that in the preceding example, PROC HMM has the LABELSWITCH=(SORT=ASC(TPM)) option in addition to the DATA= option, so a PROC parameter is included in the macro call to contain this option.

The sections that follow present several examples to demonstrate how to use the %FiniteHMM macro. First, a data set needs to be simulated for these macro examples. The following PROC IML and DATA step code simulates a data set on the basis of a finite HMM model with the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$:

$$\pi = \begin{pmatrix} 0.45 \\ 0.25 \\ 0.30 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 0.98 & 0.015 & 0.005 \\ 0.02 & 0.97 & 0.01 \\ 0.01 & 0.01 & 0.98 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{pmatrix}$$

where π is the initial state probability vector, or ISPV; $\mathbf{A}$ is the transition probability matrix, or TPM; and $\mathbf{B}$ is the category probability matrix, or CPM. The simulated data set, finite1, has 1,500 observations in three sections. Each section has 500 observations. The data set has three variables: y, t, and section.

```
proc iml;
*** Defining model parameters: A and PI **************;
   a={0.98 0.015 0.005, 0.02 0.97 0.01, 0.01 0.01 0.98};
   pi={0.45 0.25 0.30};
*** Simulating a Markov process ************************;
   seed=54321;
   call streaminit(seed);
   free st state;
   do i=1 to 500;
      tmp_st={0 0 0};
      if i=1 then do;
      check_p=pi;
      end;
      else do;
         lag_st=st[i-1,];
         check_p=lag_st*a;
      end;
      u=uniform(seed);
      if u<=check_p[1] then tmp_st[1]=1;
      else do;
         tmp=check_p[1] + check_p[2];
         if u<=tmp then tmp_st[2]=1;
         else tmp_st[3]=1;
      end;
      tmp_state=loc(tmp_st=1);
      st=st//tmp_st;
      state=state//tmp_state;
      seed=seed+1;
      end;
   create sim_stat from st[colname={"state1" "state2" "state3"}];
   append from st;
   create sim_state from state[colname={"state1"}];
   append from state;
quit;
*** Simulating finite HMM data ************************;
%let simseed=4321;
data temnm(keep = section t y);
   call streaminit(&simseed.);
   retain section 1;
   set sim_state;
```

```
   t=_n_;
   if(state1=1) then do;
      u = uniform(&simseed.);
      if (u <= 1/2) then y=1;
      else if( 1/2 <u <= 2/2) then y=2;
      else if( 2/2 <u <= 1) then y=3;
      else if( 1 <u <= 1) then y=4;
      else if( 1 <u <= 1) then y=5;
      else y = 6;
   end;
   else if state1=2 then do;
      u = uniform(&simseed.);
      if (u <= 0) then y=1;
      else if( 0 <u <= 0) then y=2;
      else if( 0 <u <= 1/2) then y=3;
      else if( 1/2 <u <= 2/2) then y=4;
      else if( 2/2 <u <= 1) then y=5;
      else y = 6;
   end;
   else do;
      u = uniform(&simseed.);
      if (u <= 0) then y=1;
      else if( 0 <u <= 0) then y=2;
      else if( 0 <u <= 0) then y=3;
      else if( 0 <u <= 0) then y=4;
      else if( 0 <u <= 1/2) then y=5;
      else y = 6;
   end;
   output;
run;
data finite1;
   set temnm;
run;
%let simseed1=1234;
data temnm(keep = section t y);
   call streaminit(&simseed1.);
   retain section 2;
   set sim_state;
   t=_n_;
   if(state1=1) then do;
      u = uniform(&simseed1.);
      if (u <= 1/2) then y=1;
      else if( 1/2 <u <= 2/2) then y=2;
      else if( 2/2 <u <= 1) then y=3;
      else if( 1 <u <= 1) then y=4;
      else if( 1 <u <= 1) then y=5;
      else y = 6;
   end;
   else if state1=2 then do;
      u = uniform(&simseed1.);
      if (u <= 0) then y=1;
      else if( 0 <u <= 0) then y=2;
      else if( 0 <u <= 1/2) then y=3;
      else if( 1/2 <u <= 2/2) then y=4;
```

```
      else if( 2/2 <u <= 1) then y=5;
      else y = 6;
   end;
   else do;
      u = uniform(&simseed1.);
      if (u <= 0) then y=1;
      else if( 0 <u <= 0) then y=2;
      else if( 0 <u <= 0) then y=3;
      else if( 0 <u <= 0) then y=4;
      else if( 0 <u <= 1/2) then y=5;
      else y = 6;
   end;
   output;
run;
data finite1;
   set finite1 temnm;
run;
%let simseed2=5432;
data temnm(keep = section t y);
   call streaminit(&simseed2.);
   retain section 3;
   set sim_state;
   t=_n_;
   if(state1=1) then do;
      u = uniform(&simseed2.);
      if (u <= 1/2) then y=1;
      else if( 1/2 <u <= 2/2) then y=2;
      else if( 2/2 <u <= 1) then y=3;
      else if( 1 <u <= 1) then y=4;
      else if( 1 <u <= 1) then y=5;
      else y = 6;
   end;
   else if state1=2 then do;
      u = uniform(&simseed2.);
      if (u <= 0) then y=1;
      else if( 0 <u <= 0) then y=2;
      else if( 0 <u <= 1/2) then y=3;
      else if( 1/2 <u <= 2/2) then y=4;
      else if( 2/2 <u <= 1) then y=5;
      else y = 6;
   end;
   else do;
      u = uniform(&simseed2.);
      if (u <= 0) then y=1;
      else if( 0 <u <= 0) then y=2;
      else if( 0 <u <= 0) then y=3;
      else if( 0 <u <= 0) then y=4;
      else if( 0 <u <= 1/2) then y=5;
      else y = 6;
   end;
   output;
run;
data finite1;
   set finite1 temnm;
```

```
run;
```

The variable y in the finite1 data set has values of the consecutive natural numbers 1 to 6. In this case, you can use either PROC HMM or the %FiniteHMM macro on y for a finite HMM. The results should be the same.

The following SAS code saves the finite1 data set to the mylib library:

```
data mylib.finite1;
   set finite1;
run;
```

Macro Example 1: Finite HMM with Consecutive Natural Number Responses

In this example, both PROC HMM and the %FiniteHMM macro should give you the same numeric results. Both the ODS table and the output parameter estimates table that this macro produces have an additional column named RespVarLevel to indicate the levels of the response variable. In practice, if a response variable has values of consecutive natural numbers, you can use PROC HMM directly.

The first part of the following SAS code uses PROC HMM on y. The ODS output tables for CPM and the parameter estimates are stored in my_cml_0 and my_est_0, respectively. Then %FiniteHMM is used on y for the same model specification. The ODS output tables for CPM and the parameter estimates are stored in my_cml_1 and my_est_1, respectively.

```
*** Using PROC HMM on y ******************************;
ods listing close;
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite1 ;
   id time=t section=section;
   model y / type=finite nstate=3 nCategory=6 method=MAP;
   estimate out=mylib.est_0 outall=mylib.estall_0;
   forecast out=mylib.fcst_0 lead=4;
run;
*** Using %FiniteHMM on y ****************************;
%FiniteHMM(
   data=%str(mylib.finite1),
   id = %str(time=t section=section),
   model = %str(y / type=finite nstate=3 nCategory=6 method=MAP),
   estimate = %str( out=mylib.est_1 outall=mylib.estall_1),
   forecast = %str(out=mylib.fcst_1 lead=4),
   odsout = %str(CPM=my_cpm_1
                 ParameterEstimates=my_est_1)
);
```

The following SAS code prints the ODS CPM output table from PROC HMM. The table is shown in Figure 20.68.

```
proc print data=my_cpm_0 label noobs;
   format category: 8.5;
run;
```

Figure 20.68 ODS CPM Output Table from PROC HMM

Group	State	1	2	3	4	5	6
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.51608	0.48392
ParameterMatrices	2	0.00000	0.00000	0.48113	0.51887	0.00000	0.00000
ParameterMatrices	3	0.52209	0.47791	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS CPM output table from the %FiniteHMM macro. The table is shown in Figure 20.69.

```
proc print data=my_cpm_1 label noobs;
   format category: 8.5;
run;
```

Figure 20.69 ODS CPM Output Table from the %FiniteHMM Macro

Group	State	1	2	3	4	5	6
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.51608	0.48392
ParameterMatrices	2	0.00000	0.00000	0.48113	0.51887	0.00000	0.00000
ParameterMatrices	3	0.52209	0.47791	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS parameter estimates table from PROC HMM. The table is shown in Figure 20.70.

```
proc print data=my_est_0  noobs;
   format Estimate StdErr 8.5;
run;
```

Figure 20.70 ODS Parameter Estimates Table from PROC HMM

Parameter	Estimate	StdErr	tValue	Probt
TPM1_1	0.98912	0.00380	260.10	<.0001
TPM1_2	0.00000	0.00000	0.00	0.9986
TPM1_3	0.01088	0.00380	2.86	0.0043
TPM2_1	0.00941	0.00543	1.73	0.0837
TPM2_2	0.97953	0.00771	127.07	<.0001
TPM2_3	0.01107	0.00595	1.86	0.0629
TPM3_1	0.01012	0.00425	2.38	0.0173
TPM3_2	0.01014	0.00425	2.39	0.0172
TPM3_3	0.97974	0.00614	159.65	<.0001
CPM1_1	0.00000	0.00000	0.00	0.9974
CPM1_2	0.00000	0.00001	0.00	0.9973
CPM1_3	0.00000	0.00000	0.00	0.9984
CPM1_4	0.00000	0.00000	0.00	0.9986
CPM1_5	0.51608	0.01911	27.01	<.0001
CPM1_6	0.48392	0.01911	25.33	<.0001
CPM2_1	0.00000	0.00001	0.00	0.9972
CPM2_2	0.00000	0.00001	0.00	0.9987
CPM2_3	0.48113	0.02802	17.17	<.0001
CPM2_4	0.51887	0.02802	18.52	<.0001
CPM2_5	0.00000	0.00009	0.00	0.9999
CPM2_6	0.00000	0.00009	0.00	0.9999
CPM3_1	0.52209	0.02238	23.32	<.0001
CPM3_2	0.47791	0.02238	21.35	<.0001
CPM3_3	0.00000	0.00002	0.00	0.9999
CPM3_4	0.00000	0.00001	0.00	0.9997
CPM3_5	0.00000	0.00000	0.00	1.0000
CPM3_6	0.00000	0.00001	0.00	0.9997

The following SAS code prints the ODS parameter estimates table from the %FiniteHMM macro. The table is shown in Figure 20.71.

```
proc print data=my_est_1  noobs;
   format Estimate StdErr 8.5;
run;
```

Figure 20.71 ODS Parameter Estimates Table from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	StdErr	tValue	Probt
	TPM1_1	0.98912	0.00380	260.10	<.0001
	TPM1_2	0.00000	0.00000	0.00	0.9986
	TPM1_3	0.01088	0.00380	2.86	0.0043
	TPM2_1	0.00941	0.00543	1.73	0.0837
	TPM2_2	0.97953	0.00771	127.07	<.0001
	TPM2_3	0.01107	0.00595	1.86	0.0629
	TPM3_1	0.01012	0.00425	2.38	0.0173
	TPM3_2	0.01014	0.00425	2.39	0.0172
	TPM3_3	0.97974	0.00614	159.65	<.0001
1	CPM1_1	0.00000	0.00000	0.00	0.9974
2	CPM1_2	0.00000	0.00001	0.00	0.9973
3	CPM1_3	0.00000	0.00000	0.00	0.9984
4	CPM1_4	0.00000	0.00000	0.00	0.9986
5	CPM1_5	0.51608	0.01911	27.01	<.0001
6	CPM1_6	0.48392	0.01911	25.33	<.0001
1	CPM2_1	0.00000	0.00001	0.00	0.9972
2	CPM2_2	0.00000	0.00001	0.00	0.9987
3	CPM2_3	0.48113	0.02802	17.17	<.0001
4	CPM2_4	0.51887	0.02802	18.52	<.0001
5	CPM2_5	0.00000	0.00009	0.00	0.9999
6	CPM2_6	0.00000	0.00009	0.00	0.9999
1	CPM3_1	0.52209	0.02238	23.32	<.0001
2	CPM3_2	0.47791	0.02238	21.35	<.0001
3	CPM3_3	0.00000	0.00002	0.00	0.9999
4	CPM3_4	0.00000	0.00001	0.00	0.9997
5	CPM3_5	0.00000	0.00000	0.00	1.0000
6	CPM3_6	0.00000	0.00001	0.00	0.9997

Figure 20.71 shows that the parameter estimates table that the %FiniteHMM macro produces has an additional column named RespVarLevel to indicate the levels of the original response variable.

Macro Example 2: Finite HMM with Categorical Responses

In this example, the variable y_1 has the categorical values 'aa email', 'bb mail', 'cc phone', 'dd flyer', 'ee seminar', and 'ff forum'. In this case, directly applying PROC HMM to y_1 generates error messages, and PROC HMM quits. Using the %FiniteHMM macro on y_1 gives you the expected results.

The following SAS code creates the finite2 data set:

```
data finite2;
   set finite1;
   if y=1 then y1='aa email  ';
   if y=2 then y1='bb mail   ';
   if y=3 then y1='cc phone  ';
   if y=4 then y1='dd flyer  ';
   if y=5 then y1='ee seminar';
   if y=6 then y1='ff forum  ';
```

```
run;
data mylib.finite2;
   set finite2;
run;
```

The following SAS code uses PROC HMM:

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite2;
   id time=t section=section;
   model y1 / type=finite nstate=3 nCategory=6 method=MAP;
   estimate out=mylib.est_0 outall=mylib.estall_0;
   forecast out=mylib.fcst_0 lead=4;
run;
ERROR: The data type of dependent variables must be numeric.
```

The following SAS code applies the %FiniteHMM macro:

```
%FiniteHMM(
   data = %str(mylib.finite2),
   proc = %str(labelSwitch=(sort=asc(cpm))),
   id = %str(time=t section=section),
   model = %str(y1 / type=finite nstate=3 nCategory=6 method=MAP),
   estimate = %str( out=mylib.est_1 outall=mylib.estall_1),
   forecast = %str(out=mylib.fcst_1 lead=4),
   odsout = %str(CPM=my_cpm_1
                 ParameterEstimates=my_est_1));
```

The following SAS code prints the ODS CPM output table from the %FiniteHMM macro. The table is shown in Figure 20.72.

```
proc print data=my_cpm_1 label noobs;
   format category: 8.5;
run;
```

Figure 20.72 ODS CPM Output Table from the %FiniteHMM Macro

Group	State	aa email	bb mail	cc phone	dd flyer	ee seminar	ff forum
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.51608	0.48392
ParameterMatrices	2	0.00000	0.00000	0.48113	0.51887	0.00000	0.00000
ParameterMatrices	3	0.52209	0.47791	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS parameter estimates table from the %FiniteHMM macro. The table is shown in Figure 20.73.

```
proc print data=my_est_1  noobs;
   format Estimate StdErr 8.5;
run;
```

Figure 20.73 ODS Parameter Estimates Table from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	StdErr	tValue	Probt
	TPM1_1	0.98912	0.00380	260.10	<.0001
	TPM1_2	0.00000	0.00000	0.00	0.9986
	TPM1_3	0.01088	0.00380	2.86	0.0043
	TPM2_1	0.00941	0.00543	1.73	0.0837
	TPM2_2	0.97953	0.00771	127.07	<.0001
	TPM2_3	0.01107	0.00595	1.86	0.0629
	TPM3_1	0.01012	0.00425	2.38	0.0173
	TPM3_2	0.01014	0.00425	2.39	0.0172
	TPM3_3	0.97974	0.00614	159.65	<.0001
aa email	CPM1_1	0.00000	0.00000	0.00	0.9974
bb mail	CPM1_2	0.00000	0.00001	0.00	0.9973
cc phone	CPM1_3	0.00000	0.00000	0.00	0.9984
dd flyer	CPM1_4	0.00000	0.00000	0.00	0.9986
ee seminar	CPM1_5	0.51608	0.01911	27.01	<.0001
ff forum	CPM1_6	0.48392	0.01911	25.33	<.0001
aa email	CPM2_1	0.00000	0.00001	0.00	0.9972
bb mail	CPM2_2	0.00000	0.00001	0.00	0.9987
cc phone	CPM2_3	0.48113	0.02802	17.17	<.0001
dd flyer	CPM2_4	0.51887	0.02802	18.52	<.0001
ee seminar	CPM2_5	0.00000	0.00009	0.00	0.9999
ff forum	CPM2_6	0.00000	0.00009	0.00	0.9999
aa email	CPM3_1	0.52209	0.02238	23.32	<.0001
bb mail	CPM3_2	0.47791	0.02238	21.35	<.0001
cc phone	CPM3_3	0.00000	0.00002	0.00	0.9999
dd flyer	CPM3_4	0.00000	0.00001	0.00	0.9997
ee seminar	CPM3_5	0.00000	0.00000	0.00	1.0000
ff forum	CPM3_6	0.00000	0.00001	0.00	0.9997

The following SAS code prints the output forecast table from the %FiniteHMM macro. The table is shown in Figure 20.74.

```
proc print data=mylib.fcst_1 label noobs;
   format State: Category: 8.5;
run;
```

Figure 20.74 Output Forecast Table from the %FiniteHMM Macro

section	t	Step	State1	State2	State3	aa email	bb mail	cc phone	dd flyer	ee seminar	ff forum
1	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
1	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
1	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
1	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350
2	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
2	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
2	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
2	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350
3	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
3	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
3	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
3	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350

The RespVarLevel column in Figure 20.73 provides information about the levels of response variables for the CPM parameters. The labels in Figure 20.74 clearly show the categorical levels of the response variable y_1.

Macro Example 3: Finite HMM for Responses with Nonconsecutive Natural Number Values

In this example, although y_1 has values of 2 and 6, PROC HMM still assumes that it has consecutive values of 1, 2, 3, 4, 5, and 6, and the output tables for CPM and the forecast will contain extra columns for the values 1, 3, 4, and 5. The %FiniteHMM macro will consider 2 and 6 as two categorical values, so it will give you the desired results.

The following SAS code creates the finite3 data set:

```
data finite3;
   set finite1;
   if y<=3 then y1=2; else y1=6;
run;
data mylib.finite3;
   set finite3;
run;
```

The following SAS code uses PROC HMM on y_1:

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite3 labelSwitch=(sort=asc(cpm));
   id time=t section=section;
   model y1 / type=finite nstate=3 nCategory=6 method=MAP;
   estimate out=mylib.est_0 outall=mylib.estall_0;
   forecast out=mylib.fcst_0 lead=4;
run;
```

The following SAS code prints the ODS CPM output table, parameter estimates table, and output forecast table, respectively, from PROC HMM. The results are shown in Figure 20.75, Figure 20.76, and Figure 20.77.

```
proc print data=my_cpm_0 label noobs;
   format  Category: 8.5;
run;

proc print data=my_est_0 label noobs;
   format Estimate StdErr 8.5;
run;

proc print data=mylib.fcst_0 label noobs;
   format State: Category: 8.5;
run;
```

Figure 20.75 ODS CPM Output Table from PROC HMM

Group	State	1	2	3	4	5	6
ParameterMatrices	1	0.00000	0.48869	0.00000	0.00000	0.00000	0.51131
ParameterMatrices	2	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
ParameterMatrices	3	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000

Figure 20.76 ODS Parameter Estimates Table from PROC HMM

Parameter	Estimate	Standard Error	t value	Pr > |t|
TPM1_1	0.97835	0.00819	119.47	<.0001
TPM1_2	0.01251	0.00727	1.72	0.0857
TPM1_3	0.00915	0.00634	1.44	0.1496
TPM2_1	0.01144	0.00482	2.37	0.0177
TPM2_2	0.97817	0.00666	146.93	<.0001
TPM2_3	0.01038	0.00465	2.23	0.0256
TPM3_1	0.00000	0.00001	0.00	0.9968
TPM3_2	0.01120	0.00393	2.85	0.0045
TPM3_3	0.98880	0.00393	251.34	<.0001
CPM1_1	0.00000	0.00000	0.00	0.9998
CPM1_2	0.48869	0.02941	16.62	<.0001
CPM1_3	0.00000	0.00000	0.00	0.9998
CPM1_4	0.00000	0.00000	0.00	0.9998
CPM1_5	0.00000	0.00000	0.00	0.9998
CPM1_6	0.51131	0.02941	17.39	<.0001
CPM2_1	0.00000	0.00000	0.00	0.9998
CPM2_2	1.00000	0.00000	1735239	<.0001
CPM2_3	0.00000	0.00000	0.00	1.0000
CPM2_4	0.00000	0.00000	0.00	1.0000
CPM2_5	0.00000	0.00000	0.00	1.0000
CPM2_6	0.00000	0.00000	0.00	1.0000
CPM3_1	0.00000	0.00000	0.00	0.9997
CPM3_2	0.00000	0.00000	0.00	0.9995
CPM3_3	0.00000	0.00000	0.00	0.9997
CPM3_4	0.00000	0.00000	0.00	0.9996
CPM3_5	0.00000	0.00000	0.00	0.9996
CPM3_6	1.00000	0.00000	636590	<.0001

Figure 20.77 Output Forecast Table from PROC HMM

section	t	Step	State1	State2	State3	Category1	Category2	Category3	Category4	Category5	Category6
1	500	1	0.06414	0.01129	0.92457	0.00000	0.04264	0.00000	0.00000	0.00000	0.95736
1	500	2	0.06288	0.02221	0.91491	0.00000	0.05294	0.00000	0.00000	0.00000	0.94706
1	500	3	0.06178	0.03276	0.90547	0.00000	0.06295	0.00000	0.00000	0.00000	0.93705
1	500	4	0.06081	0.04296	0.89623	0.00000	0.07268	0.00000	0.00000	0.00000	0.92732
2	500	1	0.06414	0.01129	0.92457	0.00000	0.04264	0.00000	0.00000	0.00000	0.95736
2	500	2	0.06288	0.02221	0.91491	0.00000	0.05294	0.00000	0.00000	0.00000	0.94706
2	500	3	0.06178	0.03276	0.90547	0.00000	0.06295	0.00000	0.00000	0.00000	0.93705
2	500	4	0.06081	0.04296	0.89623	0.00000	0.07268	0.00000	0.00000	0.00000	0.92732
3	500	1	0.06414	0.01129	0.92457	0.00000	0.04264	0.00000	0.00000	0.00000	0.95736
3	500	2	0.06288	0.02221	0.91491	0.00000	0.05294	0.00000	0.00000	0.00000	0.94706
3	500	3	0.06178	0.03276	0.90547	0.00000	0.06295	0.00000	0.00000	0.00000	0.93705
3	500	4	0.06081	0.04296	0.89623	0.00000	0.07268	0.00000	0.00000	0.00000	0.92732

If you use the %FiniteHMM macro on y_1, as in the following code, the results become concise and correct. Be aware that because the number of levels of the response variable differs in these three instances, the output state orders could be different even if you use the same LABELSWITCH= option value in both PROC HMM and the %FiniteHMM macro.

The following SAS program uses %FiniteHMM on y_1:

```
%FiniteHMM(
   data = %str(mylib.finite3),
   proc = %str(labelSwitch=(sort=asc(cpm))),
   id = %str(time=t section=section),
   model = %str(y1 / type=finite nstate=3 nCategory=6 method=MAP),
   estimate = %str( out=mylib.est_1 outall=mylib.estall_1),
   forecast = %str(out=mylib.fcst_1 lead=4),
   odsout = %str(CPM=my_cpm_1
                 ParameterEstimates=my_est_1));
```

The following three SAS programs print the ODS CPM output table, parameter estimates table, and output forecast table, respectively, from the %FiniteHMM macro. The results are shown in Figure 20.78, Figure 20.79, and Figure 20.80.

```
proc print data=my_cpm_1 label noobs;
   format  Category: 8.5;
run;

proc print data=my_est_1 label noobs;
   format Estimate StdErr 8.5;
run;

proc print data=mylib.fcst_1 label noobs;
   format State: Category: 8.5;
run;
```

Figure 20.78 ODS CPM Output Table from the %FiniteHMM Macro

Group	State	2	6
ParameterMatrices	1	0.00000	1.00000
ParameterMatrices	2	0.48869	0.51131
ParameterMatrices	3	1.00000	0.00000

Figure 20.79 ODS Parameter Estimates Table from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	Standard Error	t value	Pr > |t|
	TPM1_1	0.98880	0.00393	251.34	<.0001
	TPM1_2	0.00000	0.00000	0.00	0.9989
	TPM1_3	0.01120	0.00393	2.85	0.0045
	TPM2_1	0.00915	0.00634	1.44	0.1496
	TPM2_2	0.97835	0.00819	119.47	<.0001
	TPM2_3	0.01251	0.00727	1.72	0.0857
	TPM3_1	0.01038	0.00465	2.23	0.0256
	TPM3_2	0.01144	0.00482	2.37	0.0177
	TPM3_3	0.97817	0.00666	146.93	<.0001
2	CPM1_1	0.00000	0.00000	0.00	0.9990
6	CPM1_2	1.00000	0.00000	548383	<.0001
2	CPM2_1	0.48869	0.02941	16.62	<.0001
6	CPM2_2	0.51131	0.02941	17.39	<.0001
2	CPM3_1	1.00000	0.00001	113383	<.0001
6	CPM3_2	0.00000	0.00001	0.00	0.9967

Figure 20.80 Output Forecast Table from the %FiniteHMM Macro

section	t	Step	State1	State2	State3	2	6
1	500	1	0.92457	0.06414	0.01129	0.04264	0.95736
1	500	2	0.91491	0.06288	0.02221	0.05294	0.94706
1	500	3	0.90547	0.06178	0.03276	0.06295	0.93705
1	500	4	0.89623	0.06081	0.04296	0.07268	0.92732
2	500	1	0.92457	0.06414	0.01129	0.04264	0.95736
2	500	2	0.91491	0.06288	0.02221	0.05294	0.94706
2	500	3	0.90547	0.06178	0.03276	0.06295	0.93705
2	500	4	0.89623	0.06081	0.04296	0.07268	0.92732
3	500	1	0.92457	0.06414	0.01129	0.04264	0.95736
3	500	2	0.91491	0.06288	0.02221	0.05294	0.94706
3	500	3	0.90547	0.06178	0.03276	0.06295	0.93705
3	500	4	0.89623	0.06081	0.04296	0.07268	0.92732

Figure 20.75 shows that the CPM output table from PROC HMM has six columns, but only two columns contain useful information. Similarly, the parameter estimates table from PROC HMM also contains extra parameters. On the other hand, results from the %FiniteHMM macro contain information for two levels of the values 2 and 6 only. On a large scale, if y_1 has values of 1 and 100, then PROC HMM assumes that it has the consecutive values 1 through 100, and the CPM output table for y_1 has 100 columns, of which only two

columns contain useful information. The %FiniteHMM macro gives you results on two categories, 1 and 100, and is much more efficient. The RespVarLevel column in Figure 20.79 provides information about the levels of response variables for the CPM parameters. The labels in Figure 20.80 clearly show the categorical levels of the response variable y_1.

Macro Example 4: Automatic Handling of the NCATEGORY= Option in the %FiniteHMM Macro

In this example, y_1 has integer values of 1, 3, 4, and 6. In PROC HMM, you get an error message if the value of the NCATEGORY= option is less than the maximum value of y_1, which is 6. On the other hand, if the NCATEGORY= option value is greater than the maximum value of y_1, PROC HMM assumes that y_1 has consecutive values from 1 to the NCATEGORY= option value. Hence the results will contain extra columns or parameters similar to those in Macro Example 3.

If you apply the %FiniteHMM macro, the NCATEGORY= option value is automatically updated to the number of levels of the response variable. There will be no error message if the NCATEGORY= option value is less than the maximum value of the response variable, and the results are more desirable.

The following SAS code creates the finite4 data set:

```
data finite4;
   set finite1;
   y1 = y;
   if y=2 then y1=1;
   if y=5 then y1=4;
run;
data mylib.finite4;
   set finite4;
run;
```

The following SAS code specifies NCATEGORY=4. PROC HMM generates an error message.

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite4 labelSwitch=(sort=asc(cpm));
   id time=t section=section;
   model y1 / type=finite nstate=3 nCategory=4 method=MAP;
   estimate out=mylib.est_0 outall=mylib.estall_0;
   forecast out=mylib.fcst_0 lead=4;
run;
ERROR: The dependent variable for the finite HMM must be an integer between 1
       and the number of the categories which is specified in the NCATEGORY= option
       in the MODEL statement. To correct this, you can either preprocess your data
       or specify a larger value in the NCATEGORY= option.
```

The following SAS code specifies NCATEGORY=8. PROC HMM assumes that y_1 has the consecutive integer values 1 through 8.

```
ods output CPM=my_cpm_0
           ParameterEstimates=my_est_0;
proc hmm data=mylib.finite4 labelSwitch=(sort=asc(cpm));
   id time=t section=section;
   model y1 / type=finite nstate=3 nCategory=8 method=MAP;
   estimate out=mylib.est_0 outall=mylib.estall_0;
   forecast out=mylib.fcst_0 lead=4;
```

```
run;
```

The following SAS code prints the ODS CPM output table from PROC HMM when NCATEGORY=8. The table is shown in Figure 20.81.

```
proc print data=my_cpm_0 label noobs;
format Category: 8.5;
run;
```

Figure 20.81 ODS CPM Output Table from PROC HMM When NCATEGORY=8

Group	State	1	2	3	4	5	6	7	8
ParameterMatrices	1	0.00000	0.00000	0.00000	0.51379	0.00000	0.48621	0.00000	0.00000
ParameterMatrices	2	0.00000	0.00000	0.47630	0.52370	0.00000	0.00000	0.00000	0.00000
ParameterMatrices	3	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

If you use the %FiniteHMM macro as in the following program, the NCATEGORY= option value is always updated to the actual number of levels of the response variable and a message is displayed. The ODS CPM output tables when NCATEGORY=4 and 8 are shown in Figure 20.82 and Figure 20.83, respectively.

```
%FiniteHMM(
   data = %str(mylib.finite4),
   proc = %str(labelSwitch=(sort=asc(cpm))),
   id = %str(time=t section=section),
   model = %str(y1 / type=finite nstate=3 nCategory=4 method=MAP),
   estimate = %str( out=mylib.est_1 outall=mylib.estall_1),
   forecast = %str(out=mylib.fcst_1 lead=4),
   odsout = %str(CPM=my_cpm_1
                 ParameterEstimates=my_est_1));

NOTE: The NCATEGORY= option value is always updated to the number of levels
      of the response variable.

%FiniteHMM(
   data = %str(mylib.finite4),
   proc = %str(labelSwitch=(sort=asc(cpm))),
   id = %str(time=t section=section),
   model = %str(y1 / type=finite nstate=3 nCategory=8 method=MAP),
   estimate = %str( out=mylib.est_2 outall=mylib.estall_2),
   forecast = %str(out=mylib.fcst_2 lead=4),
   odsout = %str(CPM=my_cpm_2
                 ParameterEstimates=my_est_2));

NOTE: The NCATEGORY= option value is always updated to the number of levels
      of the response variable.
```

The following SAS code prints the ODS CPM output table from the %FiniteHMM macro when NCATEGORY=4. The table is shown in Figure 20.82.

```
proc print data=my_cpm_1 label noobs;
format Category: 8.5;
run;
```

Figure 20.82 ODS CPM Table from the %FiniteHMM Macro When NCATEGORY=4

Group	State	1	3	4	6
ParameterMatrices	1	0.00000	0.47630	0.52370	0.00000
ParameterMatrices	2	0.00000	0.00000	0.51379	0.48621
ParameterMatrices	3	1.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS CPM output table from the %FiniteHMM macro when NCATEGORY=8. The table is shown in Figure 20.83.

```
proc print data=my_cpm_2 label noobs;
format Category: 8.5;
run;
```

Figure 20.83 ODS CPM Table from the %FiniteHMM Macro When NCATEGORY=8

Group	State	1	3	4	6
ParameterMatrices	1	0.00000	0.47630	0.52370	0.00000
ParameterMatrices	2	0.00000	0.00000	0.51379	0.48621
ParameterMatrices	3	1.00000	0.00000	0.00000	0.00000

Macro Example 5: Finite HMM with Decimal-Valued or Negative-Valued Responses

In this example, the response variable y_1 has decimal values, and the response variable y_2 has values of negative integers. PROC HMM generates an error message for both y_1 and y_2. But using the %FiniteHMM macro gives you correct and complete results.

The following SAS code creates the finite5 data set:

```
data finite5;
   set finite1;
   y1 = y/100;
   y2 = -y;
run;
data mylib.finite5;
   set finite5;
run;
```

Using PROC HMM on y_1 and y_2 as follows generates error messages:

```
ods output CPM=my_cpm_1
           ParameterEstimates=my_est_1;
proc hmm data=mylib.finite5 labelSwitch=(sort=asc(cpm));
   id time=t section=section;
   model y1 / type=finite nstate=3 nCategory=6 method=MAP;
   estimate out=mylib.est_1 outall=mylib.estall_1;
   forecast out=mylib.fcst_1 lead=4;
run;
ERROR: For finite HMM, the series variable must be integer-valued.

ods output CPM=my_cpm_2
           ParameterEstimates=my_est_2;
```

```
proc hmm data=mylib.finite5 labelSwitch=(sort=asc(cpm));
   id time=t section=section;
   model y2 / type=finite nstate=3 nCategory=6 method=MAP;
   estimate out=mylib.est_2 outall=mylib.estall_2;
   forecast out=mylib.fcst_2 lead=4;
run;
ERROR: The dependent variable for the finite HMM must be an integer between 1
       and the number of the categories which is specified in the NCATEGORY=
       option in the MODEL statement. To correct this, you can either
       preprocess your data or specify a larger value in the NCATEGORY= option.
```

Using the %FiniteHMM macro on y_1 and y_2 as follows provides the expected results:

```
%FiniteHMM(
   data = %str(mylib.finite5),
   proc = %str(labelSwitch=(sort=asc(cpm))),
   id = %str(time=t section=section),
   model = %str(y1 / type=finite nstate=3 nCategory=6 method=MAP),
   estimate = %str( out=mylib.est_1 outall=mylib.estall_1),
   forecast = %str(out=mylib.fcst_1 lead=4),
   odsout = %str(CPM=my_cpm_1
                 ParameterEstimates=my_est_1));
%FiniteHMM(
   data = %str(mylib.finite5),
   proc = %str(labelSwitch=(sort=asc(cpm))),
   id = %str(time=t section=section),
   model = %str(y2 / type=finite nstate=3 nCategory=6 method=MAP),
   estimate = %str( out=mylib.est_2 outall=mylib.estall_2),
   forecast = %str(out=mylib.fcst_2 lead=4),
   odsout = %str(CPM=my_cpm_2
                 ParameterEstimates=my_est_2));
```

The following SAS code prints the ODS CPM output table of y_1 from the %FiniteHMM macro. The table is shown in Figure 20.84.

```
proc print data=my_cpm_1 label noobs;
   format Category: 8.5;
run;
```

Figure 20.84 ODS CPM Output Table of y1 from the %FiniteHMM Macro

Group	State	0.01	0.02	0.03	0.04	0.05	0.06
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.51608	0.48392
ParameterMatrices	2	0.00000	0.00000	0.48113	0.51887	0.00000	0.00000
ParameterMatrices	3	0.52209	0.47791	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS parameter estimates table of y_1 from the %FiniteHMM macro. The table is shown in Figure 20.85.

```
proc print data=my_est_1 label noobs;
   format Estimate StdErr 8.5;
run;
```

Figure 20.85 ODS Parameter Estimates Table of y1 from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	Standard Error	t value	Pr > |t|
	TPM1_1	0.98912	0.00380	260.10	<.0001
	TPM1_2	0.00000	0.00000	0.00	0.9986
	TPM1_3	0.01088	0.00380	2.86	0.0043
	TPM2_1	0.00941	0.00543	1.73	0.0837
	TPM2_2	0.97953	0.00771	127.07	<.0001
	TPM2_3	0.01107	0.00595	1.86	0.0629
	TPM3_1	0.01012	0.00425	2.38	0.0173
	TPM3_2	0.01014	0.00425	2.39	0.0172
	TPM3_3	0.97974	0.00614	159.65	<.0001
0.01	CPM1_1	0.00000	0.00000	0.00	0.9974
0.02	CPM1_2	0.00000	0.00001	0.00	0.9973
0.03	CPM1_3	0.00000	0.00000	0.00	0.9984
0.04	CPM1_4	0.00000	0.00000	0.00	0.9986
0.05	CPM1_5	0.51608	0.01911	27.01	<.0001
0.06	CPM1_6	0.48392	0.01911	25.33	<.0001
0.01	CPM2_1	0.00000	0.00001	0.00	0.9972
0.02	CPM2_2	0.00000	0.00001	0.00	0.9987
0.03	CPM2_3	0.48113	0.02802	17.17	<.0001
0.04	CPM2_4	0.51887	0.02802	18.52	<.0001
0.05	CPM2_5	0.00000	0.00009	0.00	0.9999
0.06	CPM2_6	0.00000	0.00009	0.00	0.9999
0.01	CPM3_1	0.52209	0.02238	23.32	<.0001
0.02	CPM3_2	0.47791	0.02238	21.35	<.0001
0.03	CPM3_3	0.00000	0.00002	0.00	0.9999
0.04	CPM3_4	0.00000	0.00001	0.00	0.9997
0.05	CPM3_5	0.00000	0.00000	0.00	1.0000
0.06	CPM3_6	0.00000	0.00001	0.00	0.9997

The following SAS code prints the output forecast table of y_1 from the %FiniteHMM macro. The table is shown in Figure 20.86.

```
proc print data=mylib.fcst_1 label noobs;
   format State: Category: 8.5;
run;
```

Figure 20.86 Output Forecast Table of y1 from the %FiniteHMM Macro

section	t	Step	State1	State2	State3	0.01	0.02	0.03	0.04	0.05	0.06
1	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
1	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
1	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
1	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350
2	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
2	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
2	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
2	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350
3	500	1	0.98912	0.00000	0.01088	0.00568	0.00520	0.00000	0.00000	0.51046	0.47865
3	500	2	0.97846	0.00011	0.02143	0.01119	0.01024	0.00005	0.00006	0.50497	0.47349
3	500	3	0.96803	0.00033	0.03165	0.01652	0.01512	0.00016	0.00017	0.49958	0.46845
3	500	4	0.95781	0.00064	0.04155	0.02169	0.01986	0.00031	0.00033	0.49431	0.46350

The following SAS code prints the ODS CPM output table of y_2 from the %FiniteHMM macro. The table is shown in Figure 20.87.

```
proc print data=my_cpm_2 label noobs;
   format Category: 8.5;
run;
```

Figure 20.87 ODS CPM Output Table of y2 from the %FiniteHMM Macro

Group	State	-6	-5	-4	-3	-2	-1
ParameterMatrices	1	0.00000	0.00000	0.00000	0.00000	0.47791	0.52209
ParameterMatrices	2	0.00000	0.00000	0.51887	0.48113	0.00000	0.00000
ParameterMatrices	3	0.48392	0.51608	0.00000	0.00000	0.00000	0.00000

The following SAS code prints the ODS parameter estimates table of y_2 from the %FiniteHMM macro. The table is shown in Figure 20.88.

```
proc print data=my_est_2 label noobs;
   format Estimate StdErr 8.5;
run;
```

Figure 20.88 ODS Parameter Estimates Table of y2 from the %FiniteHMM Macro

RespVarLevel	Parameter	Estimate	Standard Error	t value	Pr > |t|
	TPM1_1	0.97974	0.00614	159.65	<.0001
	TPM1_2	0.01014	0.00425	2.39	0.0172
	TPM1_3	0.01012	0.00425	2.38	0.0173
	TPM2_1	0.01107	0.00595	1.86	0.0629
	TPM2_2	0.97953	0.00771	127.07	<.0001
	TPM2_3	0.00941	0.00543	1.73	0.0837
	TPM3_1	0.01088	0.00380	2.86	0.0043
	TPM3_2	0.00000	0.00000	0.00	0.9996
	TPM3_3	0.98912	0.00380	260.10	<.0001
-6	CPM1_1	0.00000	0.00000	0.00	0.9996
-5	CPM1_2	0.00000	0.00000	0.00	0.9996
-4	CPM1_3	0.00000	0.00001	0.00	0.9973
-3	CPM1_4	0.00000	0.00000	0.00	0.9985
-2	CPM1_5	0.47791	0.02238	21.35	<.0001
-1	CPM1_6	0.52209	0.02238	23.32	<.0001
-6	CPM2_1	0.00000	0.00000	0.00	0.9989
-5	CPM2_2	0.00000	0.00000	0.00	0.9991
-4	CPM2_3	0.51887	0.02802	18.52	<.0001
-3	CPM2_4	0.48113	0.02802	17.17	<.0001
-2	CPM2_5	0.00000	0.00000	0.00	1.0000
-1	CPM2_6	0.00000	0.00000	0.00	1.0000
-6	CPM3_1	0.48392	0.01911	25.33	<.0001
-5	CPM3_2	0.51608	0.01911	27.01	<.0001
-4	CPM3_3	0.00000	0.00000	0.00	0.9994
-3	CPM3_4	0.00000	0.00004	0.00	1.0000
-2	CPM3_5	0.00000	0.00000	0.00	1.0000
-1	CPM3_6	0.00000	0.00004	0.00	1.0000

The following SAS code prints the output forecast table of y_2 from the %FiniteHMM macro. The table is shown in Figure 20.89.

```
proc print data=mylib.fcst_2 label noobs;
   format State: Category: 8.5;
run;
```

Figure 20.89 Output Forecast Table of y2 from the %FiniteHMM Macro

section	t	Step	State1	State2	State3	-6	-5	-4	-3	-2	-1
1	500	1	0.01088	0.00000	0.98912	0.47865	0.51046	0.00000	0.00000	0.00520	0.00568
1	500	2	0.02143	0.00011	0.97846	0.47349	0.50497	0.00006	0.00005	0.01024	0.01119
1	500	3	0.03165	0.00033	0.96803	0.46845	0.49958	0.00017	0.00016	0.01512	0.01652
1	500	4	0.04155	0.00064	0.95781	0.46350	0.49431	0.00033	0.00031	0.01986	0.02169
2	500	1	0.01088	0.00000	0.98912	0.47865	0.51046	0.00000	0.00000	0.00520	0.00568
2	500	2	0.02143	0.00011	0.97846	0.47349	0.50497	0.00006	0.00005	0.01024	0.01119
2	500	3	0.03165	0.00033	0.96803	0.46845	0.49958	0.00017	0.00016	0.01512	0.01652
2	500	4	0.04155	0.00064	0.95781	0.46350	0.49431	0.00033	0.00031	0.01986	0.02169
3	500	1	0.01088	0.00000	0.98912	0.47865	0.51046	0.00000	0.00000	0.00520	0.00568
3	500	2	0.02143	0.00011	0.97846	0.47349	0.50497	0.00006	0.00005	0.01024	0.01119
3	500	3	0.03165	0.00033	0.96803	0.46845	0.49958	0.00017	0.00016	0.01512	0.01652
3	500	4	0.04155	0.00064	0.95781	0.46350	0.49431	0.00033	0.00031	0.01986	0.02169

Poisson Hidden Markov Model

The Poisson hidden Markov model (Poisson HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the Poisson distribution whose support is nonnegative integers. The observation y_t is said to have a Poisson distribution, with the parameter $\lambda_{\mathbf{S}_t} > 0$, if it has a probability mass function given by

$$p(y_t = k|\mathbf{S}_t) = \frac{e^{-\lambda_{s_t}} \lambda_{s_t}^k}{k!}$$

where the variable $\mathbf{S}_t$ is the (hidden) state, k is the number of occurrences, and ! is the factorial function. The positive real number $\lambda_{\mathbf{S}_t}$ is equal to the expected value of y_t and also to its variance in its hidden state.

Hence, the initial state probability vector (ISPV) π, the transition probability matrix (TPM) $\mathbf{A}$, and the probability of observational outcomes $\mathbf{B}$ ($\equiv \{b_{ij}, i = 1, \ldots, K, j = 1, \ldots, M\}$) together specify the Poisson HMM; that is, the parameter θ of the Poisson HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the Poisson HMM is a type of finite-state-space and homogeneous HMM, you can solve the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—by using the three algorithms that are introduced in the section "Hidden Markov Model" on page 1230. That is, you can solve the filtering, forecasting, and evaluating problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood. The dependent variable for the finite HMM must be a nonnegative integer. The number of dependent variables for the finite HMM must be one.

Parameter Estimation Methods

For the parameter estimation (the learning problem) of the HMM, the HMM procedure supports two methods: the maximum likelihood (ML) method and the maximum a posteriori (MAP) method.

Maximum Likelihood (ML) Method

For the ML method, the parameter estimator is obtained as

$$\hat{\theta} = \arg\max_{\theta} \log\left(L_T(\mathbf{Y}_1, \ldots, \mathbf{Y}_T; \theta)\right)$$

where $L_T(\cdot; \theta)$ is the likelihood of all observations for a particular parameter θ. The $\hat{\theta}$ is obtained through gradient-based optimization; the covariance matrix estimator of the $\hat{\theta}$ is obtained through the inverse of the negative Hessian.

Maximum a Posteriori (MAP) Method

For the MAP method, the parameter estimator is obtained as

$$\hat{\theta} = \arg\max_{\theta} \log\left(L_T(\mathbf{Y}_1, \ldots, \mathbf{Y}_T; \theta)\right) + \log\left(p(\theta)\right)$$

where $p(\theta)$ is the prior distribution function of parameter θ. The prior distribution can be specified in the PRIOR statement. The $\hat{\theta}$ is obtained through gradient-based optimization; the covariance matrix estimator of the $\hat{\theta}$ is obtained through the inverse of the negative Hessian.

In theory, the likelihood of the hidden Markov model (HMM) is unbounded and "the ML estimator as a global maximizer of the likelihood function does not exist" (Frühwirth-Schnatter 2006, page 173). The introduction of the proper prior distribution of the parameters in the MAP method can solve the unboundedness problem.

Prior Distribution of the Initial State Probability Vector

The prior distribution of the initial state probability vector (ISPV), π, is the Dirichlet distribution,

$$\pi \sim \mathrm{Dir}(\alpha)$$

where the hyperparameter α is the $K \times 1$ vector and K is the number of states. The probability density function (PDF) of the Dirichlet distribution is

$$f(x, \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

where $x = (x_1 \cdots x_K)'$, $x_i > 0, i = 1, \ldots, K$, and $\sum_{i=1}^{K} x_i = 1$; $\alpha = (\alpha_1 \cdots \alpha_K)'$ and $\alpha_i > 0, i = 1, \ldots, K$; $B(\alpha)$ is the beta function,

$$B(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)}$$

Prior Distribution of the Transition Probability Matrix

The prior distribution of each row of the transition probability matrix (TPM), **A**, is the Dirichlet distribution,

$$a'_{i\cdot} \sim \text{Dir}(\alpha^{(i)}), i = 1, \ldots, K$$

where $a_i\cdot$ is the ith row of the TPM **A**, $\alpha^{(i)}$ is the $K \times 1$ vector, and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{A} \sim \text{Dir}(\tilde{\mathbf{A}})$$

where the hyperparameter $\tilde{\mathbf{A}} = (\alpha^{(1)} \cdots \alpha^{(K)})'$, which is the $K \times K$ matrix that results from stacking the transpose of $\alpha^{(i)}, i = 1, \ldots, K$.

Prior Distribution of the Category Probability Matrix

The prior distribution of each row of the category probability matrix (CPM), **B**, is the Dirichlet distribution,

$$b'_{i\cdot} \sim \text{Dir}(b^{(i)}), i = 1, \ldots, K$$

where $b_i\cdot$ is the ith row of the CPM, **B**; $b^{(i)}$ is the $M \times 1$ vector; M is the number of categories at a state; and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{B} \sim \text{Dir}(\tilde{\mathbf{B}})$$

where the hyperparameter $\tilde{\mathbf{B}} = (b^{(1)} \cdots b^{(K)})'$, which is the $K \times M$ matrix that results from stacking the transpose of $b^{(i)}, i = 1, \ldots, K$.

Prior Distribution of the Mixture Component Probabilities

The prior distribution of each row of the mixture component probabilities (MCP), **W**, is the Dirichlet distribution,

$$w'_{i\cdot} \sim \text{Dir}(\alpha^{(i)}), i = 1, \ldots, K$$

where $w_i\cdot$ is the ith row of the MCP **W**, $\alpha^{(i)}$ is the $M \times 1$ vector, M is the number of components at a state, and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{W} \sim \text{Dir}(\tilde{\mathbf{A}})$$

where the hyperparameter $\tilde{\mathbf{A}} = (\alpha^{(1)} \cdots \alpha^{(K)})'$, which is the $K \times M$ matrix that results from stacking the transpose of $\alpha^{(i)}, i = 1, \ldots, K$.

Prior Distribution of the Observation Parameters

Gaussian HMM In the Gaussian HMM, the observation parameters are $\{\mu_i, \Sigma_i\}, i = 1, \ldots, K$, where μ_i and Σ_i are the mean and covariance parameters for state i. The prior distribution of μ_i and Σ_i is the normal-inverse-Wishart (NIW) distribution,

$$\mu_i, \Sigma_i \sim \mathrm{NIW}(\tilde{\mu}_i, \tilde{\Sigma}_i, \tilde{\kappa}_i, \tilde{\nu}_i), i = 1, \ldots, K$$

which is equivalent to

$$\mu_i | \Sigma_i \sim \mathrm{N}(\tilde{\mu}_i, \tilde{\kappa}_i^{-1} \Sigma_i)$$
$$\Sigma_i \sim \mathrm{IW}(\tilde{\Sigma}_i, \tilde{\nu}_i)$$

where the hyperparameter $\tilde{\mu}_i$ is a $k_y \times 1$ vector; the hyperparameter $\tilde{\Sigma}_i$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_i$ is a positive scalar; the hyperparameter $\tilde{\nu}_i$ is a positive scalar, $\tilde{\nu}_i > k_y + 3$; K is the number of states; and k_y is the number of dependent variables.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned}
f_{\mathrm{NIW}}(\mu, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_{\mathrm{N}}(\mu, \tilde{\mu}, \tilde{\kappa}^{-1}\Sigma) f_{\mathrm{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\
f_{\mathrm{N}}(\mu, \tilde{\mu}, \tilde{\kappa}^{-1}\Sigma) &= \frac{1}{\sqrt{(2\pi)^k |\tilde{\kappa}^{-1}\Sigma|}} \exp(-\frac{1}{2}(\mu - \tilde{\mu})'(\tilde{\kappa}^{-1}\Sigma)^{-1}(\mu - \tilde{\mu})) \\
f_{\mathrm{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu}k}{2}} \Gamma_k(\frac{\tilde{\nu}}{2})} |\Sigma|^{-\frac{\tilde{\nu}+k+1}{2}} \exp(-\frac{1}{2}\mathrm{tr}(\tilde{\Sigma}\Sigma^{-1}))
\end{aligned}$$

where the parameter μ is a $k \times 1$ vector; the parameter Σ is a $k \times k$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k \times k$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a positive scalar; and the hyperparameter $\tilde{\nu}$ is a positive scalar, $\tilde{\nu} > k + 3$. The $\Gamma_k(.)$ is the multivariate gamma function, which is defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^{k} \Gamma(x + \frac{1-j}{2})$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned}
\tilde{\mu} &= (\tilde{\mu}_1 \cdots \tilde{\mu}_K)' \\
\tilde{\Sigma} &= (\tilde{\Sigma}_1 \cdots \tilde{\Sigma}_K)' \\
\tilde{\kappa} &= (\tilde{\kappa}_1 \cdots \tilde{\kappa}_K)' \\
\tilde{\nu} &= (\tilde{\nu}_1 \cdots \tilde{\nu}_K)'
\end{aligned}$$

where $\tilde{\mu}$ is a $K \times k_y$ matrix, $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix, $\tilde{\kappa}$ is a $K \times 1$ vector, $\tilde{\nu}$ is a $K \times 1$ vector, K is the number of states, and k_y is the number of dependent variables.

GM HMM In the GM HMM, the mean and covariance parameters for component j at state i are $\{\mu_{ij}, \Sigma_{ij}\}, i = 1, \ldots, K, j = 1, \ldots, M$, where M is the number of components at a state and K is the number of states.

The prior distribution of μ_{ij} and Σ_{ij} is the normal-inverse-Wishart (NIW) distribution,

$$\mu_{ij}, \Sigma_{ij} \sim \text{NIW}(\tilde{\mu}_{ij}, \tilde{\Sigma}_{ij}, \tilde{\kappa}_{ij}, \tilde{\nu}_{ij}), i = 1, \ldots, K, j = 1, \ldots, M$$

which is equivalent to

$$\begin{aligned}
\mu_{ij}|\Sigma_{ij} &\sim \text{N}(\tilde{\mu}_{ij}, \tilde{\kappa}_{ij}^{-1}\Sigma_{ij}) \\
\Sigma_{ij} &\sim \text{IW}(\tilde{\Sigma}_{ij}, \tilde{\nu}_{ij})
\end{aligned}$$

where the hyperparameter $\tilde{\mu}_{ij}$ is a $k_y \times 1$ vector; the hyperparameter $\tilde{\Sigma}_{ij}$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_{ij}$ is a positive scalar; the hyperparameter $\tilde{\nu}_{ij}$ is a positive scalar, $\tilde{\nu}_i > k_y + 3$; and k_y is the number of dependent variables.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned}
f_{\text{NIW}}(\mu, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_{\text{N}}(\mu, \tilde{\mu}, \tilde{\kappa}^{-1}\Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\
f_{\text{N}}(\mu, \tilde{\mu}, \tilde{\kappa}^{-1}\Sigma) &= \frac{1}{\sqrt{(2\pi)^k |\tilde{\kappa}^{-1}\Sigma|}} \exp(-\frac{1}{2}(\mu - \tilde{\mu})'(\tilde{\kappa}^{-1}\Sigma)^{-1}(\mu - \tilde{\mu})) \\
f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu}k}{2}} \Gamma_k(\frac{\tilde{\nu}}{2})} |\Sigma|^{-\frac{\tilde{\nu}+k+1}{2}} \exp(-\frac{1}{2}\text{tr}(\tilde{\Sigma}\Sigma^{-1}))
\end{aligned}$$

where the parameter μ is a $k \times 1$ vector; the parameter Σ is a $k \times k$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k \times k$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a positive scalar; and the hyperparameter $\tilde{\nu}$ is a positive scalar, $\tilde{\nu} > k + 3$. The $\Gamma_k(.)$ is the multivariate gamma function defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^{k} \Gamma(x + \frac{1-j}{2})$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned}
\tilde{\mu} &= (\tilde{\mu}_{11} \cdots \tilde{\mu}_{1M} \tilde{\mu}_{21} \cdots \tilde{\mu}_{2M} \cdots \tilde{\mu}_{K1} \cdots \tilde{\mu}_{KM})' \\
\tilde{\Sigma} &= (\tilde{\Sigma}_{11} \cdots \tilde{\Sigma}_{1M} \tilde{\Sigma}_{21} \cdots \tilde{\Sigma}_{2M} \cdots \tilde{\Sigma}_{K1} \cdots \tilde{\Sigma}_{KM})' \\
\tilde{\kappa} &= (\tilde{\kappa}_{11} \cdots \tilde{\kappa}_{1M} \tilde{\kappa}_{21} \cdots \tilde{\kappa}_{2M} \cdots \tilde{\kappa}_{K1} \cdots \tilde{\kappa}_{KM})' \\
\tilde{\nu} &= (\tilde{\nu}_{11} \cdots \tilde{\nu}_{1M} \tilde{\nu}_{21} \cdots \tilde{\nu}_{2M} \cdots \tilde{\nu}_{K1} \cdots \tilde{\nu}_{KM})'
\end{aligned}$$

where $\tilde{\mu}$ is a $KM \times k_y$ matrix, $\tilde{\Sigma}$ is a $k_y KM \times k_y$ matrix, $\tilde{\kappa}$ is a $KM \times 1$ vector, $\tilde{\nu}$ is a $KM \times 1$ vector, M is the number of components at a state, K is the number of states, and k_y is the number of dependent variables.

Regime-Switching Regression Model and Regime-Switching Autoregression Model In regime-switching regression models and regime-switching autoregression models, the observation parameters are $\{\mathbf{B}_i, \Sigma_i\}, i = 1, \ldots, K$, where $\mathbf{B}_i$ and Σ_i are the mean and covariance parameters for state i. Because $\mathbf{B}_i z_t = (z_t' \otimes I_{k_y})\mathrm{vec}(\mathbf{B}_i) = (z_t' \otimes I_{k_y})\beta_i$, where vec is the column stacking operator, the observation parameters in this section are denoted by $\{\beta_i, \Sigma_i\}, i = 1, \ldots, K$. The prior distribution of β_i and Σ_i is the normal-inverse-Wishart (NIW) distribution,

$$\beta_i, \Sigma_i \sim \mathrm{NIW}(\tilde{\mu}_i, \tilde{\Sigma}_i, \tilde{\kappa}_i, \tilde{\nu}_i), i = 1, \ldots, K$$

which is equivalent to

$$\beta_i | \Sigma_i \sim \mathrm{N}(\tilde{\mu}_i, \tilde{\kappa}_i^{-1} \otimes \Sigma_i)$$
$$\Sigma_i \sim \mathrm{IW}(\tilde{\Sigma}_i, \tilde{\nu}_i)$$

where the hyperparameter $\tilde{\mu}_i$ is a $k_y k_z \times 1$ vector; the hyperparameter $\tilde{\Sigma}_i$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_i$ is a $k_z \times k_z$ symmetric positive definite matrix; the hyperparameter $\tilde{\nu}_i$ is a positive scalar, $\tilde{\nu}_i > k_y + 3$; K is the number of states; k_y is the number of dependent variables; and k_z is the number of regressors. For information about how to calculate k_z, see the sections "Regime-Switching Regression Model" on page 1235 and "Regime-Switching Autoregression Model" on page 1236.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned}
f_{\mathrm{NIW}}(\beta, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_{\mathrm{N}}(\beta, \tilde{\mu}, \tilde{\kappa}^{-1} \otimes \Sigma) f_{\mathrm{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\
f_{\mathrm{N}}(\beta, \tilde{\mu}, \tilde{\kappa}^{-1} \otimes \Sigma) &= \frac{1}{\sqrt{(2\pi)^{k_1 k_2} |\tilde{\kappa}^{-1} \otimes \Sigma|}} \exp(-\frac{1}{2}(\beta - \tilde{\mu})'(\tilde{\kappa}^{-1} \otimes \Sigma)^{-1}(\beta - \tilde{\mu})) \\
f_{\mathrm{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu} k_1}{2}} \Gamma_{k_1}(\frac{\tilde{\nu}}{2})} |\Sigma|^{-\frac{\tilde{\nu}+k_1+1}{2}} \exp(-\frac{1}{2}\mathrm{tr}(\tilde{\Sigma}\Sigma^{-1}))
\end{aligned}$$

where the parameter β is a $k_1 k_2 \times 1$ vector; the parameter Σ is a $k_1 \times k_1$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k_1 k_2 \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k_1 \times k_1$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a $k_2 \times k_2$ symmetric positive definite matrix; and the hyperparameter $\tilde{\nu}$ is a positive scalar, where $\tilde{\nu} > k_1 + 3$. The $\Gamma_k(.)$ is the multivariate gamma function defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^{k} \Gamma(x + \frac{1-j}{2})$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned}
\tilde{\mu} &= (\tilde{\mu}_1 \cdots \tilde{\mu}_K)' \\
\tilde{\Sigma} &= (\tilde{\Sigma}_1 \cdots \tilde{\Sigma}_K)' \\
\tilde{\kappa} &= (\tilde{\kappa}_1 \cdots \tilde{\kappa}_K)' \\
\tilde{\nu} &= (\tilde{\nu}_1 \cdots \tilde{\nu}_K)'
\end{aligned}$$

where $\tilde{\mu}$ is a $K \times k_y k_z$ matrix, $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix, $\tilde{\kappa}$ is a $k_z K \times k_z$ matrix, $\tilde{\nu}$ is a $K \times 1$ vector, K is the number of states, k_y is the number of dependent variables, and k_z is the number of regressors.

Poisson HMM In the Poisson HMM, the mean and variance for state i are both equal to $\lambda_i, i = 1, \ldots, K$. The prior distribution of λ_i, the gamma distribution, is

$$\lambda \sim \text{gamma}(\alpha, \beta)$$

where the hyperparameter α, β is the $K \times 2$ matrix and K is the number of states. The probability density function (PDF) of the gamma distribution is

$$f(\lambda) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} \lambda^{(\alpha-1)} \exp(-\beta\lambda)$$

where $\lambda > 0$.

Optimization Algorithms

The HMM procedure supports four types of optimization algorithms: the expectation-maximization (EM) algorithm, the active-set algorithm, the interior point algorithm, and the stochastic gradient descent (SGD) algorithm.

The EM algorithm is an iterative method of finding the local maximum likelihood estimates of parameters in models. After initializing the parameters—the ISPV π, the TPM A, the mean μ, and the covariance Σ—the EM algorithm iterates between two steps: the expectation (E) step and the maximization (M) step.

In the E step, γ_t and ξ_t are computed using α_t and β_t, which are based on the forward and backward algorithm:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{K} \alpha_t(j)\beta_t(j)}$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}\, p(y_{t+1}|s_{t+1} = j)\beta_{t+1}(j)}{\sum_{i=1}^{K}\sum_{i=1}^{K} \alpha_t(i)a_{ij}\, p(y_{t+1}|s_{t+1} = j)\beta_{t+1}(j)}$$

where $i, j \in [1, K]$. In the M step, the parameters are updated using γ_t and ξ_t from the E step: When ESTISPV is specified, ISPV is updated according to $\pi_i = \gamma_1(i)$. For finite-state-space and homogeneous HMM, the stationary distribution of the Markov chain is the value of π such that $\pi' A = \pi'$, where A is the transition probability matrix (TPM); that is, π is the leading left eigenvector of A. Other parameters are updated according to the following equations, if there are no sections in the data:

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\mu_i = \frac{\sum_{t=1}^{T} y_t * \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

$$\Sigma_i = \frac{\sum_{t=1}^{T} (y_t - \mu_i) * (y_t - \mu_i)^T * \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

Otherwise, the preceding equations become

$$\pi_i = \frac{\sum_{s=1}^{\texttt{nsec}} \gamma_1^{(s)}(i)}{\texttt{nsec}}$$

$$a_{ij} = \frac{\sum_{s=1}^{\texttt{nsec}} \sum_{t=1}^{T^{(s)}-1} \xi_t(i,j)}{\sum_{s=1}^{\texttt{nsec}} \sum_{t=1}^{T^{(s)}-1} \gamma_t^{(s)}(i)}$$

$$\mu_i = \frac{\sum_{s=1}^{\texttt{nsec}} \sum_{t=1}^{T^{(s)}} y_t * \gamma_t^{(s)}(i)}{\sum_{s=1}^{\texttt{nsec}} \sum_{t=1}^{T^{(s)}} \gamma_t^{(s)}(i)}$$

$$\Sigma_i = \frac{\sum_{s=1}^{\texttt{nsec}} \sum_{t=1}^{T^{(s)}} (y_t - \mu_i) * (y_t - \mu_i)^T * \gamma_t^{(s)}(i)}{\sum_{s=1}^{\texttt{nsec}} \sum_{t=1}^{T^{(s)}} \gamma_t^{(s)}(i)}$$

where s is the index of section and `nsec` is the number of sections. Notice that μ_i is used to demean y_t when Σ_i is updated. The EM algorithm is supported only for the Gaussian HMM and the maximum likelihood method.

Starting in version 2023.05, you can execute the EM algorithm in parallel on multiple threads and nodes if you divide the data into sections and specify the SECTION= option in the ID statement. In this case, each thread is assigned some sections of the data on the available nodes. Or if backward compatibility is a concern, you can run the EM algorithm in nonparallel mode by specifying the SINGLE option in the OPTIMIZE statement. Parallel mode can reduce run time and improve the algorithm's performance. Also, there might be a numerical difference between your results in these two modes.

For more information about the active set and interior point algorithms and their related options, see the chapter "The Nonlinear Programming Solver" in *SAS/OR User's Guide: Mathematical Programming*. For more information about the SGD algorithm, see the section "OPTIMIZE Statement" on page 1221.

Distributed Multistart

In the distributed multistart algorithm, the local optimizations that start from various initial points run concurrently. Each of these local optimizations requires evaluation of the objective or constraints as well as their first or second derivatives at different points, which are referred to as function evaluations. If function evaluations are performed in distributed mode, as in the case of PROC HMM, the evaluations necessary for each local optimization are performed in a subsession that stems from the current main CAS session. You can specify the SESSNODES=, FESESSIONS=, and FESESSNODES= options to achieve a balanced and more efficient usage of the computing resources, in particular the available nodes and the threads on each node.

Suppose that there are a total of W worker nodes in the current main CAS session and you specify SESSNODES=m, FESESSIONS=s, and FESESSNODES=n. Then m out of W worker nodes are dedicated to the local optimizations of the multistart algorithm. Hence the total number of local optimizations that run concurrently is

$$N_{\text{locopt}} = \sum_{i=1}^{m} t_i$$

where t_i is the number of threads on node i. In addition, s subsessions, each of which has n worker nodes, are created and dedicated to function evaluations. When a function evaluation is requested in one of N_{locopt} local

optimizations, the request is sent to one of the s subsessions. Use the following guidelines when you select the values of the SESSNODES=, FESESSIONS=, and FESESSNODES= options:

- The value of m or n should never be greater than W.
- The total number of local optimizations N_{locopt} should be as close to s as possible. This allows a good balance between the resources for local optimizations and function evaluations.
- If you have a limited number of worker nodes W, then in order to achieve the correct balance, preference is often given to a combination that favors a larger s and smaller n over one that favors a smaller s and larger n. This helps reduce the number of function evaluation requests that are waiting to be processed in a function evaluation subsession.
- Avoid a scenario in which $m + sn$ is far greater than W. In such a scenario, competition for resources would occur and would degrade performance.

Matrix Expression

The INITIAL and PRIOR statements operate on matrices. That is, you can specify the parameter matrices or constant matrices through the INITIAL and PRIOR statements' built-in operators and functions. You can add elements of the matrices **A** and **B** by using the expression **A+B**, perform matrix multiplication by using the expression **A*B**, and perform elementwise multiplication by using the expression **A#B**. You can get the diagonal elements of the matrix **A** by using the function **DIAG(A)**, and you can get the $n \times n$ identity matrix by using the function **I(n)**.

Each equation is written as a matrix expression that is composed of constants, operators, and functions.

Constants

Constants are either scalar constants (such as –1.2, 0.3, and so on) or matrix constants enclosed in braces (such as the 2×2 matrix **{1 2, 3 4}** or the 1×3 row vector **{-0.2 5.3 12}**).

The matrix constant cannot be the first item in the INITIAL statement unless you enclose it in parentheses. For example, you cannot specify the following statement:

```
initial {-0.1 -0.2, -0.3 -0.4} = MU;
```

You can specify the preceding example by enclosing the first matrix constant in parentheses, as follows:

```
initial ({-0.1 -0.2, -0.3 -0.4}) = MU;
```

Operators

Operators define the operations on operands. Table 20.2 lists all built-in operators that the INITIAL statement supports.

Table 20.2 Operators

Operator	Name	Description
+	Addition	Adds corresponding matrix elements
=	Comparison, equal	Compares matrix elements
<	Comparison, less than	Compares matrix elements
<=	Comparison, not greater than	Compares matrix elements
>	Comparison, greater than	Compares matrix elements
>=	Comparison, not less than	Compares matrix elements
||	Concatenation, horizontal	Concatenates matrices horizontally
//	Concatenation, vertical	Concatenates matrices vertically
@	Direct product	Takes the direct product of two matrices
∼	Distribution	Links the distribution function to the matrix elements
:	Index creation	Creates an index vector
#	Multiplication, elementwise	Performs elementwise multiplication
*	Multiplication, matrix	Performs matrix multiplication
−	Sign reverse	Reverses the signs of elements
[ ]	Subscripts	Selects submatrices
−	Subtraction	Subtracts corresponding matrix elements
`	Transpose	Transposes a matrix

For more information about each operator, see the section "Details of Operators" on page 1278.

Table 20.3 shows the precedence of matrix operators.

Table 20.3 Precedence of Operators

Priority Group	Operators			
I (highest)	[ ] (subscripts)	` (transpose)		
II	− (sign reverse)			
III	*	#	@	
IV	− (subtraction)	+		
V	||	//	:	
VI (lowest)	=	<	<= > >= ∼	

Each equation can be a compound expression that involves several matrix operators and operands. The rules for evaluating compound expressions are as follows:

- Evaluation follows the order of operator precedence as described in Table 20.3. Group I has the highest priority; that is, Group I operators are evaluated first. Group II operators are evaluated after Group I operators, and so on. For example, $1 + 2 * 3$ returns 7.
- If neighboring operators in an expression have equal precedence, the expression is evaluated from left to right, except for the Group I operators. For example, $1 - 2 - 3$ returns -4.
- All expressions in parentheses are evaluated first, following the two preceding rules. For example, $3 * (2 + 1)$ returns 9.

Functions

Functions are divided into two types. One type of function refers to parameters to be estimated; examples are **MU(K,I)** and **SIGMA(K,I,J)**. The other type does not; examples are **I(n)** and **DIAG(A)**.

Functions that refer to parameters are listed in Table 20.4. The arguments of functions can be matrices. The simplest case, scalar arguments, is discussed first. For convenience, the scalar indices **i** and **j** refer to the position of the element in the coefficient matrix, the scalar **l** refers to the lag value, the scalar **c** refers to the component index, and the scalar **k** refers to the state value.

Table 20.4 Functions That Refer to Parameters

Function	Description
AR(k,l,i,j)	Autoregressive parameter of the lag l of the jth dependent variable, $y_{j,t-l}$, to the ith dependent variable at time t, y_{it}, for state k in regime-switching autoregression
CONST(k,i)	Intercept parameter of the ith time series, y_{it}, for state k in regime-switching regression or regime-switching autoregression, or the alias of MU for Gaussian HMM or GM HMM
COV(k,i,j)	Covariance of innovations parameter between the ith and jth error processes for state k in regime-switching regression or regime-switching autoregression, or the alias of SIGMA for Gaussian HMM or GM HMM
CPM(i,j)	The (i, j)th element in the category probability matrix in the finite HMM
LAMBDA(K)	Mean parameter of the time series, y_{it}, for state k in the Poisson HMM
ISPV(i)	The ith element in the initial state probability vector
LTREND(k,i)	Linear trend parameter of the ith time series, y_{it}, for state k in regime-switching regression or regime-switching autoregression when the TREND= option is specified in the MODEL statement
MCP(i,j)	The (i, j)th element in the mixture component probabilities in the GM HMM
MU(k,i)	Mean parameter of the ith time series, y_{it}, for state k in the Gaussian HMM, or alias for CONST for regime-switching regression or regime-switching autoregression
MU(k,c,i)	Mean parameter of the ith time series, y_{it}, for the cth component at state k in the GM HMM
QTREND(k,i)	Quadratic trend parameter of the ith time series, y_{it}, for state k in regime-switching regression or regime-switching autoregression when TREND=QUAD is specified in the MODEL statement
SD(k,i,j)	Alias of SDUMMY
SDUMMY(k,i,j)	The jth seasonal dummy of the ith time series at time t, y_{it}, for state k in regime-switching regression or regime-switching autoregression, where $j = 1, \ldots, (nseason-1)$, where *nseason* is the value of the NSEASON= option in the MODEL statement
SIGMA(k,i,j)	Covariance parameter between the ith and jth time series, y_{it} and y_{jt}, for state k in the Gaussian HMM, or the alias of COV for regime-switching regression or regime-switching autoregression
SIGMA(k,c,i,j)	Covariance parameter between the ith and jth time series, y_{it} and y_{jt}, for cth component at state k in the GM HMM
TPM(i,j)	The (i, j)th element in the transition probability matrix

Table 20.4 *continued*

Function	Description
`XL(k,l,i,j)`	Exogenous parameter of the lag l of the jth exogenous variable, $x_{j,t-l}$, to the ith dependent variable at time t, y_{it}, for state k in regime-switching regression or regime-switching autoregression

The functions that refer to parameters, as shown in Table 20.4, are valid only when the parameters are included in the model. For example, the **ISPV** function is valid only when you specify the ESTISPV option in the MODEL statement; otherwise, the HMM procedure issues errors and then stops.

The functions that refer to parameters, as shown in Table 20.4, accept vector and matrix arguments and return the matrix that is constructed by the corresponding parameters. According to the number of arguments, the following list shows what matrix a function returns when the arguments are vectors:

- A function, $\mathbf{FUNC}_1$, that has one vector argument I, where $I = (i_1\ i_2\ \ldots\ i_{n_I})'$, returns a vector $R = (r_1\ r_2\ \ldots\ r_{n_I})'$, where $r_k = \mathbf{FUNC}_1(i_k)$, $k = 1, \ldots, n_I$. **ISPV** is a type of $\mathbf{FUNC}_1$.
- A function, $\mathbf{FUNC}_2$, that has two vector arguments I and J, where $I = (i_1\ i_2\ \ldots\ i_{n_I})'$ and $J = (j_1\ j_2\ \ldots\ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_J} \\ \cdots & & & \\ r_{n_I,1} & r_{n_I,2} & \cdots & r_{n_I,n_J} \end{pmatrix}$$

  where $r_{k,m} = \mathbf{FUNC}_2(i_k, j_m)$, $k = 1, \ldots, n_I, m = 1, \ldots, n_J$. **CONST**, **LTREND**, **MU**, **QTREND**, and **TPM** are types of $\mathbf{FUNC}_2$.

- A function, $\mathbf{FUNC}_N$, that has N vector arguments $I_1, I_2, \ldots,$ and I_N, where $N > 2$ and N is odd and where $I_n = (i_{n,1}\ i_{n,2}\ \ldots\ i_{n,j_n})', n = 1, \ldots, N$, returns a block matrix,

$$R = \begin{pmatrix} B_1 \\ B_2 \\ \cdots \\ B_{j_n} \end{pmatrix}$$

  where $B_k = \mathbf{FUNC}_N(i_{1,k}, I_2, \ldots, I_N)$, $k = 1, \ldots, j_1$, and B_k is constructed in the same way as $\mathbf{FUNC}_{N-1}(I_2, \ldots, I_N)$.

  For example, a function, $\mathbf{FUNC}_3$, that has three vector arguments K, I, and J, where $K = (k_1\ k_2\ \ldots\ k_{n_K})'$, $I = (i_1\ i_2\ \ldots\ i_{n_I})'$, and $J = (j_1\ j_2\ \ldots\ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_J} \\ \cdots & & & \\ r_{n_K n_I,1} & r_{n_K n_I,2} & \cdots & r_{n_K n_I,n_J} \end{pmatrix}$$

where $r_{u,v} = \texttt{FUNC}_3\,(k_{d_1(u,n_I)}, i_{d_2(u,n_I)}, j_v)$, $u = 1, \ldots, n_K n_I$, $v = 1, \ldots, n_J$, and $d_1(m,n)$ is the integer such that $d_1(m,n)n \geq m > (d_1(m,n)-1)n$ and $d_2(m,n) = m - (d_1(m,n)-1)n$. **COV** and **MU** for GM HMM and **SIGMA** for Gaussian HMM are types of $\texttt{FUNC}_3$.

- A function, $\texttt{FUNC}_N$, that has N vector arguments $I_1, I_2, \ldots$, and I_N, where $N > 2$ and N is even and where $I_n = (i_{n,1}\ i_{n,2}\ \ldots\ i_{n,j_n})'$, $n = 1, \ldots, N$, returns a block matrix,

$$R = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,j_2} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,j_2} \\ \cdots & & & \\ B_{j_1,1} & B_{j_1,2} & \cdots & B_{j_1,j_2} \end{pmatrix}$$

where $B_{k,l} = \texttt{FUNC}_N\,(i_{1,k}, i_{2,l}, I_3, \ldots, I_N)$, $k = 1, \ldots, j_1$, $l = 1, \ldots, j_2$, and $B_{k,l}$ is constructed in the same way as $\texttt{FUNC}_{N-2}\,(I_3, \ldots, I_N)$.

For example, a function, $\texttt{FUNC}_4$, that has four vector arguments K, L, I, and J, where $K = (k_1\ k_2\ \ldots\ k_{n_K})'$, $L = (l_1\ l_2\ \ldots\ l_{n_L})'$, $I = (i_1\ i_2\ \ldots\ i_{n_I})'$, and $J = (j_1\ j_2\ \ldots\ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_L n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_L n_J} \\ \cdots & & & \\ r_{n_K n_I,1} & r_{n_K n_I,2} & \cdots & r_{n_K n_I,n_L n_J} \end{pmatrix}$$

where $r_{u,v} = \texttt{FUNC}_4\,(k_{d_1(u,n_I)}, l_{d_1(v,n_J)}, i_{d_2(u,n_I)}, j_{d_2(v,n_J)})$, $u = 1, \ldots, n_K n_I$, $v = 1, \ldots, n_L n_J$, where $d_1(m,n)$ is the integer such that $d_1(m,n)n \geq m > (d_1(m,n)-1)n$ and $d_2(m,n) = m - (d_1(m,n)-1)n$. **AR** and **SIGMA** for GM HMM and **XL** are types of $\texttt{FUNC}_4$.

The functions that refer to parameters can accept empty arguments or omit any number of last arguments. The empty or omitted arguments are replaced by all possible values for those arguments. For example, PROC HMM is used to fit a bivariate three-state Gaussian HMM as follows:

```
model y1 y2 / type=gaussian nstate=3;
```

In order to initialize the second row of **TPM** to be {0.1 0.7 0.2}, you can use the following statement:

```
initial tpm(2,{1 2 3}) =  {0.1 0.7 0.2};
```

Taking advantage of empty arguments, you can specify the preceding example as follows:

```
initial tpm(2,) =  {0.1 0.7 0.2};
```

To get all coefficients of the covariance matrix for state 2, you can use **SIGMA(2, {1 2}, {1 2})**, or **SIGMA(2, ,)**, or **SIGMA(2)**. To get all coefficients of exogenous variables on dependent variables, you can use **SIGMA({1 2 3}, {1 2}, {1 2})**, or **SIGMA(, ,)**, or **SIGMA()**, or even just **SIGMA**.

Another type of function does not refer to parameters but generates useful matrices. Table 20.5 lists all built-in functions.

Table 20.5 Functions That Do Not Refer to Parameters

Function	Description
`DIAG(A)`	Creates a diagonal matrix from a vector or extracts the diagonal elements of a matrix
`I(n)`	Creates an $n \times n$ identity matrix
`J(m,n,elem)`	Creates an $m \times n$ matrix whose elements are all equal to `elem`
`SHAPE(A,m,n)`	Creates an $m \times n$ matrix whose elements are from matrix `A`

For more information about each function in Table 20.5, see the section "Details of Functions" on page 1282.

Details of Operators

This section describes all operators that are available in the HMM procedure. Each subsection describes the operator and shows how it is used.

Addition Operator: +

The addition operator (+) computes a new matrix whose elements are the sums of corresponding elements of the two operands. Following are examples of how the addition operator is used:

`matrix1 + matrix2` adds the element in the *i*th row and *j*th column of the first matrix to the element in the *i*th row and *j*th column of the second matrix, for $i = 1, \ldots, n$, $j = 1, \ldots, p$, where `matrix1` and `matrix2` are both $n \times p$ matrices.

For example, `{1 2 3, 4 5 6} + {7 8 9, 10 11 12}` results in `{8 10 12, 14 16 18}`.

`matrix + scalar` adds the `scalar` value to each element of the `matrix`. For example, you can obtain `{2 3 4, 5 6 7}` from `{1 2 3, 4 5 6} + 1`.

`matrix + vector` adds the `vector` value to each row or column of the $n \times p$ `matrix`:

- If you add an $n \times 1$ column vector, each row of the vector is added to each row of the matrix. For example, you can obtain `{2 3 4, 5 6 7}` from `{1 2 3, 4 5 6} + {1, 1}`.
- If you add a $1 \times p$ row vector, each column of the vector is added to each column of the matrix. For example, you can obtain `{2 3 4, 5 6 7}` from `{1 2 3, 4 5 6} + {1 1 1}`.

Comparison Operators: =, <, <=, >, >=

The comparison operators (=, <, <=, >, >=) compare two matrices element by element and return a list of equivalent restrictions on only scalar constants and parameters.

`matrix1 = matrix2`

`matrix1 < matrix2`

`matrix1 <= matrix2`

`matrix1 > matrix2`

matrix1 >= matrix2

For example, the INITIAL statement with matrix expressions

```
initial SIGMA(1,{1,2},{1,2}) = SIGMA(2,{3,4},{3,4});
```

is transformed into the following equivalent INITIAL statement with scalar parameters:

```
initial SIGMA(1,1,1) = SIGMA(2,3,3),
        SIGMA(1,1,2) = SIGMA(2,3,4),
        SIGMA(1,2,1) = SIGMA(2,4,3),
        SIGMA(1,2,2) = SIGMA(2,4,4);
```

You can also use the comparison operators to conveniently compare all elements of a matrix to a scalar. If either argument is a scalar, then the HMM procedure performs an elementwise comparison between each element of the matrix and the scalar.

You can also compare an $n \times p$ matrix to a row or column vector:

- If the comparison is to an $n \times 1$ column vector, the HMM procedure compares each row of the vector to each row of the matrix.
- If the comparison is to a $1 \times p$ row vector, the HMM procedure compares each column of the vector to each column of the matrix.

For example, the following statements are equivalent:

```
initial SIGMA(1,4:5,1:3) = 0.2;

initial SIGMA(1,4:5,1:3) = {0.2, 0.2};

initial SIGMA(1,4:5,1:3) = {0.2 0.2 0.2};
```

Concatenation Operator, Horizontal: ||

The horizontal concatenation operator (||) produces a new matrix by horizontally joining **matrix1** and **matrix2**.

matrix1 || **matrix2**

The matrices must have the same number of rows, which is also the number of rows in the new matrix. The number of columns in the new matrix is the number of columns in **matrix1** plus the number of columns in **matrix2**.

For example, `{1 1 1, 7 7 7} || {0 0 0, 8 8 8}` returns `{1 1 1 0 0 0, 7 7 7 8 8 8}`.

Concatenation Operator, Vertical: //

The vertical concatenation operator (//) produces a new matrix by vertically joining **matrix1** and **matrix2**.

matrix1 // matrix2

The matrices must have the same number of columns, which is also the number of columns in the new matrix. The number of rows in the new matrix is the number of rows in **matrix1** plus the number of rows in **matrix2**.

For example, `{1 1 1} // {0 0 0, 8 8 8}` returns `{1 1 1, 0 0 0, 8 8 8}`.

Direct Product Operator: @

The direct product operator (@) computes a new matrix that is the direct product (also called the Kronecker product) of **matrix1** and **matrix2**.

```
matrix1 @ matrix2
```

For matrices $\mathbf{A}$ and $\mathbf{B}$, the direct product is denoted by $\mathbf{A} \otimes \mathbf{B}$. The number of rows in the new matrix equals the product of the number of rows in **matrix1** and the number of rows in **matrix2**; the number of columns in the new matrix equals the product of the number of columns in **matrix1** and the number of columns in **matrix2**.

Specifically, if $\mathbf{A}$ is an $n \times p$ matrix and $\mathbf{B}$ is an $m \times q$ matrix, then the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is the following $nm \times pq$ block matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{11}B & \cdots & A_{1p}B \\ \vdots & \ddots & \vdots \\ A_{n1}B & \cdots & A_{np}B \end{bmatrix}$$

For example, `{1 2, 3 4} @ {0 2}` returns `{0 2 0 4, 0 6 0 8}`, and `{0 2} @ {1 2, 3 4}` returns `{0 0 2 4, 0 0 6 8}`. Note that the direct product of two matrices is not commutative.

Distribution Operator: ~

The distribution operator links the matrix to its distribution function.

```
matrix ~ distribution-function
```

This operator is used in the PRIOR statement. For more information, see the section "PRIOR Statement" on page 1226.

For example, `ISPV ~ DIR({1,1,1})` means that the initial state probability vector (ISPV) follows a Dirichlet distribution with an argument of a 3×1 vector `{1,1,1}`.

Index Creation Operator: :

The index creation operator (:) creates a column vector whose first element is **value1**, second element is **value1**+1, and so on, until the last element, which is less than or equal to **value2**.

```
value1 : value2
```

For example, `3 : 6` returns `{3 4 5 6}`.

If **value1** is greater than **value2**, a reverse-order index is created. For example, `6 : 3` returns `{6 5 4 3}`.

Neither **value1** nor **value2** is required to be an integer.

Multiplication Operator, Elementwise:

The elementwise multiplication operator (#) computes a new matrix whose elements are the products of the operands. Following are examples of how the elementwise multiplication operator is used:

`matrix1 # matrix2`	computes a new matrix whose elements are the products of the corresponding elements of **matrix1** and **matrix2**. For example, `{1 2, 3 4} # {4 8, 0 5}` returns `{4 16, 0 20}`.
`matrix # scalar`	multiplies each element in **matrix** by the **scalar** value.

`matrix # vector` multiplies each row or column of the $n \times p$ matrix by a corresponding element of the vector:

- If you multiply by an $n \times 1$ column vector, each row of the matrix is multiplied by the corresponding row of the vector.
- If you multiply by a $1 \times p$ row vector, each column of the matrix is multiplied by the corresponding column of the vector.

For example, a 2×3 matrix can be multiplied on either side by a 2×3, 1×3, 2×1, or 1×1 scalar.

The product of elementwise multiplication is also known as the Schur or Hadamard product. Elementwise multiplication (which uses the # operator) should not be confused with matrix multiplication (which uses the * operator).

*Multiplication Operator, Matrix: **

The matrix multiplication operator (*) computes a new matrix by performing matrix multiplication.

```
matrix1 * matrix2
```

The first matrix must have the same number of columns as the second matrix has rows. The new matrix has the same number of rows as the first matrix and the same number of columns as the second matrix. That is, if $\mathbf{A}$ is an $n \times p$ matrix and $\mathbf{B}$ is a $p \times m$ matrix, then the product $\mathbf{A} * \mathbf{B}$ is an $n \times m$ matrix. The (i, j) element of the product is the sum $\sum_{k=1}^{p} A_{ik} B_{kj}$.

For example, `{1 2, 3 4} * {1, 2}` returns `{5, 11}`.

Sign Reversal Operator: –

The sign reversal operator (−) computes a new matrix whose elements are formed by reversing the sign of each element in **`matrix`**. The sign reversal operator is also called the unary minus operator.

```
– matrix
```

For example, `–{–1 7 6, 2 0 –8}` returns `{1 –7 –6, –2 0 8}`.

Subscripts: [ ]

Subscripts are used with matrices to select submatrices, where **`rows`**, **`columns`**, and **`elements`** are expressions that evaluate to scalars or vectors. If these expressions are numeric, they must contain valid subscript values of rows and columns, or the indices, in the argument matrix.

```
matrix[rows, columns]
```

```
matrix[elements]
```

For example, `{1 2 3, 4 5 6, 7 8 9}[2,3]` returns `6`; `{1 2 3, 4 5 6, 7 8 9}[2,1:3]` returns `{4 5 6}`; and `{1 2 3, 4 5 6, 7 8 9}[,3]` returns `{3, 6, 9}`. Because the HMM procedure stores matrices in row-major order, `{11 22 33, 44 55 66, 77 88 99}[{3 5 9}]` returns `{33, 55, 99}`.

Subtraction Operator: –

The subtraction operator (−) computes a new matrix whose elements are formed by subtracting the corresponding elements of the second operand from the first operand.

matrix1 – matrix2 computes a new matrix whose elements are formed by subtracting the corresponding elements of **matrix2** from those of **matrix1**.

For example, `{1 2 3, 4 5 6} – {1 1 1, 1 1 1}` returns `{0 1 2, 3 4 5}`.

matrix – scalar subtracts the **scalar** value from each element of the **matrix**.

For example, `{1 2 3, 4 5 6} – 1` returns `{0 1 2, 3 4 5}`.

matrix – vector subtracts the **vector** from each element of the **matrix**:

- If you subtract an $n \times 1$ column vector, each row of the vector is subtracted from each row of the matrix. For example, `{1 2 3, 4 5 6} – {1, 1}` returns `{0 1 2, 3 4 5}`.
- If you subtract a $1 \times p$ row vector, each column of the vector is subtracted from each column of the matrix. For example, `{1 2 3, 4 5 6} – {1 1 1}` returns `{0 1 2, 3 4 5}`.

Transpose Operator: `

The transpose operator, denoted by the backquote (`), exchanges the rows and columns of **matrix**, producing the transpose of **matrix**.

```
matrix`
```

If v is the value in the ith row and jth column of **matrix**, then the transpose of **matrix** contains v in the jth row and ith column. If **matrix** contains n rows and p columns, the transpose has p rows and n columns.

For example, `{1 2, 3 4, 5 6}`` returns `{1 3 5, 2 4 6}`.

Details of Functions

DIAG Function

The **DIAG** function creates a diagonal matrix from a vector or extracts the diagonal elements of a matrix.

```
DIAG(matrix)
```

The **matrix** argument can be either a square matrix or a vector:

- If **matrix** is a square matrix, the **DIAG** function creates a vector from the diagonal elements of the matrix.

  For example, `DIAG({1 2 3, 4 5 6, 7 8 9})` returns `{1, 5, 9}`.

- If **matrix** is a vector, the **DIAG** function creates a matrix whose diagonal elements are the values in the vector. All off-diagonal elements are zeros.

  For example, `DIAG({1 5 9})` or `DIAG({1, 5, 9})` returns `{1 0 0, 0 5 0, 0 0 9}`.

You can call the **DIAG** function repeatedly. For example, `DIAG(DIAG({1 2 3, 4 5 6, 7 8 9}))` returns `{1 0 0, 0 5 0, 0 0 9}`.

I Function

The **I** function creates an identity matrix that contains **dim** rows and columns.

```
I(dim)
```

The diagonal elements of an identity matrix are ones; all other elements are zeros. The value of **dim** must be an integer greater than or equal to 1. Noninteger operands are rounded to the nearest integer.

For example, **I(3)** returns **{1 0 0, 0 1 0, 0 0 1}**.

J Function

The **J** function creates a matrix that contains **nrow** rows and **ncol** columns, in which all elements are equal to **value**.

```
J(nrow, ncol, value)
```

The arguments **nrow** and **ncol** are both integers; **value** can be any expression that returns a linear combination of scalar constants and parameters.

For example, **J(2, 3, 1)** returns **{1 1 1, 1 1 1}**.

SHAPE Function

The **SHAPE** function creates a new matrix from data in **matrix**.

```
SHAPE(matrix, nrow, ncol)
```

The values **nrow** and **ncol** specify the number of rows and columns, respectively, in the new matrix. The **SHAPE** function produces the new matrix by traversing the argument matrix in row-major order until it reaches the specified number of elements. If necessary, the SHAPE function reuses elements.

For example, **SHAPE({1 2 3, 4 5 6}, 3, 2)** returns **{1 2, 3 4, 5 6}**; **SHAPE({1 2 3, 4 5 6}, 5, 2)** returns **{1 2, 3 4, 5 6, 1 2, 3 4}**; and **SHAPE({1 2 3, 4 5 6}, 1, 4)** returns **{1 2 3 4}**.

Label Switching Problem

Label switching is an important and severe problem for all kinds of HMMs. Nonidentifiability is caused by invariance to relabeling the states of the hidden Markov chain (or the components of the mixture distribution in the case of GH HMMs); for further discussion and illustration, see Frühwirth-Schnatter (2006, page 15 and 173) and the references therein.

Any two HMMs are mathematically equivalent if the only difference between them is that the states or components are labeled in different ways. For example, consider the following models, which are both univariate bistate Gaussian HMMs described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, and μ_1 and Σ_1 are mean and covariance parameters for state 1, and μ_2 and Σ_2 are mean and covariance parameters for state 2:

$$\text{model 1: } \pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{pmatrix}, \mu_1 = 1, \Sigma_1 = 1, \mu_2 = -1, \Sigma_2 = 4$$

$$\text{model 2: } \pi = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{pmatrix}, \mu_1 = -1, \Sigma_1 = 4, \mu_2 = 1, \Sigma_2 = 1$$

In theory, these two models are equivalent, and just the state labels "1" and "2" are switched. When the labels are switched, it is inconvenient to compare any state-related results, such as filtered or smoothed probabilities or decoded path. Because of the randomness in the optimization, you might even get "different" estimation results every time because of label switching.

You can use the LABELSWITCH= option in the PROC HMM statement to solve the label switching problem. When the INMODEL= option in the SCORE statement is not specified, the LABELSWITCH=(SORT=ASC(SIGMA)) option is applied by default; that is, the states are labeled in the ascending order of the covariance parameter estimates. When the INMODEL= option in the SCORE statement is specified, the LABELSWITCH=(SORT=NONE) option is applied by default; that is, there is no relabeling. You can specify additional suboptions in the LABELSWITCH= option to label the states (and components) in whichever way you prefer.

An Example to Illustrate the LABELSWITCH= Option

The following data are simulated for this example. The data generating process (DGP) follows a bistate tricomponent GM HMM.

```
%let a11 = 0.98;
%let a22 = 0.98;
%let c1_1 = 0.3;
%let c1_2 = 0.4;
%let c1_3 = 0.3;
%let c2_1 = 0.3;
%let c2_2 = 0.4;
%let c2_3 = 0.3;
%let mu1_1_1 = -1;
%let mu1_2_1 = -6;
%let mu1_3_1 = -11;
%let sigma1_1_11 = 1;
%let sigma1_2_11 = 2;
%let sigma1_3_11 = 3;
%let mu2_1_1 = 6;
%let mu2_2_1 = 12;
%let mu2_3_1 = 20;
%let sigma2_1_11 = 4;
%let sigma2_2_11 = 5;
%let sigma2_3_11 = 6;
%let seed = 1234;
%let T = 400000;
%let nSections = 200;

data labelSwitchDGP;
   do t = 1 to &T.;
      sec = ceil(t*&nSections./&T.);
      if(t=1) then do;
         * initial probability distribution;
         p = (1-&a22.)/((1-&a11.)+(1-&a22.));;
      end;
      else do;
         * transition probability matrix;
         if(lags=1) then p = &a11.;
         else p = 1-&a22.;
```

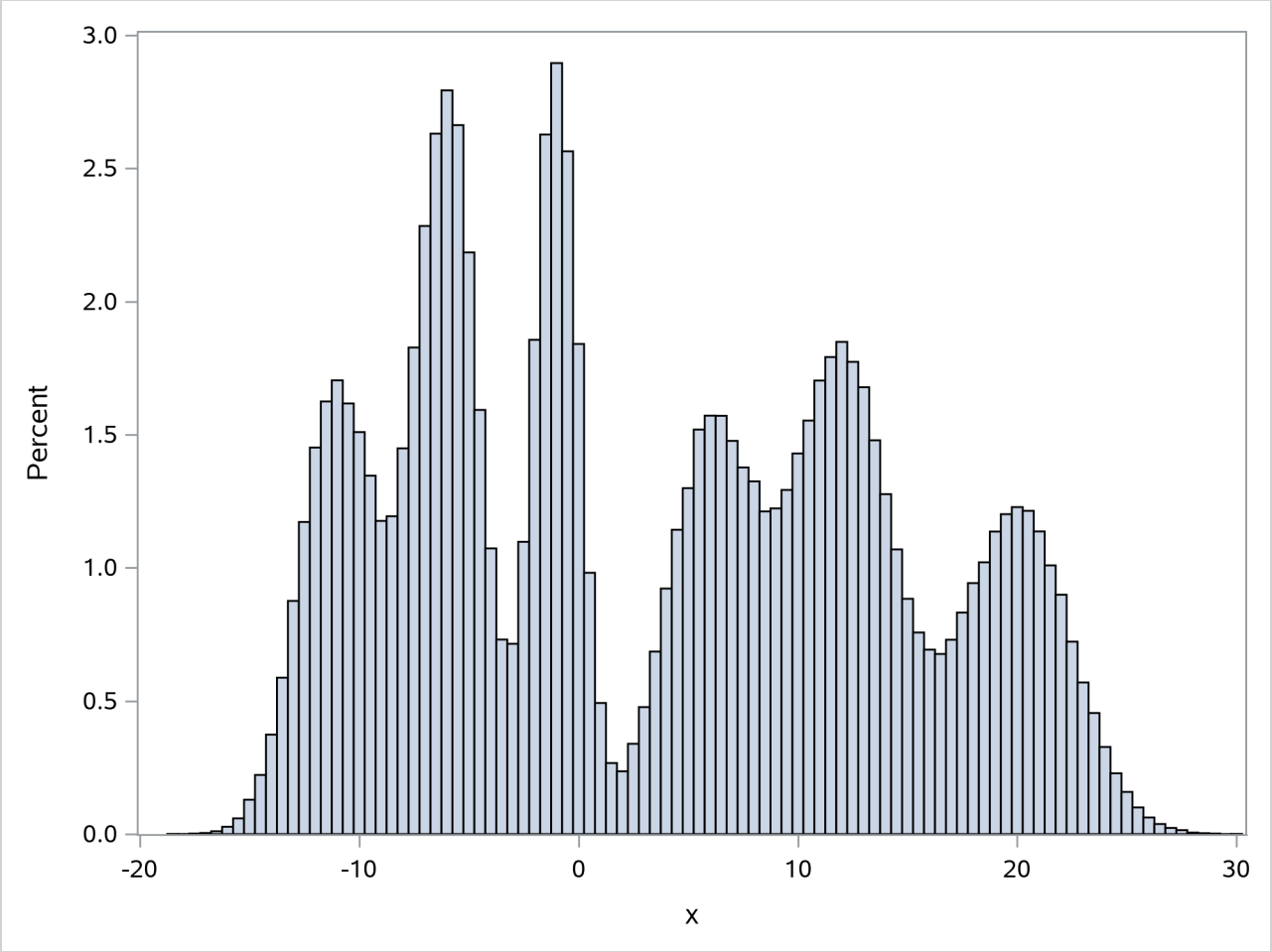
```
            end;
            u = uniform(&seed.);
            if(u<=p) then s=1;
            else s = 2;
            e1 = normal(&seed.);
            if(s=1) then do;
               * choose component;
               u = uniform(&seed.);
               if(u<=&c1_1.) then do;
                  * x ~ N(mu, Sigma) at state 1, component 1;
                  x = &mu1_1_1. + sqrt(&sigma1_1_11.)*e1;
               end;
               else do;
                  if(u<=&c1_1.+&c1_2.) then do;
                     * x ~ N(mu, Sigma) at state 1, component 2;
                     x = &mu1_2_1. + sqrt(&sigma1_2_11.)*e1;
                  end;
                  else do;
                     * x ~ N(mu, Sigma) at state 1, component 3;
                     x = &mu1_3_1. + sqrt(&sigma1_3_11.)*e1;
                  end;
               end;
            end;
            else do;
               * choose component;
               u = uniform(&seed.);
               if(u<=&c2_1.) then do;
                  * x ~ N(mu, Sigma) at state 2, component 1;
                  x = &mu2_1_1. + sqrt(&sigma2_1_11.)*e1;
               end;
               else do;
                  if(u<=&c1_1.+&c1_2.) then do;
                     * x ~ N(mu, Sigma) at state 2, component 2;
                     x = &mu2_2_1. + sqrt(&sigma2_2_11.)*e1;
                  end;
                  else do;
                     * x ~ N(mu, Sigma) at state 2, component 3;
                     x = &mu2_3_1. + sqrt(&sigma2_3_11.)*e1;
                  end;
               end;
            end;
            output;
            lags = s;
         end;
      run;
```

The following statements plot the histogram of the simulated x by ignoring the sectional and sequential information.

```
proc sgplot data=labelSwitchDGP;
   histogram x;
run;
```

Figure 20.90 shows the histogram of x, in which there are six modes around 20, 12, 6, –1, –6, and –11, as expected.

Figure 20.90 Plot of Histogram of x



The following statements make a copy of the data in the library:

```
data mylib.labelSwitchDGP; set labelSwitchDGP; run;
```

The following statements estimate the six-state Gaussian HMM by using the initial values {20, 12, 6, –1, –6, –11} for the mean parameter:

```
proc hmm data=mylib.labelSwitchDGP;
   id time=t section=sec;
   model x / type=gaussian nstate=6;
   optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
   initial mu={20, 12, 6, -1, -6, -11};
run;
```

The final parameter estimates after optimization, which are shown in Figure 20.91, are totally different from the reported parameter estimates in Figure 20.92. In Figure 20.91, the six states, described as Gaussian distribution, are in order N(20, 6), N(12, 5), N(6, 4), N(–1, 1), N(–6, 2), and N(–11, 3); however, in Figure 20.92, the six states are in order N(–1, 1), N(–6, 2), N(–11, 3), N(6, 4), N(12, 5), and N(20, 6). The difference is caused by the states being relabeled, by default, according to the estimates of the covariance matrix parameter of each state in ascending order; that is, the LABELSWITCH=(SORT=ASC(SIGMA)) option is applied by default.

Figure 20.91 Final Parameter Estimates after Optimization

Optimization Results	
Parameter	**Estimate**
TPM1_1	0.292461
TPM1_2	0.392902
TPM1_3	0.294507
TPM1_4	0.005954
TPM1_5	0.007939
TPM1_6	0.006237
TPM2_1	0.294241
TPM2_2	0.387707
TPM2_3	0.298507
TPM2_4	0.005716
TPM2_5	0.007708
TPM2_6	0.006121
TPM3_1	0.295218
TPM3_2	0.387390
TPM3_3	0.297737
TPM3_4	0.006181
TPM3_5	0.007747
TPM3_6	0.005726
TPM4_1	0.006300
TPM4_2	0.008122
TPM4_3	0.006358
TPM4_4	0.292704
TPM4_5	0.390938
TPM4_6	0.295579
TPM5_1	0.005966
TPM5_2	0.007509
TPM5_3	0.006372
TPM5_4	0.294351
TPM5_5	0.390263
TPM5_6	0.295538
TPM6_1	0.005998
TPM6_2	0.008252
TPM6_3	0.005837
TPM6_4	0.292779
TPM6_5	0.392673
TPM6_6	0.294461
MU1_1	19.994009
MU2_1	12.015924
MU3_1	6.036520
MU4_1	-1.002846
MU5_1	-5.994957
MU6_1	-10.977249
SIGMA1_1_1	6.082481
SIGMA2_1_1	4.881128
SIGMA3_1_1	4.037455
SIGMA4_1_1	1.007495
SIGMA5_1_1	1.987658
SIGMA6_1_1	3.042159

Figure 20.92 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.292704	0.001985	147.45	<.0001
TPM1_2	0.390938	0.002676	146.08	<.0001
TPM1_3	0.295579	0.002419	122.18	<.0001
TPM1_4	0.006358	0.000369	17.23	<.0001
TPM1_5	0.008122	0.000429	18.93	<.0001
TPM1_6	0.006300	0.000352	17.91	<.0001
TPM2_1	0.294351	0.001805	163.09	<.0001
TPM2_2	0.390263	0.002506	155.74	<.0001
TPM2_3	0.295538	0.002251	131.31	<.0001
TPM2_4	0.006372	0.000331	19.24	<.0001
TPM2_5	0.007509	0.000368	20.40	<.0001
TPM2_6	0.005966	0.000304	19.62	<.0001
TPM3_1	0.292779	0.002057	142.35	<.0001
TPM3_2	0.392673	0.002761	142.23	<.0001
TPM3_3	0.294461	0.002483	118.61	<.0001
TPM3_4	0.005837	0.000367	15.91	<.0001
TPM3_5	0.008252	0.000435	18.99	<.0001
TPM3_6	0.005998	0.000350	17.14	<.0001
TPM4_1	0.006181	0.000354	17.46	<.0001
TPM4_2	0.007747	0.000411	18.83	<.0001
TPM4_3	0.005726	0.000356	16.10	<.0001
TPM4_4	0.297737	0.003360	88.62	<.0001
TPM4_5	0.387390	0.004081	94.94	<.0001
TPM4_6	0.295218	0.002374	124.34	<.0001
TPM5_1	0.005716	0.000305	18.73	<.0001
TPM5_2	0.007708	0.000360	21.42	<.0001
TPM5_3	0.006121	0.000320	19.13	<.0001
TPM5_4	0.298507	0.003211	92.97	<.0001
TPM5_5	0.387707	0.003928	98.70	<.0001
TPM5_6	0.294241	0.002163	136.02	<.0001
TPM6_1	0.005954	0.000337	17.67	<.0001
TPM6_2	0.007939	0.000400	19.87	<.0001
TPM6_3	0.006237	0.000354	17.64	<.0001
TPM6_4	0.294507	0.003291	89.49	<.0001
TPM6_5	0.392902	0.004027	97.58	<.0001
TPM6_6	0.292461	0.002318	126.17	<.0001
MU1_1	-1.002846	0.005048	-198.67	<.0001
MU2_1	-5.994957	0.008460	-708.59	<.0001
MU3_1	-10.977249	0.014627	-750.46	<.0001
MU4_1	6.036520	0.023829	253.33	<.0001
MU5_1	12.015924	0.019460	617.48	<.0001
MU6_1	19.994009	0.018676	1070.59	<.0001
SIGMA1_1_1	1.007495	0.007790	129.34	<.0001
SIGMA2_1_1	1.987658	0.023807	83.49	<.0001
SIGMA3_1_1	3.042159	0.033805	89.99	<.0001
SIGMA4_1_1	4.037455	0.053806	75.04	<.0001
SIGMA5_1_1	4.881128	0.097545	50.04	<.0001
SIGMA6_1_1	6.082481	0.063053	96.47	<.0001

The following statements explicitly show how the states are relabeled by using the LABELSWITCH=(OUT=) option:

```
proc hmm data=mylib.labelSwitchDGP labelSwitch=(out=mylib.olabel1);
   id time=t section=sec;
   model x / type=gaussian nstate=6;
   optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
   initial mu={20, 12, 6, -1, -6, -11};
run;

data olabel1; set mylib.olabel1; run;
proc sort data=olabel1; by oldStateLabel; run;
proc print data=olabel1 noobs label; run;
```

The first row in Figure 20.93 shows that the old state label "1" is relabeled as the new state "6"; that is, N(20, 6) was labeled as state 1 but is relabeled as state 6. The same logic applies to other rows. The second row in Figure 20.93 shows that the old state label "2" is relabeled as the new state "5"; the third row shows the old state label "3" is relabeled as the new state "4"; and so on. Note that all parameter estimates (ISPV, TPM, and so on) are permuted accordingly after the states are relabeled.

Figure 20.93 Old and New Labels of States

Old State Label	New State Label
1	6
2	5
3	4
4	1
5	2
6	3

The following statements request that the states be labeled in ascending order of the mean parameter estimates. As shown in Figure 20.94, all parameters are permuted according to the new state labels.

```
proc hmm data=mylib.labelSwitchDGP labelSwitch=(sort=asc(mu) out=mylib.olabel2);
   id time=t section=sec;
   model x / type=gaussian nstate=6;
   optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
   initial mu={20, 12, 6, -1, -6, -11};
run;
```

Figure 20.94 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.294461	0.002483	118.61	<.0001
TPM1_2	0.392673	0.002761	142.23	<.0001
TPM1_3	0.292779	0.002057	142.35	<.0001
TPM1_4	0.005837	0.000367	15.91	<.0001
TPM1_5	0.008252	0.000435	18.99	<.0001
TPM1_6	0.005998	0.000350	17.14	<.0001
TPM2_1	0.295538	0.002251	131.31	<.0001
TPM2_2	0.390263	0.002506	155.74	<.0001
TPM2_3	0.294351	0.001805	163.09	<.0001
TPM2_4	0.006372	0.000331	19.24	<.0001
TPM2_5	0.007509	0.000368	20.40	<.0001
TPM2_6	0.005966	0.000304	19.62	<.0001
TPM3_1	0.295579	0.002419	122.18	<.0001
TPM3_2	0.390938	0.002676	146.08	<.0001
TPM3_3	0.292704	0.001985	147.45	<.0001
TPM3_4	0.006358	0.000369	17.23	<.0001
TPM3_5	0.008122	0.000429	18.93	<.0001
TPM3_6	0.006300	0.000352	17.91	<.0001
TPM4_1	0.005726	0.000356	16.10	<.0001
TPM4_2	0.007747	0.000411	18.83	<.0001
TPM4_3	0.006181	0.000354	17.46	<.0001
TPM4_4	0.297737	0.003360	88.62	<.0001
TPM4_5	0.387390	0.004081	94.94	<.0001
TPM4_6	0.295218	0.002374	124.34	<.0001
TPM5_1	0.006121	0.000320	19.13	<.0001
TPM5_2	0.007708	0.000360	21.42	<.0001
TPM5_3	0.005716	0.000305	18.73	<.0001
TPM5_4	0.298507	0.003211	92.97	<.0001
TPM5_5	0.387707	0.003928	98.70	<.0001
TPM5_6	0.294241	0.002163	136.02	<.0001
TPM6_1	0.006237	0.000354	17.64	<.0001
TPM6_2	0.007939	0.000400	19.87	<.0001
TPM6_3	0.005954	0.000337	17.67	<.0001
TPM6_4	0.294507	0.003291	89.49	<.0001
TPM6_5	0.392902	0.004027	97.58	<.0001
TPM6_6	0.292461	0.002318	126.17	<.0001
MU1_1	-10.977249	0.014627	-750.46	<.0001
MU2_1	-5.994957	0.008460	-708.59	<.0001
MU3_1	-1.002846	0.005048	-198.67	<.0001
MU4_1	6.036520	0.023829	253.33	<.0001
MU5_1	12.015924	0.019460	617.48	<.0001
MU6_1	19.994009	0.018676	1070.59	<.0001
SIGMA1_1_1	3.042159	0.033805	89.99	<.0001
SIGMA2_1_1	1.987658	0.023807	83.49	<.0001
SIGMA3_1_1	1.007495	0.007790	129.34	<.0001
SIGMA4_1_1	4.037455	0.053806	75.04	<.0001
SIGMA5_1_1	4.881128	0.097545	50.04	<.0001
SIGMA6_1_1	6.082481	0.063053	96.47	<.0001

The following statements estimate the bistate tricomponent GM HMM. If you compare the final parameter estimates after optimization that are shown in Figure 20.95 and the parameter estimates that are reported in Figure 20.96, you see that the states and components are relabeled. State labels are exchanged, and the labels of component 1 and 3 for old state 1 are also exchanged, which is clearly shown in Figure 20.97.

```
proc hmm data=mylib.labelSwitchDGP labelSwitch=(out=mylib.olabel3);
   id time=t section=sec;
   model x / type=gaussianMixture nstate=2 ncomponent=3;
   optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
   initial mu={20, 12, 6, -1, -6, -11};
   score outmodel=mylib.omodel;
run;

data olabel3; set mylib.olabel3; run;
proc sort data=olabel3; by oldStateLabel oldComponentLabel; run;
proc print data=olabel3 noobs label; run;
```

Figure 20.95 Final Parameter Estimates after Optimization

Optimization Results	
Parameter	**Estimate**
TPM1_1	0.980249
TPM1_2	0.019751
TPM2_1	0.020196
TPM2_2	0.979804
MCP1_1	0.299927
MCP1_2	0.396951
MCP1_3	0.303122
MCP2_1	0.299444
MCP2_2	0.399150
MCP2_3	0.301406
MU1_1_1	19.994224
MU1_2_1	12.015837
MU1_3_1	6.036275
MU2_1_1	-1.002864
MU2_2_1	-5.994918
MU2_3_1	-10.977185
SIGMA1_1_1_1	6.081830
SIGMA1_2_1_1	4.882495
SIGMA1_3_1_1	4.037831
SIGMA2_1_1_1	1.007494
SIGMA2_2_1_1	1.987574
SIGMA2_3_1_1	3.042284

Figure 20.96 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.979804	0.000315	3107.71	<.0001
TPM1_2	0.020196	0.000315	64.06	<.0001
TPM2_1	0.019751	0.000308	64.04	<.0001
TPM2_2	0.980249	0.000308	3178.10	<.0001
MCP1_1	0.299444	0.001126	265.87	<.0001
MCP1_2	0.399150	0.001943	205.44	<.0001
MCP1_3	0.301406	0.001712	176.01	<.0001
MCP2_1	0.303122	0.002797	108.36	<.0001
MCP2_2	0.396951	0.003540	112.13	<.0001
MCP2_3	0.299927	0.001544	194.29	<.0001
MU1_1_1	-1.002864	0.005048	-198.67	<.0001
MU1_2_1	-5.994918	0.008460	-708.63	<.0001
MU1_3_1	-10.977185	0.014627	-750.48	<.0001
MU2_1_1	6.036275	0.023837	253.23	<.0001
MU2_2_1	12.015837	0.019464	617.34	<.0001
MU2_3_1	19.994224	0.018675	1070.63	<.0001
SIGMA1_1_1_1	1.007494	0.007789	129.35	<.0001
SIGMA1_2_1_1	1.987574	0.023804	83.50	<.0001
SIGMA1_3_1_1	3.042284	0.033806	89.99	<.0001
SIGMA2_1_1_1	4.037831	0.053783	75.08	<.0001
SIGMA2_2_1_1	4.882495	0.097590	50.03	<.0001
SIGMA2_3_1_1	6.081830	0.063045	96.47	<.0001

Figure 20.97 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	2	3
1	2	2	2
1	3	2	1
2	1	1	1
2	2	1	2
2	3	1	3

The following statements specify the INMODEL= option in the SCORE statement, which causes the LABELSWITCH=(SORT=NONE) option to be applied by default. It is not surprising to see that the parameter estimates in Figure 20.98 are exactly same as those in Figure 20.96; that is, labels of states and components are not reordered, as confirmed by Figure 20.99.

```
proc hmm data=mylib.labelSwitchDGP
         labelSwitch=(out=mylib.olabel4);;
   score inmodel=mylib.omodel;
run;
```

```
data olabel4; set mylib.olabel4; run;
proc sort data=olabel4; by oldStateLabel oldComponentLabel; run;
proc print data=olabel4 noobs label; run;
```

Figure 20.98 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.979804	0.000315	3107.71	<.0001
TPM1_2	0.020196	0.000315	64.06	<.0001
TPM2_1	0.019751	0.000308	64.04	<.0001
TPM2_2	0.980249	0.000308	3178.10	<.0001
MCP1_1	0.299444	0.001126	265.87	<.0001
MCP1_2	0.399150	0.001943	205.44	<.0001
MCP1_3	0.301406	0.001712	176.01	<.0001
MCP2_1	0.303122	0.002797	108.36	<.0001
MCP2_2	0.396951	0.003540	112.13	<.0001
MCP2_3	0.299927	0.001544	194.29	<.0001
MU1_1_1	-1.002864	0.005048	-198.67	<.0001
MU1_2_1	-5.994918	0.008460	-708.63	<.0001
MU1_3_1	-10.977185	0.014627	-750.48	<.0001
MU2_1_1	6.036275	0.023837	253.23	<.0001
MU2_2_1	12.015837	0.019464	617.34	<.0001
MU2_3_1	19.994224	0.018675	1070.63	<.0001
SIGMA1_1_1_1	1.007494	0.007789	129.35	<.0001
SIGMA1_2_1_1	1.987574	0.023804	83.50	<.0001
SIGMA1_3_1_1	3.042284	0.033806	89.99	<.0001
SIGMA2_1_1_1	4.037831	0.053783	75.08	<.0001
SIGMA2_2_1_1	4.882495	0.097590	50.03	<.0001
SIGMA2_3_1_1	6.081830	0.063045	96.47	<.0001

Figure 20.99 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	1	1
1	2	1	2
1	3	1	3
2	1	2	1
2	2	2	2
2	3	2	3

However, you can reorder the labels of states and components in whichever way you prefer by using the LABELSWITCH=(REORDER=) and LABELSWITCH=(REORDERCOMPONENT=) options, as illustrated in the following statements:

```
proc hmm data=mylib.labelSwitchDGP
         labelSwitch=(reorder=(2 1) reorderComponent=(2 1 3 2 3 1)
                      out=mylib.olabel5);
   score inmodel=mylib.omodel;
run;

data olabel5; set mylib.olabel5; run;
proc sort data=olabel5; by oldStateLabel oldComponentLabel; run;
proc print data=olabel5 noobs label; run;
```

The parameter estimates are shown in Figure 20.100, and the label comparison is shown in Figure 20.101.

Figure 20.100 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.980249	0.000308	3178.10	<.0001
TPM1_2	0.019751	0.000308	64.04	<.0001
TPM2_1	0.020196	0.000315	64.06	<.0001
TPM2_2	0.979804	0.000315	3107.71	<.0001
MCP1_1	0.299927	0.001544	194.29	<.0001
MCP1_2	0.303122	0.002797	108.36	<.0001
MCP1_3	0.396951	0.003540	112.13	<.0001
MCP2_1	0.399150	0.001943	205.44	<.0001
MCP2_2	0.299444	0.001126	265.87	<.0001
MCP2_3	0.301406	0.001712	176.01	<.0001
MU1_1_1	19.994224	0.018675	1070.63	<.0001
MU1_2_1	6.036275	0.023837	253.23	<.0001
MU1_3_1	12.015837	0.019464	617.34	<.0001
MU2_1_1	-5.994918	0.008460	-708.63	<.0001
MU2_2_1	-1.002864	0.005048	-198.67	<.0001
MU2_3_1	-10.977185	0.014627	-750.48	<.0001
SIGMA1_1_1_1	6.081830	0.063045	96.47	<.0001
SIGMA1_2_1_1	4.037831	0.053783	75.08	<.0001
SIGMA1_3_1_1	4.882495	0.097590	50.03	<.0001
SIGMA2_1_1_1	1.987574	0.023804	83.50	<.0001
SIGMA2_2_1_1	1.007494	0.007789	129.35	<.0001
SIGMA2_3_1_1	3.042284	0.033806	89.99	<.0001

Figure 20.101 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	2	2
1	2	2	1
1	3	2	3
2	1	1	2
2	2	1	3
2	3	1	1

The preceding statements illustrate how to use and interpret the LABELSWITCH= option. The statements use both Gaussian HMMs and GM HMMs, which differ as follows: Although it is trivial to know that a K-state Gaussian HMM can be regarded as a special case of a K-state unicomponent GM HMM, a K-state M-component GM HMM can also be regarded as a special case of a KM-state Gaussian HMM. In most cases, a KM-state Gaussian HMM has many more parameters (because of the huge transition probability matrix) than a K-state M-component GM HMM has. For the preceding statements, all information criteria, although not reported here, prefer the bistate tricomponent GM HMM (which the true data generating process follows) to the six-state Gaussian HMM.

Analytic Store Technology

Scoring new data is a common and important task in deploying a model. You can score new data by using the SCORE statement in the HMM procedure, or you can do it by taking advantage of the analytic store technology. By using the STORE statement, you can save estimated models in an analytic store table. Then, you can download the table to a local analytic store file. Later, you can load the model and score new data on any platform that supports the analytic store technology—and you do not even need to have the HMM procedure available on those platforms.

The following example illustrates how to use the analytic store technology.

Generate your training data and upload them to the library. The following statements assume that your engine libref is named mylib, but you can substitute any appropriately defined libref.

```
%let pi1 = 0.5;
%let a11 = 0.2;
%let a22 = 0.2;
%let mu1_1 = 1;
%let mu1_2 = 1;
%let sigma1_11 = 1;
%let sigma1_21 = 0;
%let sigma1_22 = 1;
%let mu2_1 = -1;
%let mu2_2 = -1;
%let sigma2_11 = 1;
%let sigma2_21 = 0;
%let sigma2_22 = 1;

%macro dgpGHMM(table,T,nSections,seed);
   data &table;
      retain cd1_11 cd1_21 cd1_22 cd2_11 cd2_21 cd2_22;
      secLen = ceil(&T./&nSections.);
      do t = 1 to &T.;
         sec = floor((t-1)/secLen)+1;
         if(t=1) then do;
            /* initial probability distribution */
            p = &pi1.;
            /* Cholesky decomposition of sigma1 */
            cd1_11 = sqrt(&sigma1_11.);
            cd1_21 = &sigma1_21./sqrt(&sigma1_11.);
            cd1_22 = sqrt(&sigma1_22.-&sigma1_21.*&sigma1_21./&sigma1_11.);
            /* Cholesky decomposition of sigma2 */
```

```
            cd2_11 = sqrt(&sigma2_11.);
            cd2_21 = &sigma2_21./sqrt(&sigma2_11.);
            cd2_22 = sqrt(&sigma2_22.-&sigma2_21.*&sigma2_21./&sigma2_11.);
         end;
         else do;
            /* transition probability matrix */
            if(lags=1) then p = &a11.;
            else p = 1-&a22.;
         end;
         u = uniform(&seed.);
         if(u<=p) then s=1;
         else s = 2;
         e1 = normal(&seed.); e2 = normal(&seed.);
         if(s=1) then do;
            /* (x,y) ~ N(mu1, Sigma1) at state 1 */
            x = &mu1_1. + cd1_11*e1;
            y = &mu1_2. + cd1_21*e1+cd1_22*e2;
         end;
         else do;
            /* (x,y) ~ N(mu2, Sigma2) at state 2 */
            x = &mu2_1. + cd2_11*e1;
            y = &mu2_2. + cd2_21*e1+cd2_22*e2;
         end;
         output;
         lags = s;
      end;
   run;
   proc sort data=&table.; by sec t; run;
%mend;

* prepare the training data;
%let T = 10000;
%let nSections = 10;
%let seed = 1234;
%dgpGHMM(gHmmTrain,&T.,&nSections.,&seed.);
data mylib.gHmmTrain; set gHmmTrain; run;
```

First, you estimate the model and store it in an analytic store table by using the STORE statement as follows: You must perform this step in the HMM procedure within a CAS session.

```
* estimate the model and save it in an analytic store table;
proc hmm data=mylib.gHmmTrain labelSwitch=(sort=desc(mu));
   id time=t section=sec;
   model x y / type=gaussian nstate=2;
   store out=mylib.ghmmModel;
run;
```

Then, you produce a local analytic store file as follows. You must perform this step in a CAS session that has access to the mylib.ghmmModel data table.

```
* produce a local analytic store file;
proc astore;
    download rstore=mylib.ghmmModel store="ls_ghmm";
run;
```

Now, you can use the local analytic store file to score new data on the platform of interest. For example, you can score new data as in the following code. In the SETOPTION statements, you set the DECODE option to 1 and the DECODE_WINDOW option to 8 to request the decoding results for an 8-unit time window; you set the EVALUATE option to 1 to request the evaluation up to current observation; that is, the log likelihood; you set the FILTER option to 1 to request the filtered probabilities up to current observation; you set the FORECAST option to 1, the FORECAST_ALPHA to 0.8, the FORECAST_COV to 1, and the FORECAST_LEAD to 4 to request the 4-step-ahead forecasts, their covariance matrices, and their confidence intervals at the significance level 0.80; you set the SMOOTH option to 1 and the SMOOTH_WINDOW option to 12 to request the smoothing results for a 12-unit time window. You use the DESCRIBE statement to output the basic model information and the information about input and output variables. In the SCORE statement, you specify the data to be scored in the DATA= option, the local analytic store file in the STORE= option, and the output table for the scored result in the OUT= option. For more information about the ASTORE procedure, see the chapter "The ASTORE Procedure" in *SAS Viya: Machine Learning Procedures*.

```
* get new data;
%let T = 40;
%let nSections = 2;
%let seed = 12345;
%dgpGHMM(gHmmTest,&T.,&nSections.,&seed.);

* score new data;
proc astore;
    setoption decode 1;
    setoption decode_window 8;
    setoption evaluate 1;
    setoption filter 1;
    setoption forecast 1;
    setoption forecast_alpha 0.8;
    setoption forecast_cov 1;
    setoption forecast_lead 4;
    setoption smooth 1;
    setoption smooth_window 12;
    describe store="ls_ghmm";
    score data=gHmmTest store="ls_ghmm" out=ghmmOut;
run;
```

If you need to score new data within a session, you can upload the analytic store file first, and then score the new data by using the RSTORE= option in the ASTORE procedure, as in the following code. Of course, if your session has access to the original analytic store table mylib.ghmmModel, you can skip the uploading step and directly use the option RSTORE=mylib.ghmmModel.

```
* get new data;
%let T = 40;
%let nSections = 2;
%let seed = 12345;
%dgpGHMM(gHmmTest,&T.,&nSections.,&seed.);
data mylib.gHmmTest; set gHmmTest; run;

* upload the local analytic store;
proc astore;
    upload store="ls_ghmm" rstore=mylib.rs_ghmm;
run;
```

```
* score new data;
proc astore;
    setoption decode 1;
    setoption decode_window 8;
    setoption evaluate 1;
    setoption filter 1;
    setoption forecast 1;
    setoption forecast_alpha 0.8;
    setoption forecast_cov 1;
    setoption forecast_lead 4;
    setoption smooth 1;
    setoption smooth_window 12;
    describe rstore=mylib.rs_ghmm;
    score data=mylib.gHmmTest rstore=mylib.rs_ghmm out=mylib.ghmmCasOut;
run;
```

To score new data, you can use the SCORE statement in the HMM procedure, or you can use the STORE statement in the HMM procedure and then use the ASTORE procedure or other products that support the analytic store technology. Sometimes the two methods do not generate the same inference results for the same data set, because these methods treat the data in different ways. That is, using the SCORE statement in the HMM procedure reads all observations in the data set and then scores them; this is the so-called batch mode. In contrast, the analytic store technology deals with observations one by one, which is extremely suitable for streaming analytics; this is the so-called online mode. The following list compares the different kinds of results that these two methods produce:

- For evaluation and filtered results, the two methods are the same.
- For smoothed and decoded results, the two methods are different. Let T be the sample size. For smoothing, in batch mode, $p(S_\tau = i|y_1, \ldots, y_T), 1 \leq \tau \leq T$, can be calculated; but in online mode, at time t, only $y_1, \ldots, y_t$ are available, and then $p(S_\tau = i|y_1, \ldots, y_t)$ can be calculated only if $\tau \leq t$. Because of the memory decay of the Markovian structure, $p(S_\tau = i|y_1, \ldots, y_t)$ can approximate $p(S_\tau = i|y_1, \ldots, y_T)$ very well when $t - \tau$ is large enough. When you set the value of the SMOOTH_WINDOW option to w in the SETOPTION statement in PROC ASTORE, the smoothed probabilities $p(S_\tau = i|y_1, \ldots, y_t), \tau = t - w, \ldots, t, i = 1, \ldots, K$, are calculated and output, where K is the number of states. In some literature, the window length is also called the detection delay, because $p(S_{t-w} = i|y_1, \ldots, y_t)$ is assumed to be close enough to $p(S_{t-w} = i|y_1, \ldots, y_T)$; that is, additional data after time t would not increase the accuracy of the estimation of the smoothed probabilities at time $t - w$. The same logic can be applied to the decoded results.
- For forecasted results, the two methods are the same if there are no exogenous variables in the models; otherwise, the methods are different only if the values of the exogenous variables at time $t + h, h > 0$, are needed in some forecast at time t. Let x_t denote the exogenous variables at time t. In batch mode, the h-step-ahead forecast $p(y_{t+h}|y_1, \ldots, y_t; x_1, \ldots, x_{t+h}), h > 0$, can be calculated when $t + h \leq T$ or extra exogenous variable values after T are provided. However, in online mode, $x_{t+h}, h > 0$, is not available at time t, and then some forecasts that need x_{t+h} are not calculable. In practice, in order to use the analytic store technology in online mode, you must make sure that the h-step-ahead forecast of interest is calculable. For example, by using the NOCURRENTX option in the MODEL statement in PROC HMM, and later by setting the FORECAST_LEAD option to 1 in the SETOPTION statement in PROC ASTORE, you can get the one-step-ahead forecast of dependent variables.

Information Criteria

When you compare models, choose the model that has the smallest information criterion values. The information criteria include Akaike's information criterion (AIC), the corrected Akaike's information criterion (AICC), the Bayesian information criterion (BIC, also referred to as the Schwarz Bayesian criterion, SBC), and the Hannan-Quinn criterion (HQC). These criteria are defined as

$$\begin{aligned} \text{AIC} &= -2\ell + 2r \\ \text{AICC} &= -2\ell + 2rT/(T-r-1) \\ \text{BIC} &= -2\ell + r\log(T) \\ \text{HQC} &= -2\ell + 2r\log(\log(T)) \end{aligned}$$

where ℓ is the log likelihood, r is the total number of parameters in the model, and T is the number of observations that are used to estimate the model.

Missing Values

Exogenous variables cannot have missing values in all types of models. Missing values of dependent variables are treated as follows:

- For Gaussian HMMs and Gaussian mixture HMMs, all or some of the dependent variables in an observation can be missing. The estimation and inference are based on all observations. For example, if the maximum likelihood method is applied, the likelihood of the nonmissing observations and the partially nonmissing observations is calculated and maximized.
- For the regime-switching regression models, all or some of the dependent variables in an observation can be missing. The estimation and inference are based on the observations from the beginning to the last nonmissing or partially nonmissing observation. For example, if the maximum likelihood method is applied, the likelihood of the nonmissing observations and the partially nonmissing observations is calculated and maximized. For the purpose of forecasting, at the end of the data table (or the end of each section if you specify the SECTION= option), the dependent variables can be set as missing values while the exogenous variables are provided.
- For the regime-switching autoregression models, missing values in the dependent variables are not supported. But at the end of the data table (or the end of each section if you specify the SECTION= option), for the purpose of forecasting, the dependent variables can be set as missing values while the exogenous variables are provided.

Data Table Output

The HMM procedure can create the data tables that are specified in the OUT= suboption of the LABELSWITCH= option in the PROC HMM statement; the OUTSTAT= option in the PROC HMM statement; the OUT= options in the DECODE, EVALUATE, FILTER, and SMOOTH statements; the OUT= and OUTALL= options in the ESTIMATE and FORECAST statements; and the OUTMODEL= option in the SCORE statement. The column information for each table is described in the following sections.

OUT= Data Table Generated from the LABELSWITCH= Option in the PROC HMM Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
OldStateLabel	old label of the state
OldComponentLabel	old label of the component for a GM HMM
NewStateLabel	new label of the state
NewComponentLabel	new label of the component for a GM HMM

OUTSTAT= Data Table Generated from the PROC HMM Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
LogLikelihood	log likelihood
AIC	Akaike's information criterion
AICC	corrected Akaike's information criterion
BIC	Bayesian information criterion (also referred to as the Schwarz Bayesian criterion, SBC)
HQC	Hannan-Quinn criterion

OUT= Data Table Generated from the DECODE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State	state value in the most possible path

OUT= Data Table Generated from the ESTIMATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
Index	index of the parameter
Type	type of values in the State$k$ variables: EST for estimate and STD for standard error
Parameter	parameter name
State$k$	value for state k, $k = 1, \ldots, K$, where K is the number of states if you specify the NSTATE=K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option

OUTALL= Data Table Generated from the ESTIMATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
Index	index of the parameter
Parameter	parameter name
Estimate	parameter estimate
StdErr	standard error

COV_n	value for column n, $n = 1, \ldots, N$, in the covariance matrix of parameter estimates, where N is the number of parameters if only one model is estimated or scored; otherwise, N is the maximum of numbers of parameters for all models

OUT= Data Table Generated from the EVALUATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
LogLikelihood	log likelihood

OUT= Data Table Generated from the FILTER Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State$k$	filtered probability value for state k, $k = 1, \ldots, K$, where K is the number of states if you specify the NSTATE=K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option

OUT= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement

Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
Step	value of h in the h-step-ahead forecast
State$k$	forecasted probability value for state k, $k = 1, \ldots, K$, where K is the number of states if you specify the NSTATE=K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
{DVp}_Forecast	mean forecast for dependent variable p, $p = 1, \ldots, k_y$, where k_y is the number of dependent variables; {DVp} stands for the name of dependent variable p
Q1	first quantile value, $(1 - \alpha)/2$, where α is specified in the ALPHA=α option in the FORECAST statement
{DVp}_Q1	first quantile forecast for dependent variable p, $p = 1, \ldots, k_y$, where k_y is the number of dependent variables; {DVp} stands for the name of dependent variable p
Q2	second quantile value, $(1 + \alpha)/2$, where α is specified in the ALPHA=α option in the FORECAST statement
{DVp}_Q2	second quantile forecast for dependent variable p, $p = 1, \ldots, k_y$, where k_y is the number of dependent variables; {DVp} stands for the name of dependent variable p
Q3	third quantile value, $1/2$, for median
{DVp}_Q3	median forecast for dependent variable p, $p = 1, \ldots, k_y$, where k_y is the number of dependent variables; {DVp} stands for the name of dependent variable p

OUTALL= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
Step	value of h in the h-step-ahead forecast
Index	index of the dependent variable
Variable	dependent variable name
State$k$	forecasted probability value for state k, $k = 1, \ldots, K$, where K is the number of states if you specify the NSTATE=K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
Forecast	mean forecast for the dependent variable
StdErr	standard error of forecast for the dependent variable
LowerCL	lower confidence limit of forecast for the dependent variable

UpperCL	upper confidence limit of forecast for the dependent variable
Median	median forecast for the dependent variable
COV_{DVp}	value for column p, $p = 1, \ldots, k_y$, in the covariance matrix of forecasts of dependent variables, where k_y is the number of dependent variables; {DVp} stands for the name of the dependent variable p

OUTMODEL= Data Table Generated from the SCORE Statement

The output data table consists of binary large object columns to store the binary data, which should be consumed only by the INMODEL= option in the SCORE statement. The number of columns and the definition of columns might be changed without notice in future releases. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters because the data type of each column is binary.

OUT= Data Table Generated from the SMOOTH Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option
YLag	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=$n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State$k$	smoothed probability value for state k, $k = 1, \ldots, K$, where K is the number of states if you specify the NSTATE=K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE=$n_1 : n_2$ option

ODS Table Names

The HMM procedure assigns a name to each table that it creates. You can use this name to refer to the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in Table 20.6.

Table 20.6 ODS Tables Produced in the HMM Procedure

ODS Table Name	Description	Option
Algorithm	Option values for the optimization algorithm	PRINTLEVEL=n, where n =2 or 3
ARCoef	Coefficients of autoregressive variables for a regime-switching autoregression model	Default

Table 20.6 *continued*

ODS Table Name	Description	Option
ConstCoef	Coefficients of constants for a regime-switching regression model or regime-switching autoregression model	Default
Cov	Covariance matrices for a regime-switching regression model or regime-switching autoregression model	Default
CPM	Category probability matrix for a finite HMM	Default
ExogCoef	Coefficients of exogenous variables for a regime-switching regression model or regime-switching autoregression model	Default
FinalObjectiveFunction	Objective function value at the end of the optimization	PRINTLEVEL=3
FinalParameterEstimates	Final parameter values at the end of the optimization	PRINTLEVEL=3
FitStatistics	Log likelihood and information criteria	Default
InitialObjectiveFunction	Objective function value at the start of the optimization	PRINTLEVEL=*n*, where $n = 1, 2,$ or 3
InitialParameterEstimates	Initial parameter values at the start of the optimization	PRINTLEVEL=*n*, where $n = 1, 2,$ or 3
ISPV	Initial state probability vector	Default
IterHistory	Optimization iterations for the active set and interior point algorithms	PRINTITERFREQ=*n*, $n > 0$
Lambda	Mean and variance vectors, λ, for a Poisson HMM	Default
LinearCoef	Coefficients of linear trends for a regime-switching regression model or regime-switching autoregression model	Default
MCP	Mixture component probabilities for a GM HMM	Default
ModelInfo	Model information	Default
Mu	Mean vectors, μ, for a Gaussian or GM HMM	Default
NObs	Observation information	Default
OptIterHistory	Optimization iterations for the SGD algorithm	PRINTITERFREQ=*n*, $n > 0$
ParameterEstimates	Model parameter estimates	Default
PriorHyperparm	Prior hyperparameters	METHOD=MAP

Table 20.6 *continued*

ODS Table Name	Description	Option
QuadCoef	Coefficients of quadratic trends for a regime-switching regression model or regime-switching autoregression model	Default
SeasCoef	Coefficients of seasonal dummies for a regime-switching regression model or regime-switching autoregression model	Default
Sigma	Covariance matrices, σ, for a Gaussian or GM HMM	Default
TPM	Transition probability matrix	Default

Several Types of Models as Special Cases of Hidden Markov Models

Several types of models can be treated as special cases of hidden Markov models, including Gaussian mixture models (GMMs), Gaussian mixture regressions (GMRs), regressions, autoregressive error models, vector autoregressions (VARs) with or without exogenous variables in the standard or mean-adjusted form, and so on. In this section, a few of these models are illustrated.

Gaussian Mixture Model and Gaussian Mixture Regressions

Let $\mathbf{Y}_i = (y_{1i}, \ldots, y_{pi})'$, $i = 1, \ldots, n$, denote a p-dimensional vector of random variables, and $\mathbf{Y}_i$ follows an iid Gaussian mixture distribution,

$$\mathbf{Y}_i \sim \mathrm{GM}(\{w_1, \ldots, w_M\}, \{\mu_1, \ldots, \mu_M\}, \{\Sigma_1, \ldots, \Sigma_M\})$$

where $\mathrm{GM}(.,.,.)$ represents the Gaussian mixture distribution; M is the number of components; and the $w_j, j = 1, \ldots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_j \geq 0, j = 1, \ldots, M$, and $\sum_{j=1}^{M} w_j = 1$). The μ_j and $\Sigma_j, j = 1, \ldots, M$, are the mean and covariance parameters, respectively, for the jth Gaussian component. The preceding model is the Gaussian mixture model (GMM). For more information about GMM, see Frühwirth-Schnatter (2006). For more information about Gaussian mixture regression (GMR) and its relationship to GMM, see Sung (2004).

Consider a bivariate Gaussian mixture model that has the following parameter values:

$$w = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mu_1 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_2 = \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The following statements simulate the bivariate vectors from the previous model to provide training data. In fact, for GMMs, each observation is independent from the others. In order to use the HMM procedure to estimate the GMMs, you must assign each observation in the training data set a unique section ID and a constant time ID; that is, each section contains only one observation.

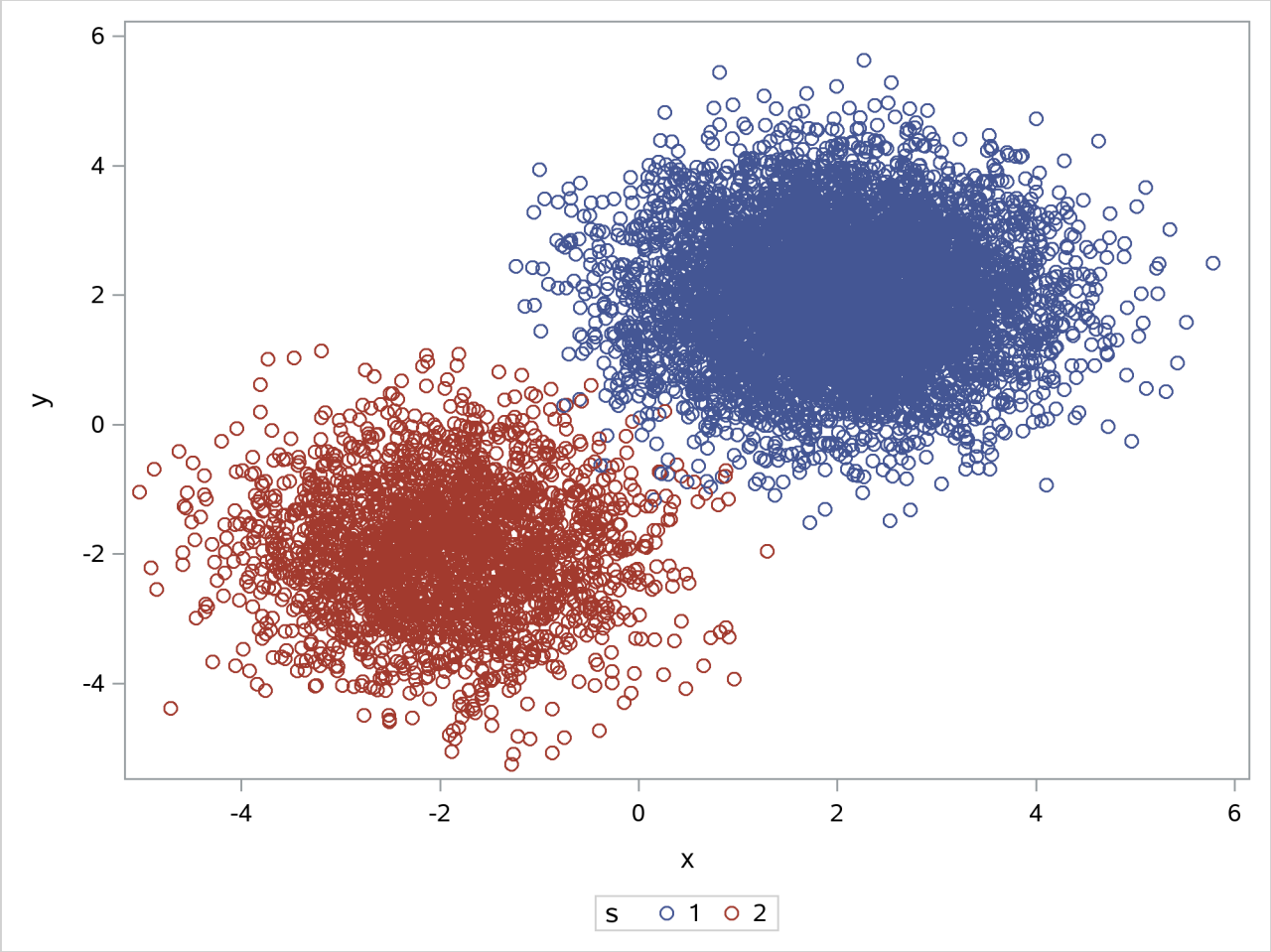
```
data one;
   call streaminit(12345);
   do sec = 1 to 10000;
      t = 1;
      u = rand('UNIFORM');
      if(u<=0.75) then do;
         s = 1;
         x = 2 + rand('NORMAL');
         y = 2 + rand('NORMAL');
      end;
      else do;
         s = 2;
         x = -2 + rand('NORMAL');
         y = -2 + rand('NORMAL');
      end;
      output;
   end;
run;

data mylib.one; set one; run;
```

The following statements plot the observed points (x, y):

```
proc sgplot data=one;
   scatter x=x y=y / group=s;
run;
```

As shown in Figure 20.102, it seems that the points come from two different clusters.

Figure 20.102 Unclassified Observations



The following statements estimate the two-component Gaussian mixture model and then classify the data points through decoding; here decoding means finding the most likely cluster for the observed data. The STORE statement is used to save the model in the analytic store table for future scoring.

```
proc hmm data=mylib.one labelSwitch=(sort=desc(mu));
   id time=t section=sec;
   model x y / nstate=2 type=gaussian estispv;
   decode out=mylib.mydecode;
   store mylib.mygmm;
run;
```

The estimates of all parameters are shown in Figure 20.103, which displays columns for the parameter name, estimate value, standard error, t value, and p-value. The TPM parameters are meaningless in this context. The ISPV parameters are the mixture component probabilities. The parameter estimates are very close to the true parameter values in the DGP.

Figure 20.103 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > |t|
ISPV1	0.741154	0.004402	168.35	<.0001
ISPV2	0.258846	0.004402	58.80	<.0001
TPM1_1	0.500000	0		
TPM1_2	0.500000	0		
TPM2_1	0.500000	0		
TPM2_2	0.500000	0		
MU1_1	2.017677	0.011520	175.14	<.0001
MU1_2	1.998486	0.011714	170.61	<.0001
MU2_1	-1.972142	0.019170	-102.88	<.0001
MU2_2	-1.971479	0.020118	-98.00	<.0001
SIGMA1_1_1	0.971840	0.016189	60.03	<.0001
SIGMA1_2_1	-0.003386	0.011752	-0.29	0.7732
SIGMA1_2_2	1.002999	0.016772	59.80	<.0001
SIGMA2_1_1	0.918582	0.026555	34.59	<.0001
SIGMA2_2_1	-0.009950	0.019897	-0.50	0.6170
SIGMA2_2_2	1.020699	0.029038	35.15	<.0001

The accuracy of classification through the GMM is calculated by the following statements:

```
data mydecode; set mylib.mydecode; run;
proc sort data=mydecode; by sec; run;

data combine;
   merge mydecode(in=a) one(in=b);
   by sec;
run;

data checkAccuracy;
   set combine;
   retain accuracy 0;
   if(state=s) then accuracy+1;
   if(_N_=10000) then do;
      accuracy = accuracy/_N_;
      output;
   end;
   keep accuracy;
run;

proc print data=checkAccuracy noobs; run;
```

The accuracy of the GMM, which is shown in Figure 20.104, is very high. It implies that you can use the HMM procedure not only to learn but also to infer the GMM.

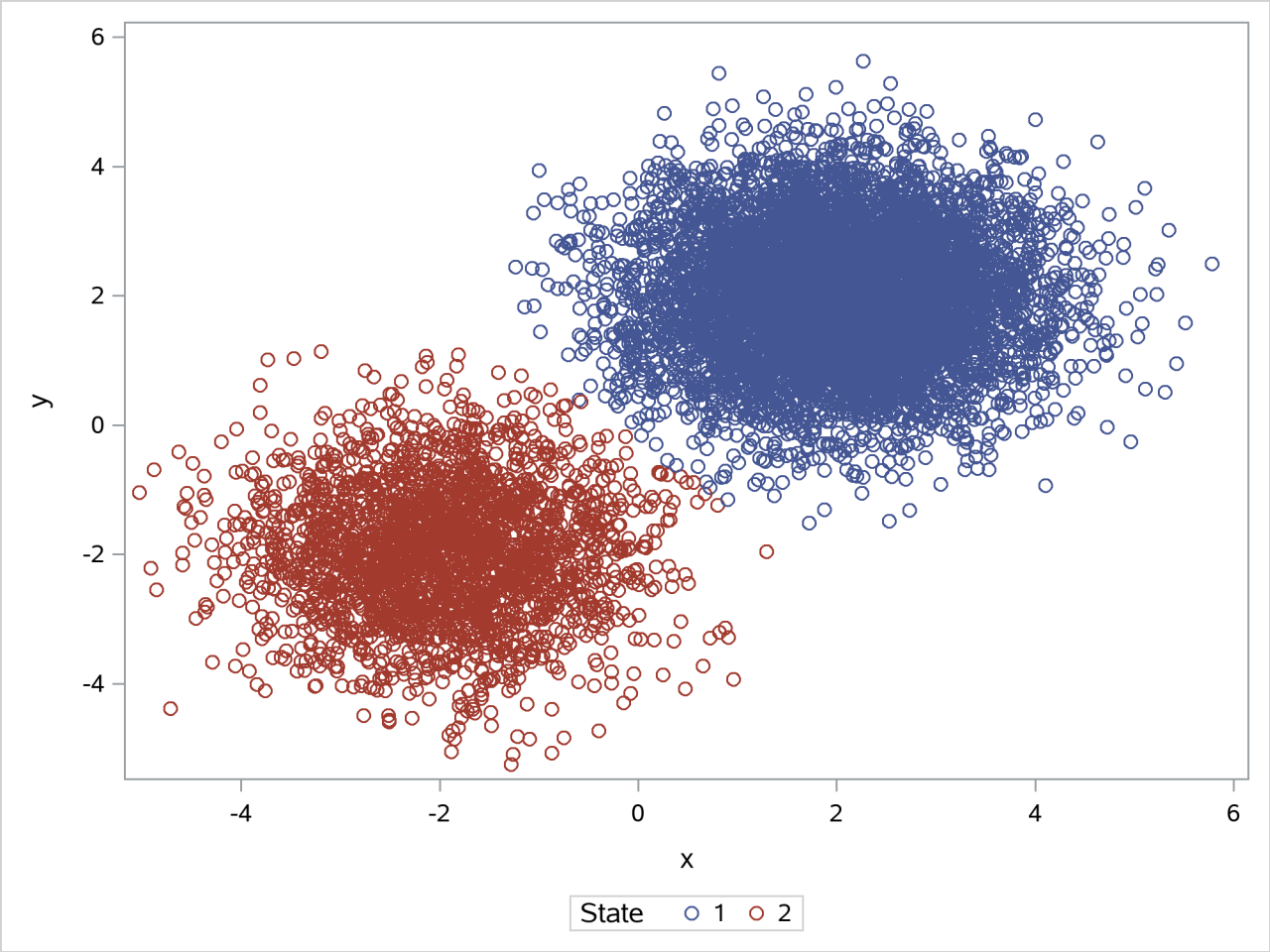
Figure 20.104 Accuracy of GMM on Training Data

accuracy
0.9985

To show how well the data are classified, the following statements plot the classified data points according to the estimated GMM. The plot is shown in Figure 20.105. Comparing it to Figure 20.102, you see that they are almost exactly same.

```
proc sgplot data=combine;
   scatter x=x y=y / group=state;
run;
```

Figure 20.105 Classified Data Points by GMM



It is also easy to apply the estimated GMM to score some new data set by using the analytic store technology. The following statements generate the testing data, apply the ASTORE procedure, and calculate the accuracy of classification. As Figure 20.106 shows, the accuracy is still very high.

```
data two;
   call streaminit(12346);
   do sec = 1 to 10000;
      t = 1;
      u = rand('UNIFORM');
      if(u<=0.75) then do;
         s = 1;
         x = 2 + rand('NORMAL');
```

```
         y = 2 + rand('NORMAL');
      end;
      else do;
         s = 2;
         x = -2 + rand('NORMAL');
         y = -2 + rand('NORMAL');
      end;
      output;
   end;
run;

proc astore;
   download rstore=mylib.mygmm store='ls_gmm';
   setoption decode 1;
   setoption decode_window 1;
   setoption evaluate 0;
   describe store='ls_gmm';
   score data=two store='ls_gmm' out=gmmout;
run;

data combine2;
   merge gmmout(in=a) two(in=b);
   by sec;
run;

data checkAccuracy2;
   set combine2;
   retain accuracy 0;
   if(DecodedState_BackStep0=s) then accuracy+1;
   if(_N_=10000) then do;
      accuracy = accuracy/_N_;
      output;
   end;
   keep accuracy;
run;

proc print data=checkAccuracy2 noobs; run;
```

Figure 20.106 Accuracy of GMM on Testing Data

accuracy
0.998

Regression and Vector Autoregression

When you specify NSTATE=1 in the MODEL statement for regime-switching regression or autoregression, you are dealing with the regression or vector autoregression (VAR). The autoregressive error model can be treated as a univariate VAR in the mean-adjusted form. For more information about the autoregressive error model, see the chapter "The AUTOREG Procedure" in *SAS/ETS User's Guide*. For more information about the VARs with or without exogenous variables, see the chapter "The VARMAX Procedure" in *SAS/ETS User's Guide*.

Examples: HMM Procedure

Example 20.1: Discovering the Hidden Market States by Using the Regime-Switching Autoregression Model

To buy or to sell, that is the question! If you knew it would be a bull market, you might choose to buy the stock; if you knew it would be a bear market, you might choose to sell the stock. However, you cannot directly observe these market states. What you can observe is the stock prices. So, how can you discover the hidden market states from the observable stock price? The question is answered in this example by using regime-switching autoregression models (RS ARs).

Data

Consider the following CRSP NYSE/NYSEMKT/Nasdaq/Arca Value-Weighted Market Index (Center for Research in Security Prices 2018) from January 19, 1926, to December 29, 2017. You can access the CRSP data through the SASECRSP interface engine; for more information, see the chapter "The SASECRSP Interface Engine" in *SAS/ETS User's Guide*. You can also directly access the data if you install the CRSP data locally. If you install the CRSP data in the folder XXX, the following statements extract the daily stock prices (that is, market indices) into the data set vwmiO:

```
*************************************************************************;
* DATA                                                                  ;
*************************************************************************;

title 'Discovering the Hidden Market States';
libname crspInd "XXX";
data vwmiO;
   set crspInd.sfz_dind(where=(KYINDNO=1000200
      and caldt>='19JAN1926'd and caldt<='29DEC2017'd));
   format date MMDDYY10.;
   date = caldt; price = aind;
   keep date price;
run;
```

The stock-price series is nonstationary and not suitable to be directly modeled; hence, the stationary weekly-return series is modeled. The following code generates the weekly returns through the daily stock prices and saves them in two data sets. One data set, vwmi, is the full-sample data set, which contains all weekly returns. The other data set, vwmiIn, is the in-sample data set, which contains weekly returns before December 31, 2000. The weekly returns after January 1, 2001, are in the out-of-sample period and are used for testing the models.

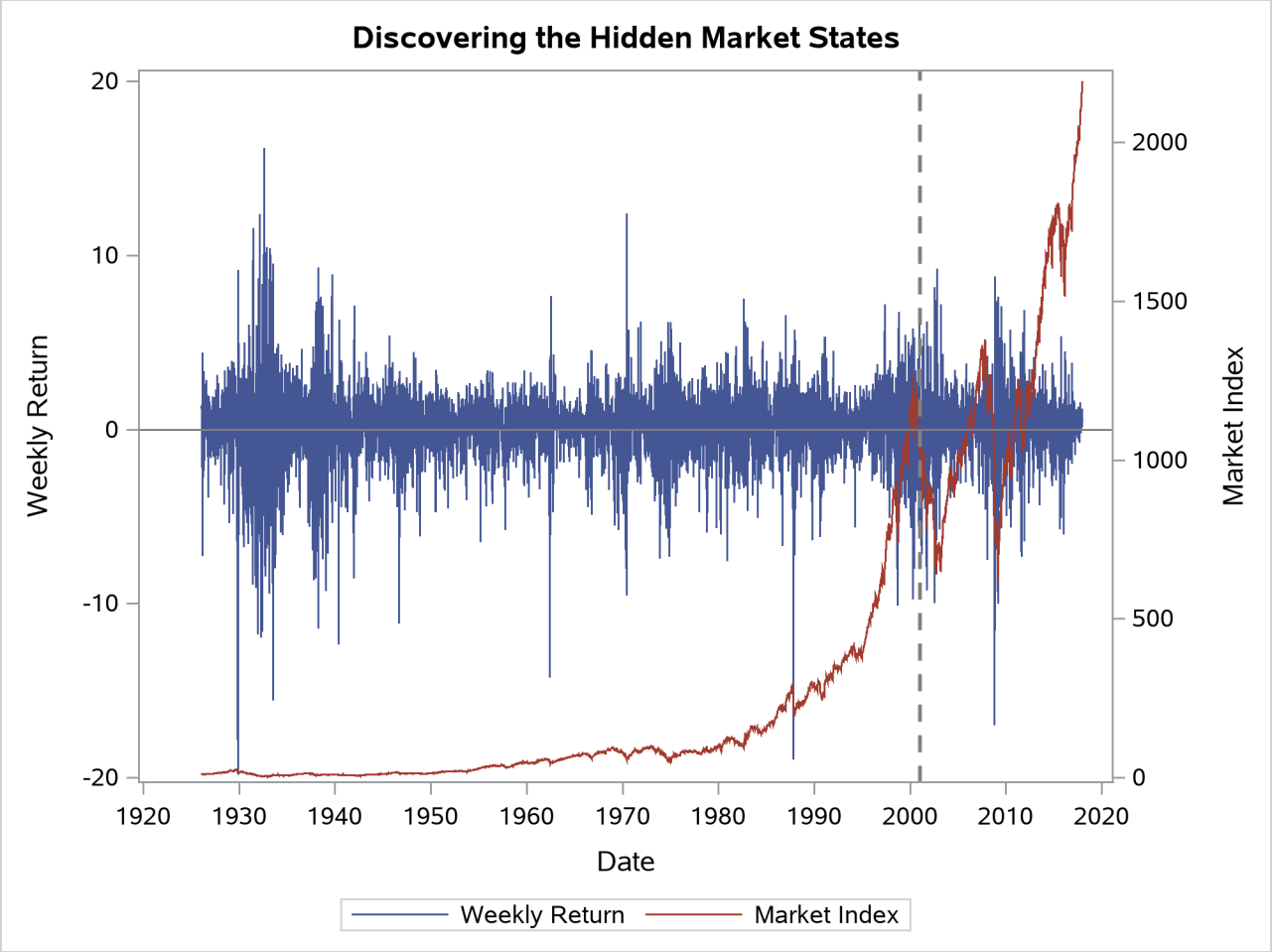
```
%let ds = vwmi;
%let cutDate =  '31DEC2000'd;
data &ds. &ds.In;
   set &ds.O;
   retain cumReturn 0;
   return = (log(price)-log(lag(price)))*100;
   if(return~=.) then cumReturn + return;
```

```
   if(mod(_N_,5)=1 and _N_>1) then do;
      returnw = cumReturn;
      w + 1;
      output &ds.;
      if (date<=&cutDate.) then output &ds.In;
      cumReturn = 0;
   end;
   keep w date returnw price;
   label w="Week Index" date="Date" returnw="Weekly Return"
         price="Market Index";
run;
```

The following statements plot the stock prices and weekly returns, as shown in Output 20.1.1. The in-sample and out-of-sample periods are separated by a vertical line.

```
proc sgplot data=&ds.;
   series x=date y=returnw;
   series x=date y=price / y2axis;
   refline 0 / axis=y lineattrs=(color=gray thickness=1);
   refline &cutDate. / axis=x
                       lineattrs=(color=gray thickness=2 pattern=dash);
   yaxis max=20 min=-20;
run;
```

Output 20.1.1 Stock Prices and Weekly Returns



Before you estimate any model, make a copy of the data in the library, as in the following DATA steps:

```
data mylib.&ds.In; set &ds.In; run;
data mylib.&ds.; set &ds.; run;
```

Analysis with a Simple Model

Consider a simple case. First, assume that there are two hidden market states, bear market state and bull market state, which can be modeled by exactly two corresponding hidden states in an HMM; that is, consider a bistate HMM. Second, assume that the lagged weekly returns cannot help predict current weekly return at each hidden state; that is, the autoregressive order is zero at each hidden state. Hence, as a start, consider a bistate RS-AR(0) model. The following statements estimate the bistate RS-AR(0) model on weekly returns. The input data table, specified in the required DATA= option in the PROC HMM statement, contains the in-sample data. In order to estimate the model, the ID and MODEL statements must be specified. Since the data are sorted by the variable date, the variable date is assigned to the required TIME= option. However, since the data can also be regarded as being sorted by the variable w, the variable w, instead of the variable date, can be assigned to the TIME= option; if you do so, there is no impact on the model estimation. In the MODEL statement, the dependent variable, the weekly return returnw, is specified first; the TYPE= option is set to AR for the RS-AR model; the NSTATE= option is set to 2 for the bistate model; and the YLAG= option

is set to 0 for an AR(0) model. The SMOOTH, DECODE, and EVALUATE statements specify where to output, respectively, the smoothed probabilities, decoded state path, and log likelihood up to each time point.

```
*************************************************************************;
* ANALYSIS WITH A SIMPLE MODEL                                          ;
*************************************************************************;

proc hmm data=mylib.&ds.In;
   id time=date;
   model returnw / type=ar nstate=2 ylag=0;
   smooth out=mylib.&ds.inSmooth_k2To2_p0To0;
   decode out=mylib.&ds.inDecode_k2To2_p0To0;
   evaluate out=mylib.&ds.inEval_k2To2_p0To0;
run;
```

The number of observations and model information are shown in Output 20.1.2.

Output 20.1.2 Number of Observations and Model Information

Discovering the Hidden Market States

Observation Information	
Number of Observations	3999
Number of Missing Observations	0

Model Information	
Type of Model	Regime-Switching Autoregression Model
Stationary	Yes
Number of States	2
Number of Dependent Variables	1
Number of Exogenous Variables	0

The estimates of the initial state probability vector (ISPV) are shown in Output 20.1.3. In this example, the hidden Markov chain is assumed to be stationary. Hence, the steady-state probability distribution (STPD, which is also called the equilibrium distribution or stationary distribution) of the Markov chain is equal to the ISPV.

Output 20.1.3 Estimates of Initial State Probability Vector

Discovering the Hidden Market States

Initial State Probabilities	
State	Estimation
1	0.76747
2	0.23253

The estimates of the transition probability matrix (TPM) are shown in Output 20.1.4.

Output 20.1.4 Estimates of Transition Probability Matrix

Estimated Transition Probability Matrix		
State	**1**	**2**
1	0.98111	0.01889
2	0.06235	0.93765

The estimates of the intercept for each state are shown in Output 20.1.5.

Output 20.1.5 Estimate of Intercept for Each State

Constant	
State	**returnw**
1	0.28710
2	-0.47243

The estimates of the variance for each state are shown in Output 20.1.6.

Output 20.1.6 Estimate of Variance for Each State

Innovations Covariances		
State	**Variable**	**returnw**
1	**returnw**	2.29936
2	**returnw**	17.02648

The estimates of all parameters are shown in Output 20.1.7, which displays columns for the parameter name, estimate value, standard error, t value, and p-value.

Output 20.1.7 Parameter Estimates

Parameter Estimates				
Parameter	**Estimate**	**Standard Error**	**t Value**	**Pr > |t|**
TPM1_1	0.981109	0.003507	279.78	<.0001
TPM1_2	0.018891	0.003507	5.39	<.0001
TPM2_1	0.062352	0.011274	5.53	<.0001
TPM2_2	0.937648	0.011274	83.17	<.0001
CONST1_1	0.287100	0.030130	9.53	<.0001
CONST2_1	-0.472426	0.142740	-3.31	0.0009
COV1_1_1	2.299364	0.090090	25.52	<.0001
COV2_1_1	17.026478	1.052833	16.17	<.0001

The log likelihood and information criteria are shown in Output 20.1.8.

Output 20.1.8 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-8497.46701
AIC	17006.934019
AICC	17006.955061
BIC	17044.696817
HQC	17020.320118

The smoothed probabilities of states for each observation are output in the data table mylib.vwmiInSmooth_k2To2_p0To0. This example discusses one type of inference. The mean and variance (or standard error) of each state can be calculated through the weighted sample, where the smoothed probabilities are the weights. The following statements calculate and print the sample mean and standard error of each state, as shown in Output 20.1.9. In the case of an RS-AR(0) model that has no regressors other than the intercept (which is equivalent to the Gaussian HMM), the sample mean and standard error of each state are almost the same as, respectively, the intercept estimate and the square root of variance estimate of each state.

```
*print sample mean and standard error for each state
 according to the weighted sample,
 where the smoothed probabilities are the weight;
data &ds.inSmooth_k2to2_p0to0;
   set mylib.&ds.inSmooth_k2to2_p0to0;
run;
proc sort data=&ds.inSmooth_k2to2_p0to0; by date; run;
data &ds.inWeighted;
   merge &ds.inSmooth_k2to2_p0to0(in=a) &ds.in(in=b);
   by date;
   if(a=b);
run;
%macro printMeanAndStdErrOfAState(k);
   data &ds.inMeanStdErr;
      set &ds.inWeighted end=eof;
      array cWeights{&k.} _TEMPORARY_;
      array cMeans[&k.] mean1-mean&k.;
      array cVars[&k.] sd1-sd&k.;
      array weights[&k.] state1-state&k.;
      retain cWeights cMeans cVars;
      if(_N_=1) then do;
         do i = 1 to &k.;
            cWeights[i] = weights[i];
            cMeans[i] = weights[i]*returnw;
            cVars[i] = weights[i]*returnw*returnw;
         end;
      end;
      else do;
         do i = 1 to &k.;
            cWeights[i] = cWeights[i] + weights[i];
            cMeans[i] = cMeans[i] + weights[i]*returnw;
            cVars[i] = cVars[i] + weights[i]*returnw*returnw;
         end;
      end;
      if eof then do;
```

```
         do i = 1 to &k.;
            if(cWeights[i]>0) then do;
               cMeans[i] = cMeans[i] / cWeights[i];
               cVars[i] = cVars[i] / cWeights[i] - cMeans[i]*cMeans[i];
               if(cVars[i]>=0) then cVars[i] = sqrt(cVars[i]);
            end;
            else do;
               cMeans[i] = .;
               cVars[i] = .;
            end;
         end;
         %do i = 1 %to &k.;
            call symputx("mu&i.",cMeans[&i.],'G');
            call symputx("sigma&i.",cVars[&i.],'G');
            state = &i.;  mean = mean&i.; stdErr = sd&i.;
            output;
         %end;
      end;
      label state="State" mean="Sample Mean"
            stdErr="Sample Standard Error";
      keep state mean stdErr;
   run;
   proc print data=&ds.inMeanStdErr noobs label; run;
%mend;
%printMeanAndStdErrOfAState(2);
```

Output 20.1.9 Sample Mean and Standard Error of Each State

Discovering the Hidden Market States

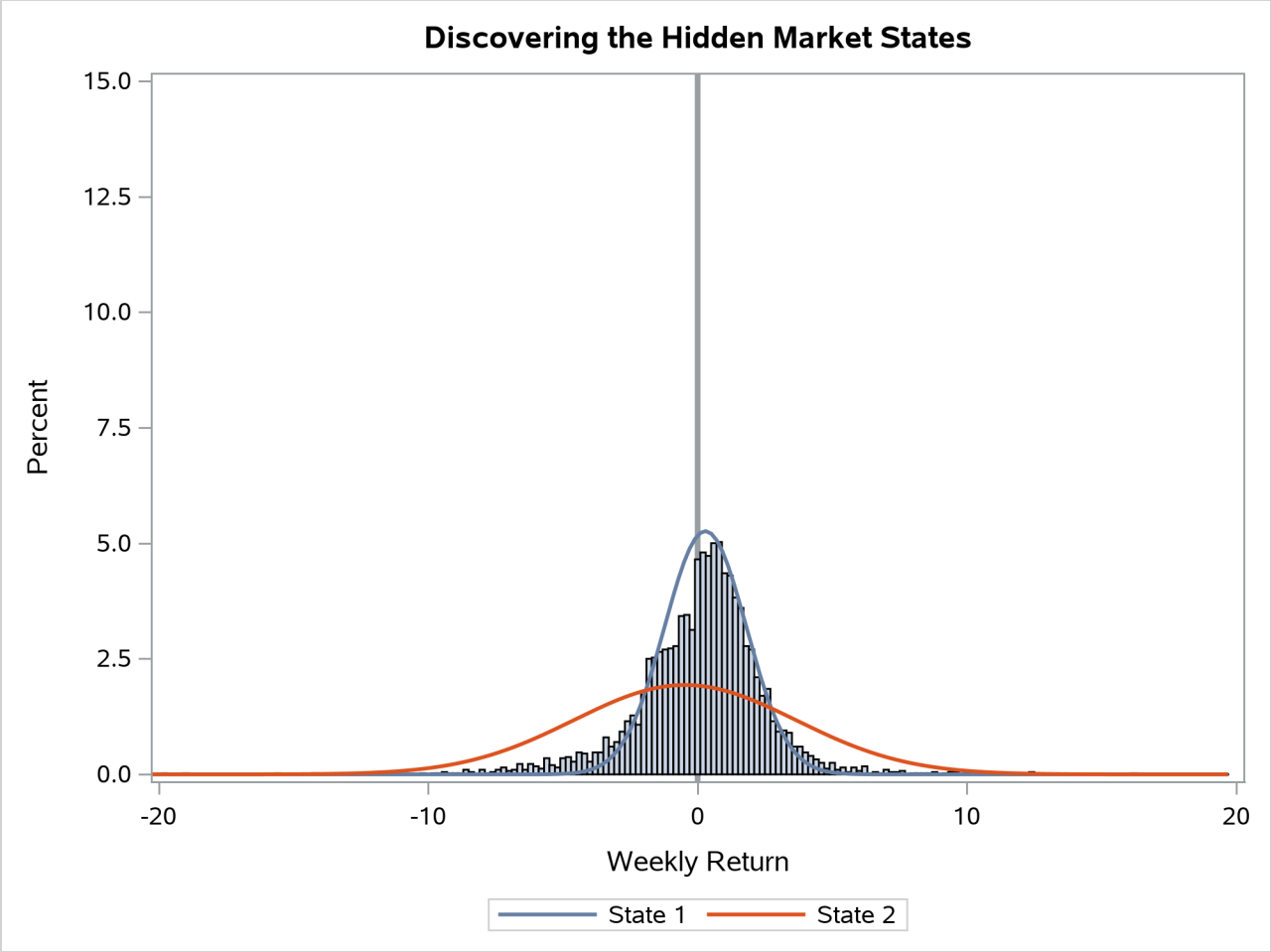
State	Sample Mean	Sample Standard Error
1	0.28710	1.51637
2	-0.47243	4.12632

The following statements plot each state as a Gaussian kernel, which shows how each state looks. As shown in Output 20.1.10, it is obvious that state 1 should represent the bull market state (positive expected return and low risk) and state 2 the bear market state (negative expected return and high risk).

```
%macro plotState(myData,myColumn,ks,ke);
   proc sgplot data=&myData.;
      refline 0 / axis=x lineattrs=(thickness=3);
      histogram &myColumn. / nbins=200;
      %do i = &ks. %to &ke.;
         density &myColumn. / type=normal(mu=&&mu&i. sigma=&&sigma&i.)
            name="state&i."
            legendlabel="State &i.";
      %end;
      keylegend %do i = &ks. %to &ke.; "state&i." %end;;
      xaxis min=-20 max=20 ;
      yaxis min=0 max=15;
   run;
%mend;
```

```
%plotState(&ds.In,returnw,1,2);
```

Output 20.1.10 Gaussian Distribution of Each State

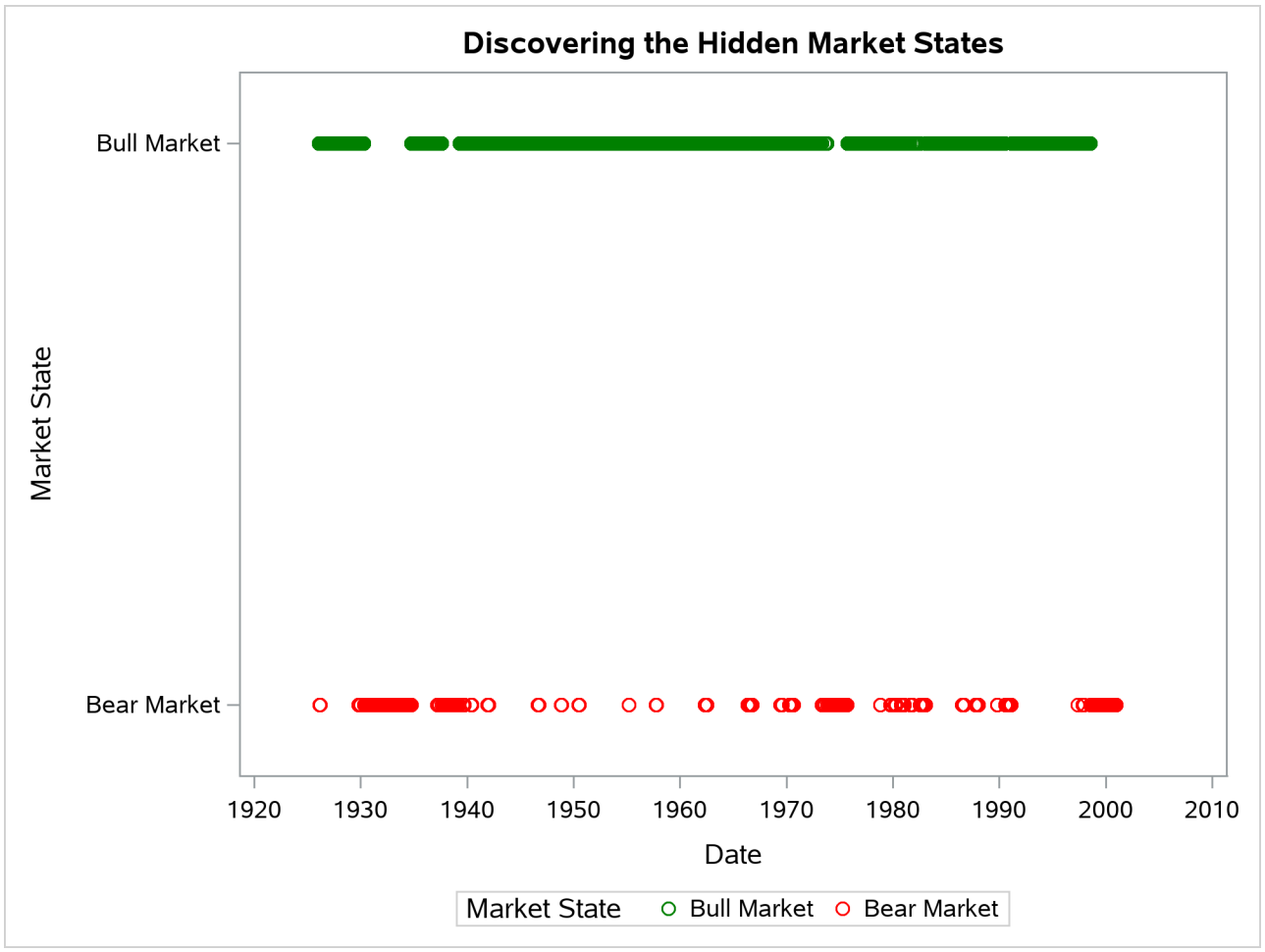


The decoded path of states, which provide a historical view of what happened, given all the available data, is output in the data table mylib.vwmiInDecode_k2To2_p0To0. The decoded path assigns each time point a market state. It seems that the hidden market states have become observable. The following statements plot the decoded market states, as shown in Output 20.1.11:

```
* plot decoded results;
data &ds.InDecode_k2to2_p0to0; set mylib.&ds.InDecode_k2to2_p0to0; run;
proc sort data=&ds.InDecode_k2to2_p0to0; by date; run;
data decodeMaketStates;
   set &ds.In;
   set &ds.InDecode_k2to2_p0to0;
   array returnws returnw1-returnw2;
   array prices price1-price2;
   if(state~=.) then do;
      if(state=1) then do;
         returnws[1] = returnw; prices[1] = price;
         marketState="Bull Market";
      end;
      if(state=2) then do;
```

```
            returnws[2] = returnw; prices[2] = price;
            marketState="Bear Market";
         end;
      end;
      label marketState="Market State";
   run;
   proc sgplot data=decodeMaketStates;
      styleattrs datacontrastcolors=(green red);
      scatter x=date y=marketState / group=marketState;
      yaxis label="Market State" offsetmin=0.1 offsetmax=0.1 reverse;
   run;
```

Output 20.1.11 Decoded Market States

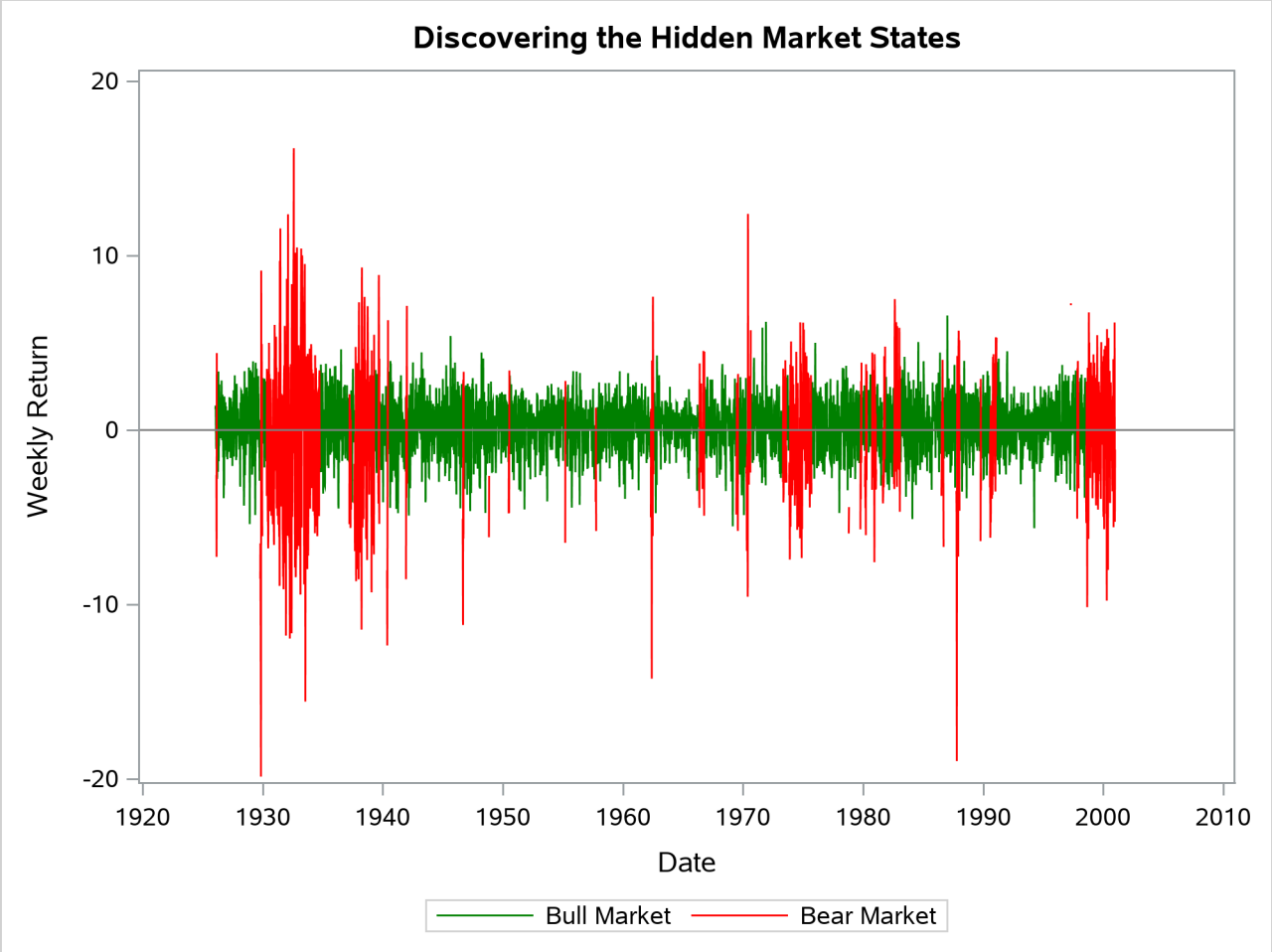


You can incorporate the decoded market states with other variables of interest. The following statements plot the decoded weekly returns, as shown in Output 20.1.12:

```
proc sgplot data=decodeMaketStates;
   series x=date y=returnw1 / break lineattrs=(color=green)
                              legendlabel="Bull Market";
   series x=date y=returnw2 / break lineattrs=(color=red)
                              legendlabel="Bear Market";
   refline 0 / axis=y lineattrs=(color=gray thickness=1);
```

```
   yaxis max=20 min=-20 label="Weekly Return";
run;
```

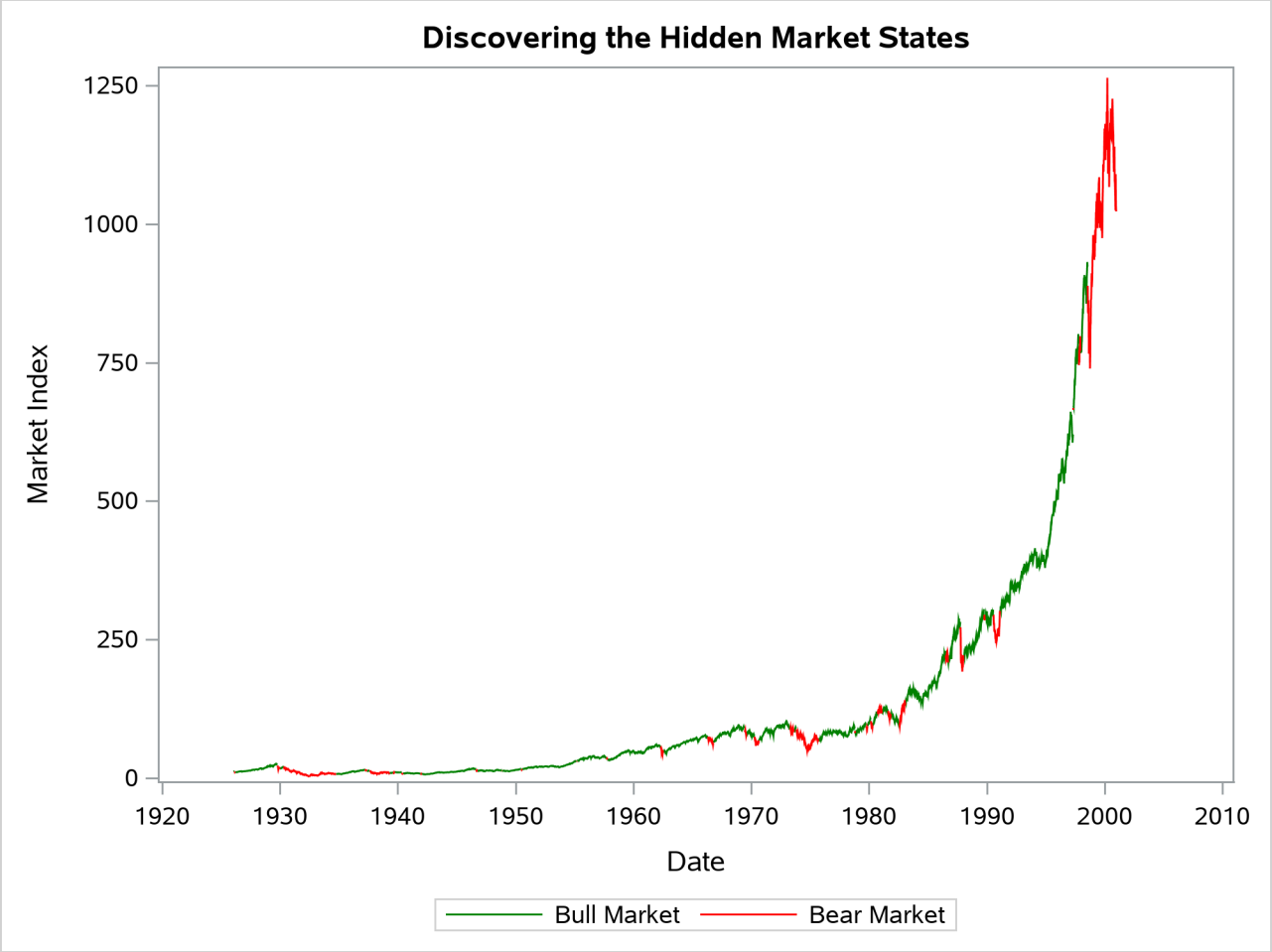
Output 20.1.12 Decoded Weekly Returns



The following statements plot the decoded stock prices, as shown in Output 20.1.13:

```
proc sgplot data=decodeMaketStates;
   series x=date y=price1 / break lineattrs=(color=green)
                            legendlabel="Bull Market";
   series x=date y=price2 / break lineattrs=(color=red)
                            legendlabel="Bear Market";
   yaxis label="Market Index";
run;
```

Output 20.1.13 Decoded Stock Prices

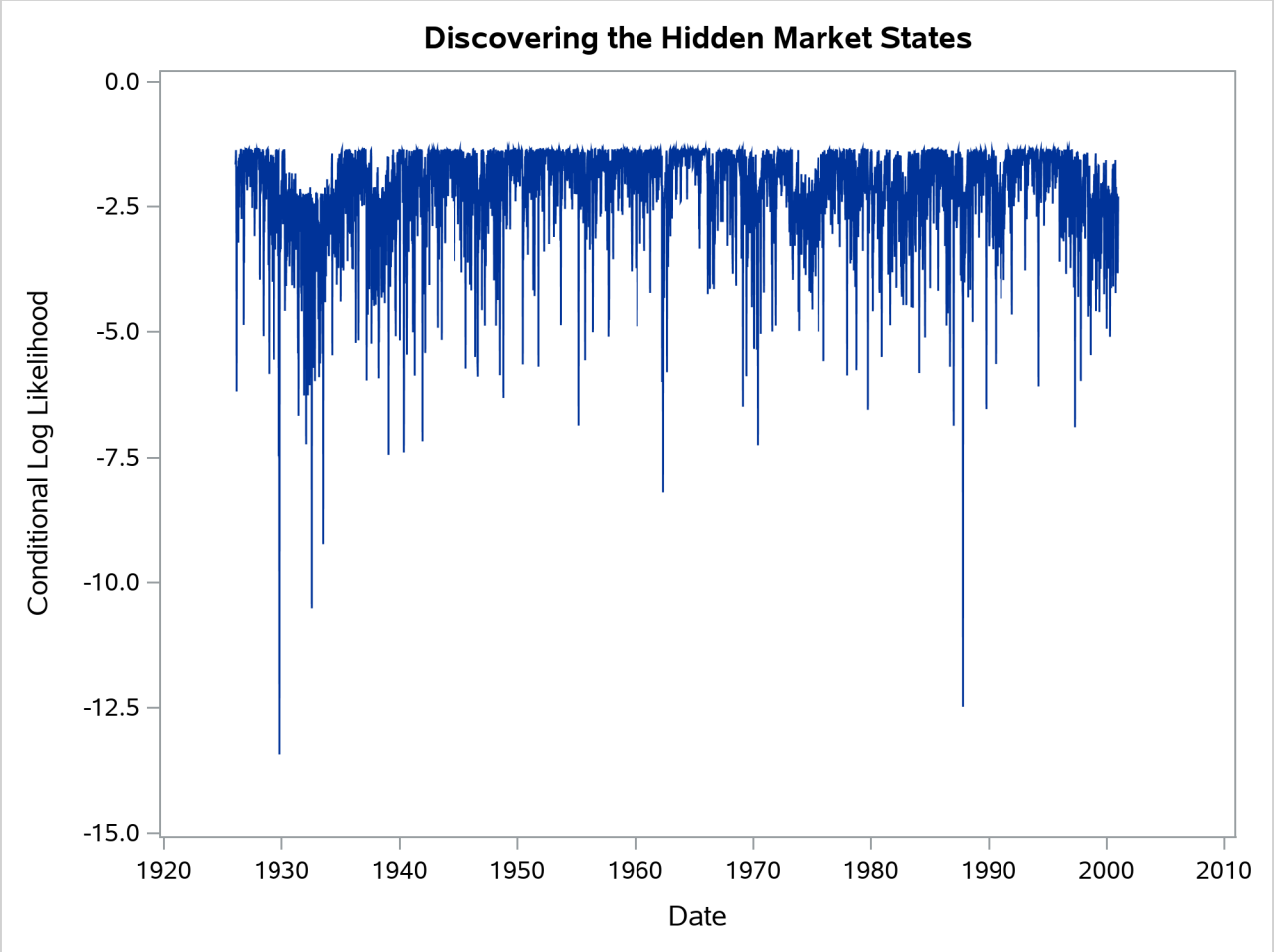


The following statements plot the log likelihood of each observation conditional on its past; the plot is shown in Output 20.1.14.

```
* plot conditional log likelihood;
%macro plotEvaluation(inds, outds, qConditional);
   data &outds.;
      set &inds.;
      format date MMDDYY10.;
   run;
   %if (&qConditional.=1) %then %do;
      proc sort data=&outds.;
         by date;
      run;
      data &outds.;
         set &outds.;
         if(_N_=1) then
            condLogLikelihood = logLikelihood;
         else
            condLogLikelihood = logLikelihood - lag(logLikelihood);
      run;
```

```
      proc sgplot data=&outds.;
         series x=date y=condLogLikelihood;
         yaxis label='Conditional Log Likelihood' min=-15 max=0;
      run;
   %end;
   %else %do;
      proc sgplot data=&outds.;
         series x=date y=logLikelihood;
         yaxis label='Log Likelihood';
      run;
   %end;
%mend plotEvaluation;

%plotEvaluation(mylib.&ds.inEval_k2to2_p0to0, &ds.inEval_k2to2_p0to0, 1);
```

Output 20.1.14 Conditional Log Likelihood



As shown in Output 20.1.14, some observations are explained well by the model, but there are still many exceptional observations whose conditional log likelihoods are much smaller than others. This means that there might be model misspecification, which can be addressed by solving the model selection problem.

Model Selection

To model the weekly returns, how many states should be considered? How many AR lags should a state include? These are common questions in the model selection process. In this example, the best model is selected using Akaike's information criterion (AIC): the smaller the AIC, the better the model.

The following macro estimates the model or models according to the arguments:

```
*************************************************************************;
* MODEL SELECTION                                                       ;
*************************************************************************;

%macro estimateRSAR(myds, inEstDs, kStart, kEnd, pStart, pEnd, method,
                    maxiter, qMultiStart);
   proc hmm data=&myds. labelSwitch=(sort=none)
      outstat=&myds.&method.Stat_k&kStart.To&kEnd._p&pStart.To&pEnd.;
      id time=date;
      model returnw / type=ar nstate=&kStart.:&kEnd. ylag=&pStart.:&pEnd.
         method=&method.;
      optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint
         maxiter=&maxiter. Multistart=&qMultiStart.;
      score outmodel=&myds&method.Model_k&kStart.To&kEnd._p&pStart.To&pEnd.;
      learn out=&myds.&method.Learn_k&kStart.To&kEnd._p&pStart.To&pEnd.
         %if %length(&inEstDs.)>0 %then %do; in=&inEstDs. %end;
         ;
      display / excludeall;
   run;
%mend;
```

First, the following statements estimate two- to nine-state RS-AR(0) models. The MAP method is applied to bound and smooth the objective functions. The parameter estimates are saved for further use as the initial values for new estimations. For the best results, multistart mode is strongly recommended, although it might be time consuming.

```
* (1) MAP + MULTISTART for AR(0);
* be aware that the following macro might take tens of hours to finish;
* uncomment it to run;
* %estimateRSAR(myds=mylib.&ds.In, inEstDs=,
   kStart=2, kEnd=9, pStart=0, pEnd=0,
   method=MAP, maxiter=128, qMultiStart=1);
```

Then, the following statements estimate two- to nine-state RS-AR(0) to AR(5) models. The ML method is used because the AIC is based on the log likelihood. The parameter estimates of RS AR(0) models from previous MAP method estimations are used to ensure that the estimates in the ML method are reasonable. The k-state RS AR(p) models, $k = 2, \ldots, 9$, $p = 1, \ldots, 5$, use the corresponding k-state RS AR($p-1$) models' parameter estimates as initial values. This is the default implementation in the HMM procedure. The multistart mode is not needed any longer because the initial values are good.

```
* (2) ML + initial values estimated from MAP's AR(0) for AR(0)
       and initial values estimated from AR(p-1) for AR(p), p>0;
* be aware that the following macro might take tens of minutes to finish;
* uncomment it to run;
* %estimateRSAR(myds=mylib.&ds.In, inEstDs=mylib.&ds.InMAPLearn_k2to9_p0to0,
   kStart=2, kEnd=9, pStart=0, pEnd=5,
```

```
   method=ML, maxiter=128, qMultiStart=0);
```

Estimating all 48 of the two- to nine-state RS AR(0) to AR(5) models produces AIC values. The following statements print the AIC values, which are shown in Output 20.1.15:

```
* print fit statistics in matrix form;
data &ds.InMLStat_k2to9_p0to5; set mylib.&ds.InMLStat_k2to9_p0to5; run;
proc sort data=&ds.InMLStat_k2to9_p0to5; by modelIndex; run;
%macro printFitStat(fitStat);
   data &ds.In&fitStat.;
      set &ds.InMLStat_k2to9_p0to5 end=eof;
      array &fitStat.s{8,6} _TEMPORARY_;
      array yLags{6} yLag0-yLag5;
      &fitStat.s[nState-1,yLag+1] = &fitStat.;
      if eof then do;
         do i = 1 to 8;
            nState = i+1;
            do j = 1 to 6; yLags(j) = &fitStat.s[i,j]; end;
            output;
         end;
      end;
      keep nState yLag0-yLag5;
   run;
   proc print data=&ds.In&fitStat. label noobs
      style(header)={textalign=center};
      var nState yLag0-yLag5;
      label nState='k' yLag0='p = 0' yLag1='p = 1' yLag2='p = 2'
         yLag3='p = 3' yLag4='p = 4' yLag5='p = 5';
   run;
%mend;
%printFitStat(AIC);

* Uncomment the following macros if you want to see other fit statistics;
* %printFitStat(logLikelihood);
* %printFitStat(AICC);
* %printFitStat(BIC);
* %printFitStat(HQC);
```

Output 20.1.15 indicates that the smallest AIC corresponds to the eight-state RS AR(4) model.

Output 20.1.15 AICs for 48 RS AR Models

Discovering the Hidden Market States

k	p = 0	p = 1	p = 2	p = 3	p = 4	p = 5
2	16987.55	16979.59	16981.64	16985.00	16988.80	16992.07
3	16820.74	16811.38	16816.45	16821.02	16825.88	16826.56
4	16772.85	16764.64	16754.79	16757.74	16763.77	16767.28
5	16728.05	16733.38	16738.14	16743.40	16750.67	16753.11
6	16697.18	16684.36	16684.90	16689.57	16695.59	16700.58
7	16701.64	16693.28	16682.39	16685.51	16690.77	16695.31
8	16685.88	16688.43	16687.35	16668.80	16650.11	16659.51
9	16711.81	16699.75	16698.66	16707.04	16679.98	16684.87

The Selected Model and In-Sample Inference

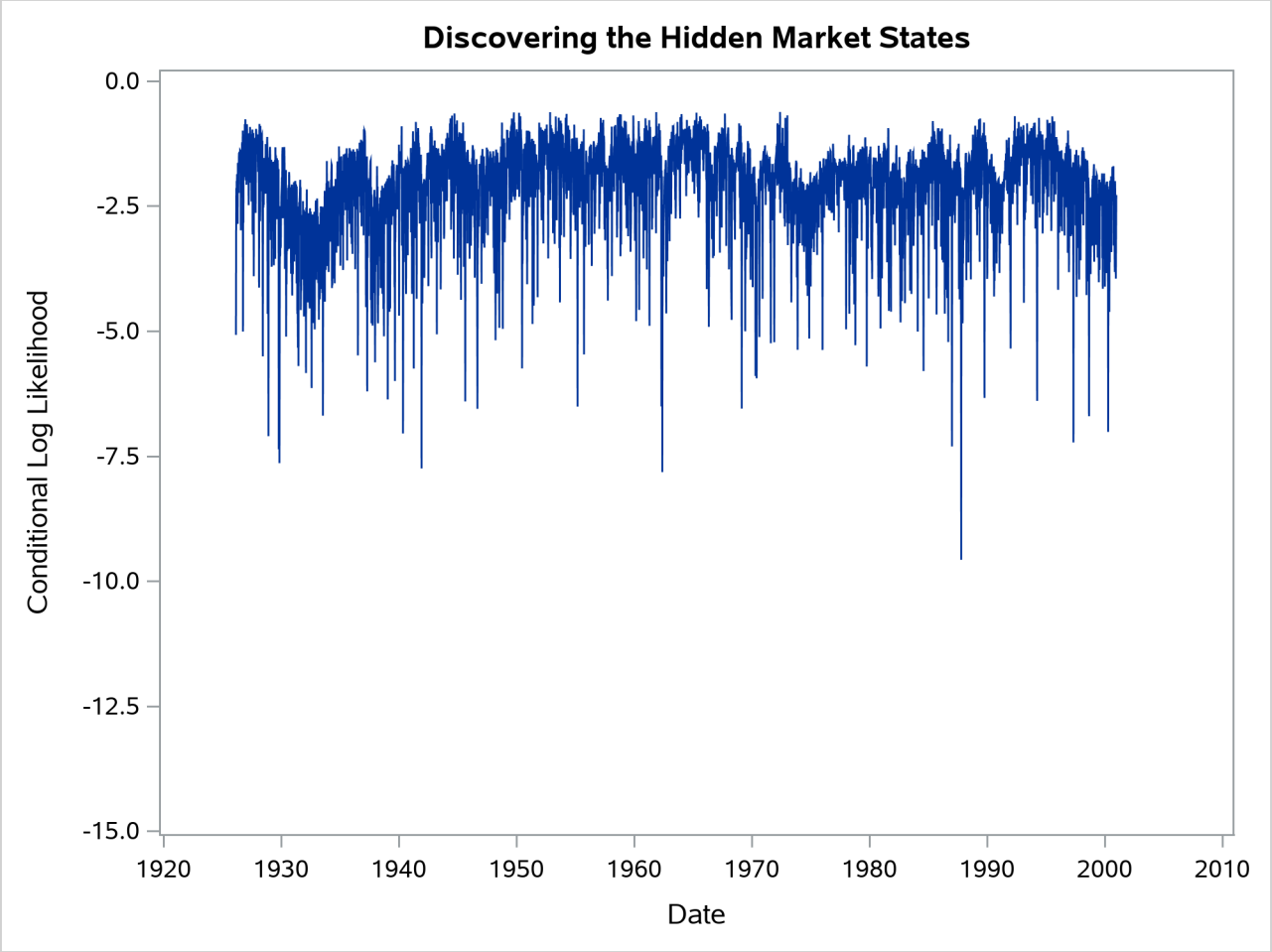
The following statements estimate the AIC-selected eight-state RS AR(4) model:

```
*************************************************************************;
* THE SELECTED MODEL AND IN-SAMPLE INFERENCE                            ;
*************************************************************************;

* select eight-state RS-AR(4) model according to AIC or AICC;
* adjust data to be the same sample size for AR(5);
proc hmm data=mylib.&ds.In(where=(date>='30JAN1926'd));
   id time=date;
   model returnw / type=ar nstate=8 ylag=4;
   optimize printLevel=3 algorithm=interiorpoint maxiter=0;
   score outmodel=mylib.&ds.InMLModel_k8To8_p4To4;
   learn out=mylib.&ds.InMLLearn_k8To8_p4To4
         in=mylib.&ds.InMLLearn_k2to9_p0to5;
   smooth out=mylib.&ds.inSmooth_k8to8_p4to4;
   decode out=mylib.&ds.inDecode_k8to8_p4to4;
   evaluate out=mylib.&ds.inEval_k8to8_p4to4;
   display tpm;
run;
```

The following statement plots the conditional log likelihoods. Compared to the bistate RS-AR(0) model (as shown in Output 20.1.14), the eight-state RS-AR(0) model (as shown in Output 20.1.16) has much higher conditional log likelihoods, which means a much better in-sample goodness of fit.

```
%plotEvaluation(mylib.&ds.inEval_k8to8_p4to4, &ds.inEval_k8to8_p4to4, 1);
```

Output 20.1.16 Conditional Log Likelihoods



The selected model has 112 parameters. 48 of which are related to observation distributions and are often called observation parameters. The following statements print the observation parameters in Output 20.1.17. The analysis of each AR(4) process for each state is beyond this example.

```
* print observation parameters;
data &ds.InObsParm;
   set mylib.&ds.InMLLearn_k8To8_p4To4(where=(type='EST' and index>=17));
   if(index=17) then parameter='Constant';
   if(index=19) then parameter='AR(1)';
   if(index=21) then parameter='AR(2)';
   if(index=23) then parameter='AR(3)';
   if(index=25) then parameter='AR(4)';
   if(index=27) then parameter='Variance';
run;
proc sort data=&ds.InObsParm; by index; run;
proc print data=&ds.InObsParm noobs label;
   var parameter state1-state8;
   label state1='State 1' state2='State 2' state3='State 3'
         state4='State 4' state5='State 5' state6='State 6'
         state7='State 7' state8='State 8';
run;
```

Output 20.1.17 Observation Parameters

Discovering the Hidden Market States

Parameter	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8
Constant	0.93305	1.15326	-0.59159	1.68887	-0.82124	-2.68701	-0.01897	-0.5056
AR(1)	-0.23663	0.03568	-0.06096	-0.17268	-0.04365	-0.34339	-0.05648	0.1213
AR(2)	0.00568	-0.07571	0.33570	-0.08520	-0.09594	-0.11391	0.06549	0.0259
AR(3)	-0.06493	-0.07301	0.01605	-0.06019	0.04540	-0.60512	0.11289	0.0025
AR(4)	-0.02869	-0.08348	-0.01595	-0.02281	0.12060	-0.29044	-0.11970	0.0613
Variance	0.23055	0.77141	1.21104	1.44655	1.54215	3.64774	9.42764	31.8773

The following statements calculate and print the sample mean and standard error for each state in Output 20.1.18 and then plot the corresponding Gaussian kernels in Output 20.1.19:

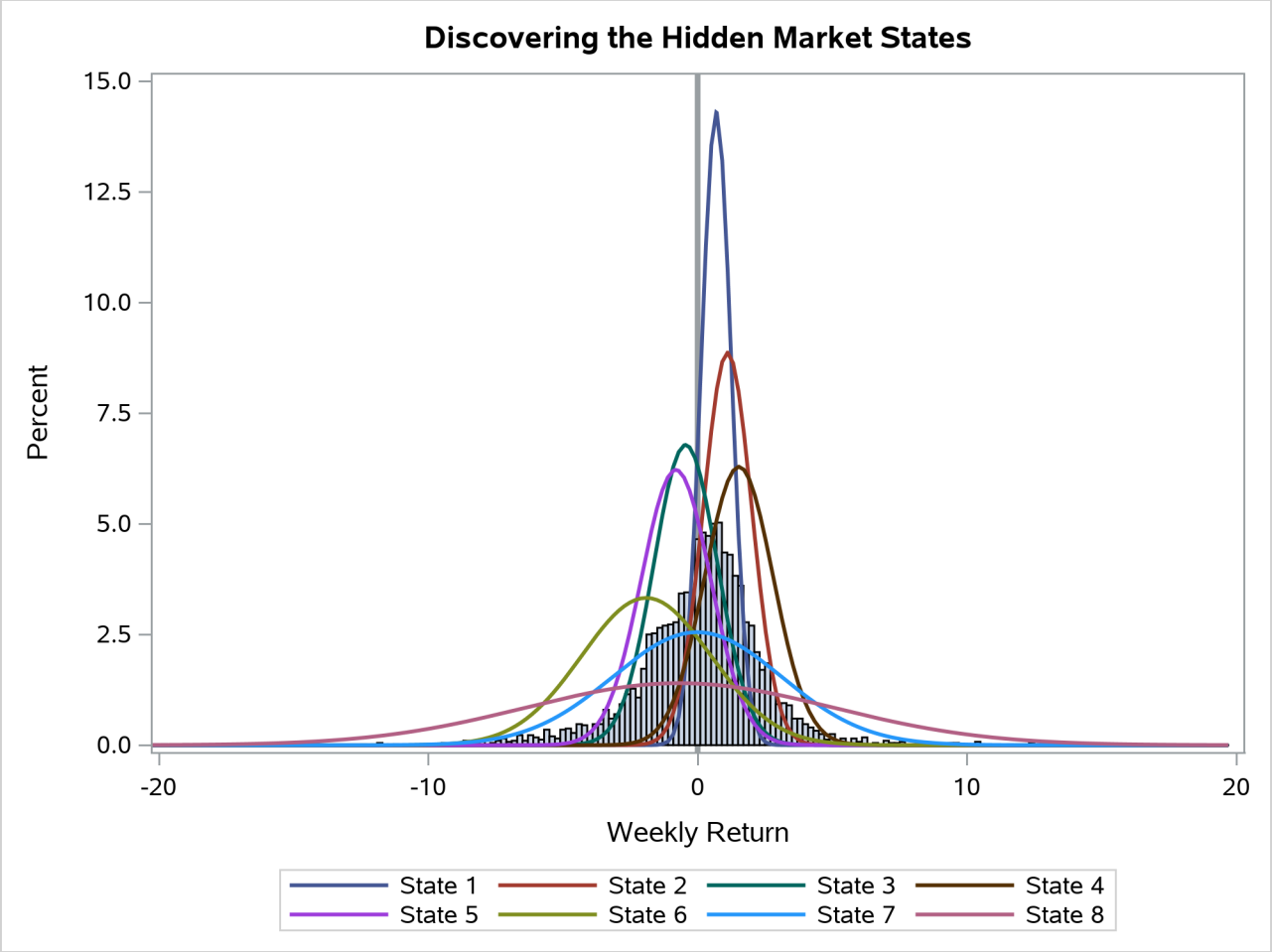
```
data &ds.inSmooth_k8to8_p4to4;
   set mylib.&ds.inSmooth_k8to8_p4to4(where=(date>='25FEB1926'd));
run;
proc sort data=&ds.inSmooth_k8to8_p4to4; by date; run;
data &ds.inTruncated; set &ds.in(where=(date>='25FEB1926'd)); run;
data &ds.inWeighted;
   merge &ds.inSmooth_k8to8_p4to4(in=a) &ds.inTruncated(in=b);
   by date;
   if(a=b);
run;
%printMeanAndStdErrOfAState(8);

%plotState(&ds.In,returnw,1,8);
```

Output 20.1.18 Sample Mean and Standard Error of Each State

Discovering the Hidden Market States

State	Sample Mean	Sample Standard Error
1	0.68995	0.55836
2	1.10751	0.90016
3	-0.44647	1.17678
4	1.53207	1.26888
5	-0.80688	1.28357
6	-1.91024	2.39908
7	-0.01947	3.12820
8	-0.67999	5.70094

Output 20.1.19 Gaussian Kernels for Eight States



As shown in Output 20.1.19, states 1, 2, and 4 seem like a bull market; they have positive expected returns with low risk. The remaining states seem like a bear market; they have negative expected returns or very high risk (or both). However, the market states and their corresponding states are not that simple. The transition probability matrix of the eight-state AR(4) model is shown in Output 20.1.20.

Output 20.1.20 Estimates of Transition Probability Matrix

Discovering the Hidden Market States

Estimated Transition Probability Matrix								
State	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0.00000	0.00000	0.98736	0.00000	0.00000	0.01264	0.00000	0.00000
2	0.57934	0.36501	0.00000	0.00000	0.02789	0.02776	0.00000	0.00000
3	0.00000	0.42100	0.57900	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.01976	0.00000	0.61521	0.36503	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.00000	0.33913	0.55466	0.10621	0.00000	0.00000
6	0.05948	0.00653	0.00000	0.17954	0.05488	0.59808	0.03352	0.06797
7	0.00000	0.00000	0.00000	0.02594	0.00000	0.00000	0.97186	0.00220
8	0.00000	0.00000	0.00000	0.03679	0.00000	0.00000	0.02078	0.94243

The eight states seem to be categorized into three groups:

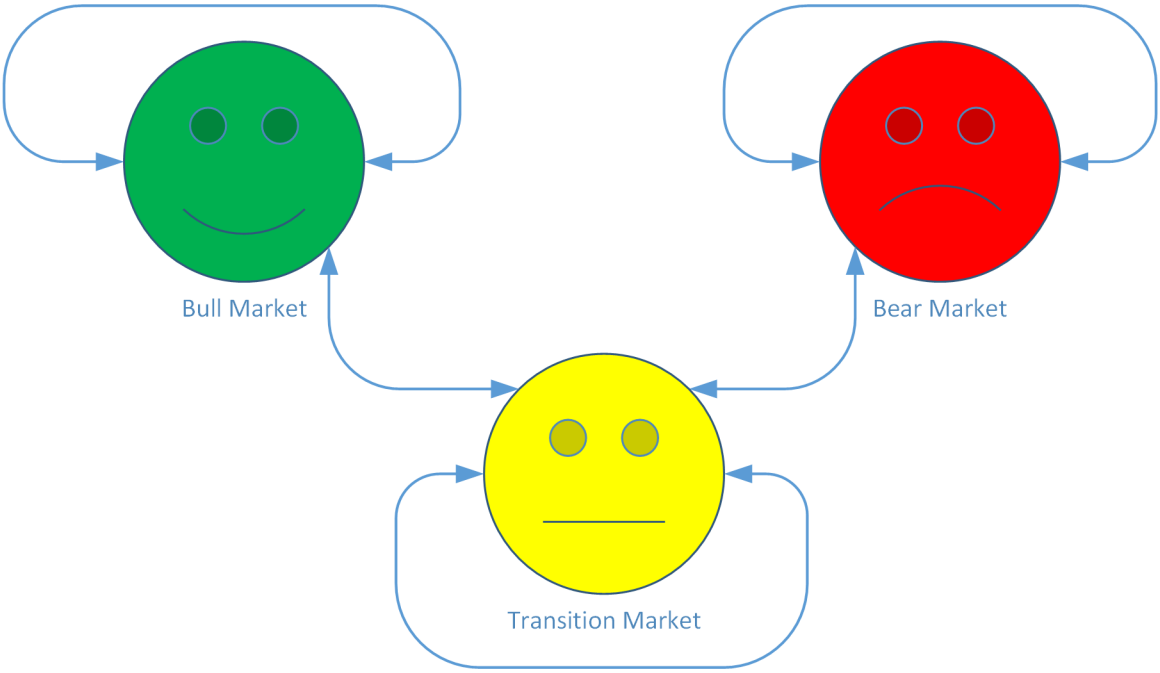
- States 1, 2, and 3 belong to first group. Although there are very upward-moving days (state 2) and somewhat downward-moving days (state 3), the main trend is upward (state 1). The risk in this group is pretty low.
- States 4, 5, and 6 belong to second group. There are good days (state 4), bad days (state 5), and even very bad days (state 6). This group has medium risk.
- States 7 and 8 belong to the third group, where the risk is extremely high.

The transition probability matrix tells the more detailed story:

- The first group has a 98% chance of switching to itself and a 2% chance of switching to the second group.
- The third group has a 97% chance of switching to itself and a 3% chance of switching to the second group.
- The second group has a 94% chance of switching to itself and equal 3% chances of switching to the first and third groups.
- The first and third group have zero chance of switching to each other.
- When the market is in one group, it lasts for a long time before switching to the other groups.

Hence, according to the eight-state RS-AR(4) model, there is one more market state besides the well-known bear market (the first group) and bull market (the third group): the transition market state (the second group). Figure 20.107 shows the relationship between these market states.

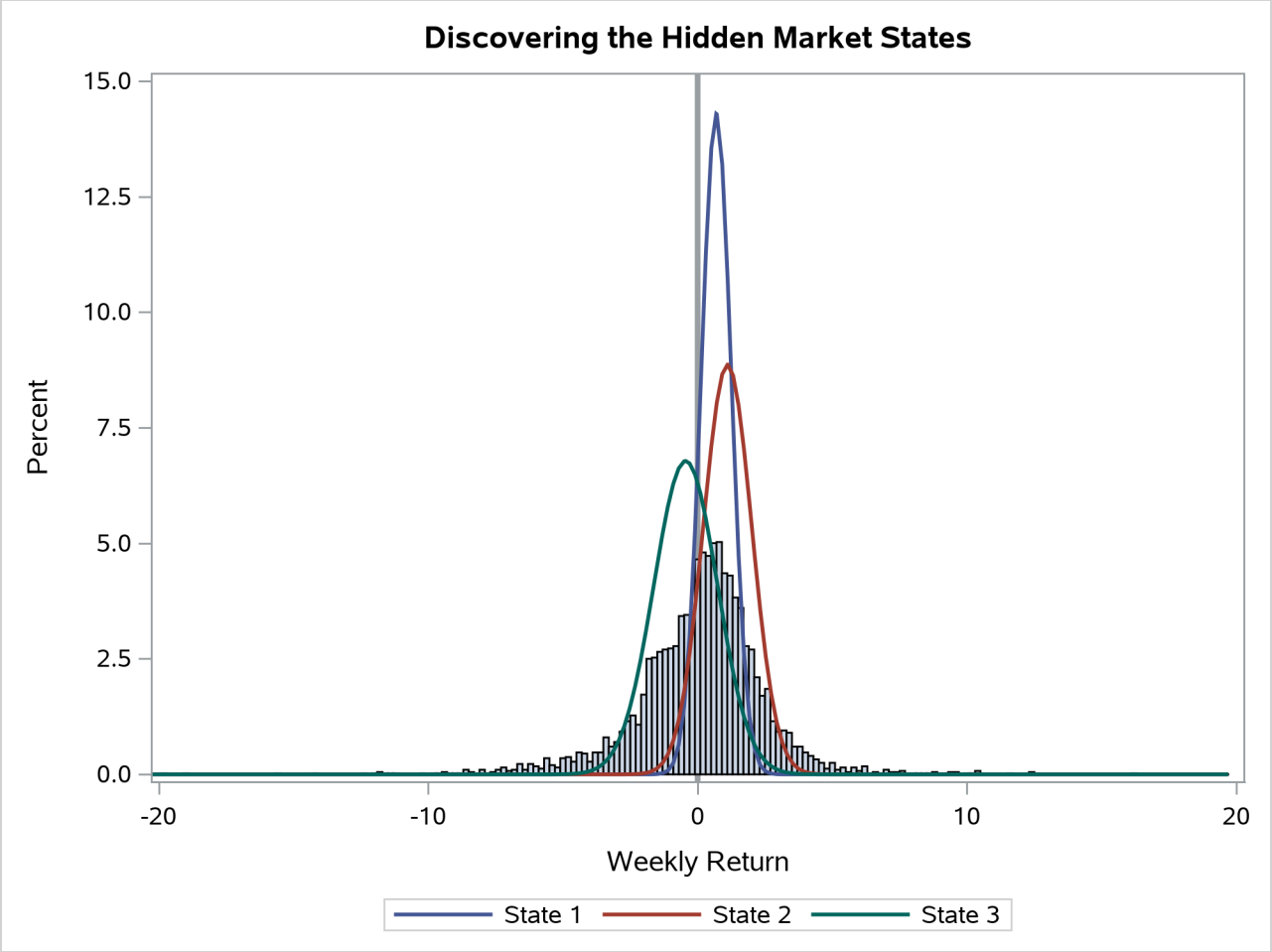
Figure 20.107 Market States and Their Transitions



The following statement plots three states that correspond to the bull market state, which are shown in Output 20.1.21:

```
%plotState(&ds.In,returnw,1,3);
```

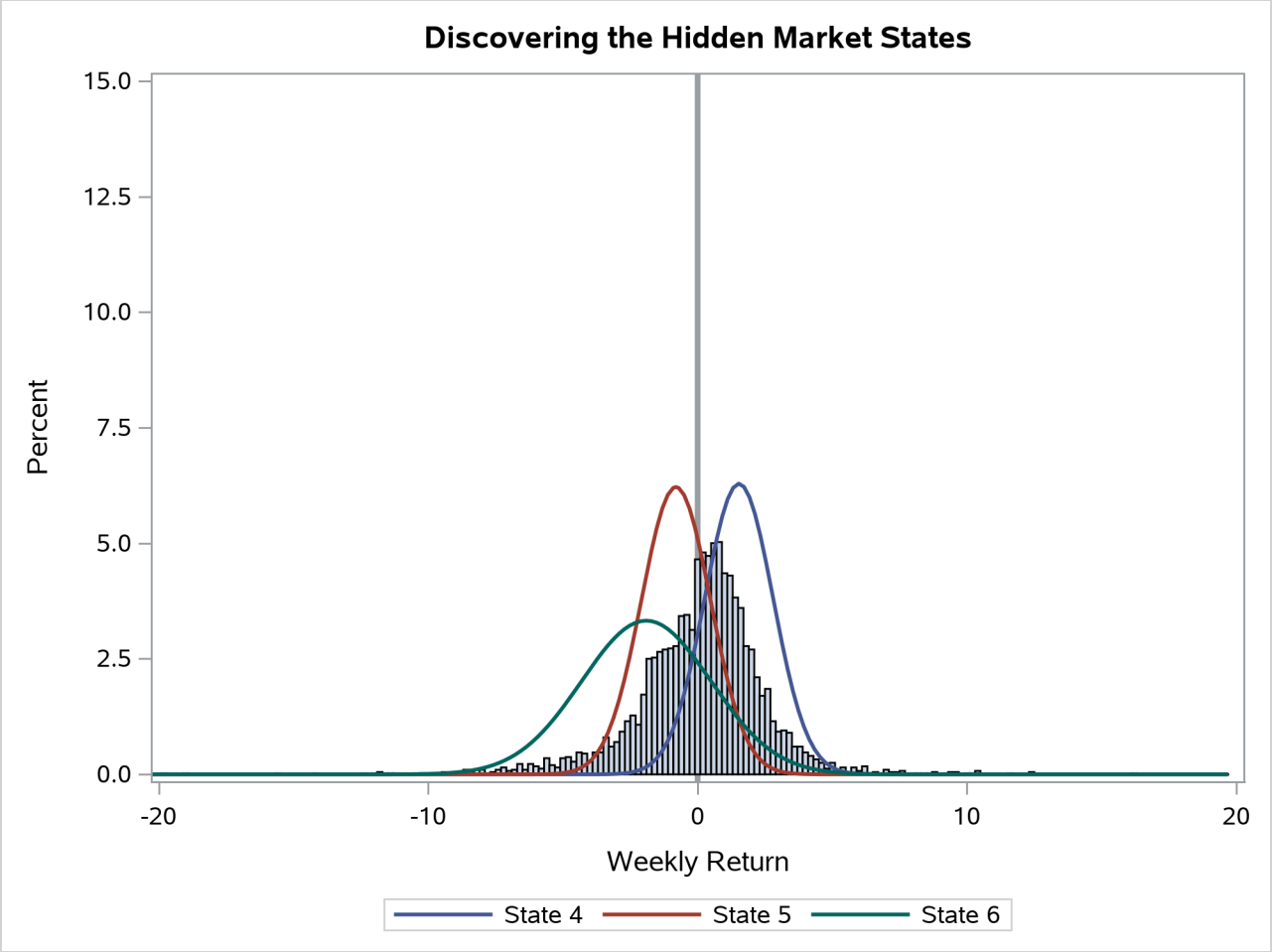
Output 20.1.21 Three Gaussian Kernels for the Bull Market State



The following statement plots three states that correspond to the transition market state, which are shown in Output 20.1.22:

```
%plotState(&ds.In,returnw,4,6);
```

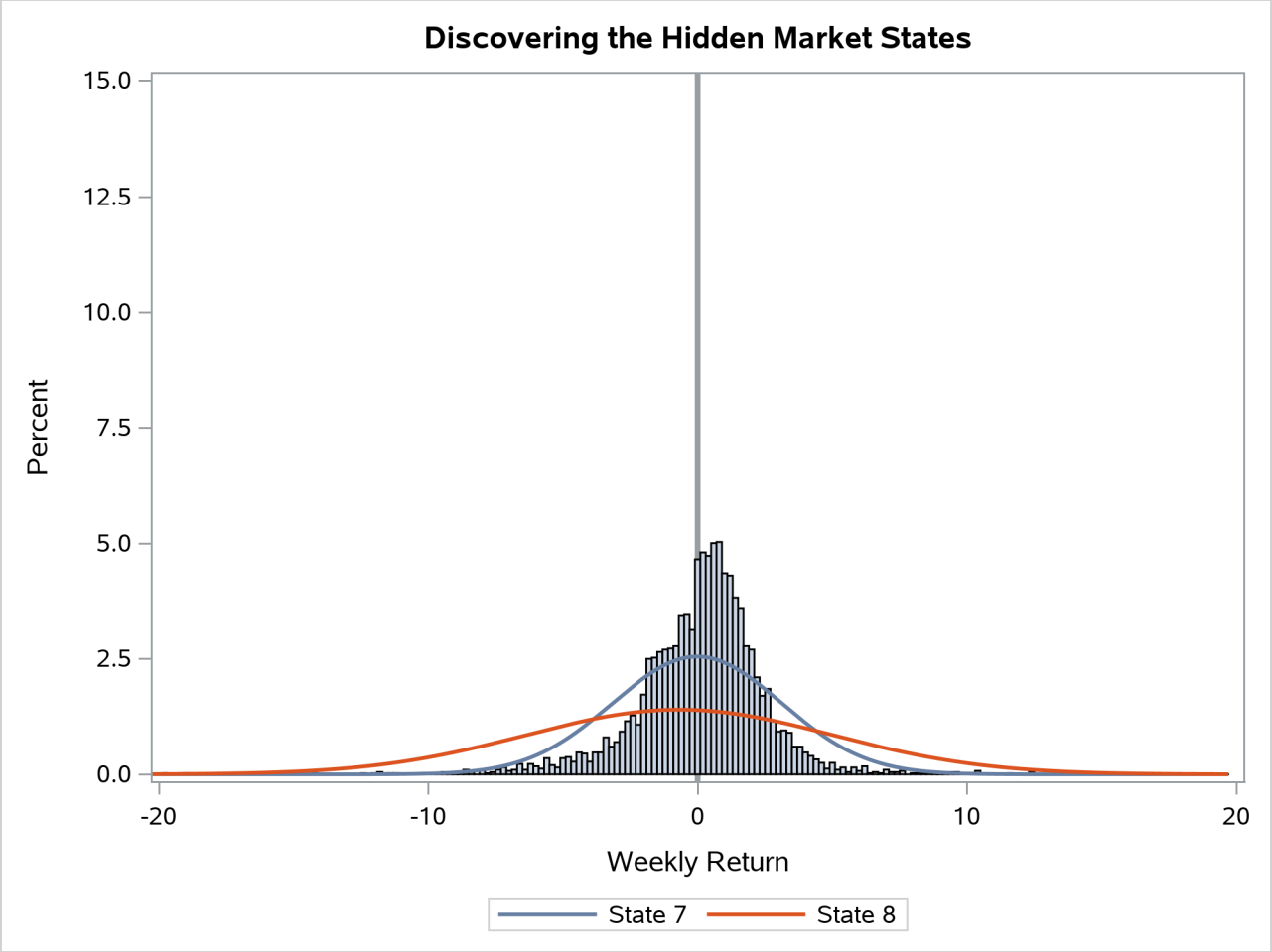
Output 20.1.22 Three Gaussian Kernels for the Transition Market State



The following statement plots two states that correspond to the bear market state, which are shown in Output 20.1.23:

```
%plotState(&ds.In,returnw,7,8);
```

Output 20.1.23 Two Gaussian Kernels for the Bear Market State

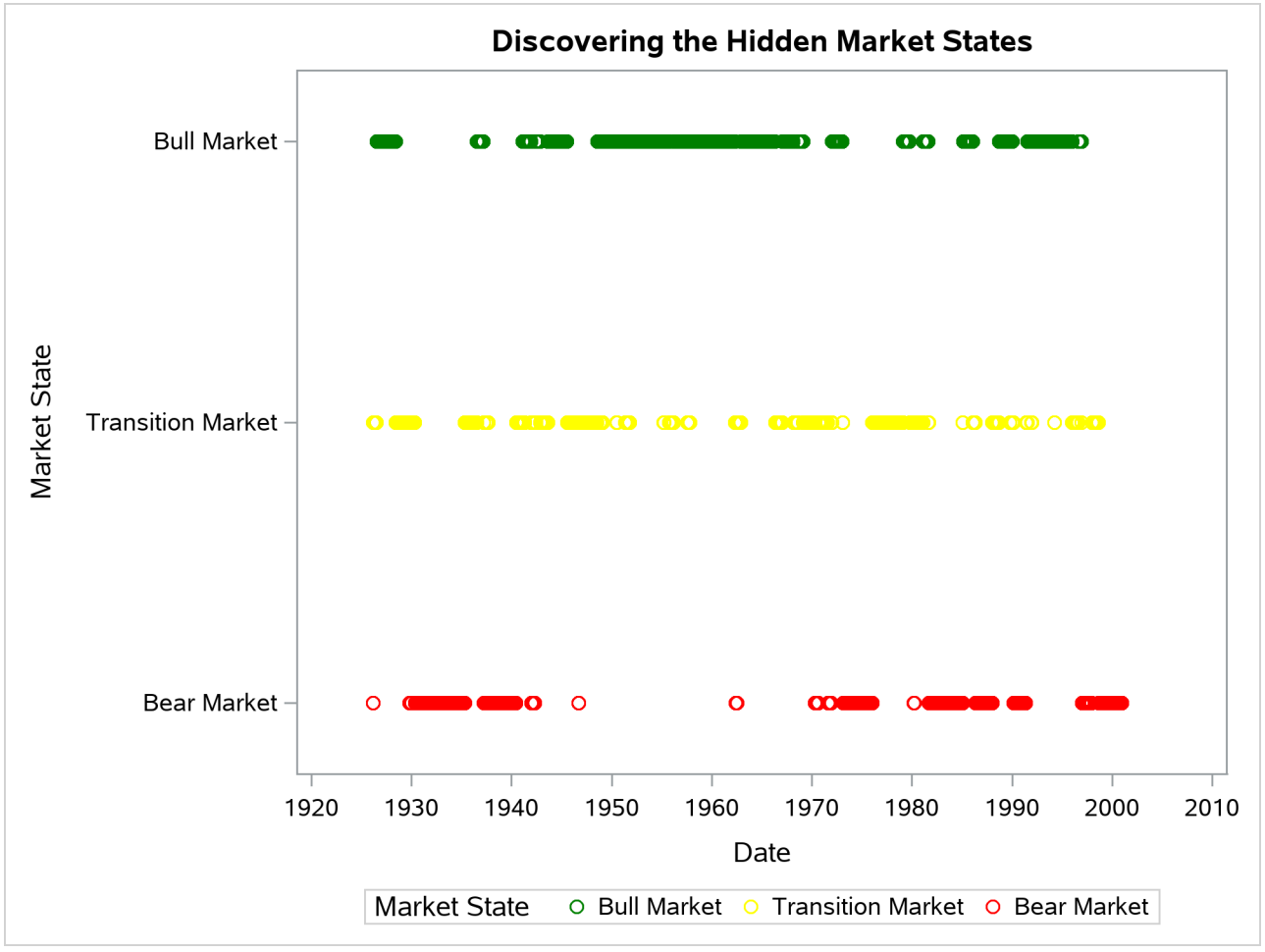


The following statements plot the decoded path of states according to the eight-state RS-AR(4) model; the result is shown in Output 20.1.24.

```
* plot decoded results;
data &ds.InDecode_k8to8_p4to4; set mylib.&ds.InDecode_k8to8_p4to4; run;
proc sort data=&ds.InDecode_k8to8_p4to4; by date; run;
data decodeMaketStates;
   set &ds.In(where=(date>='30JAN1926'd));
   set &ds.InDecode_k8to8_p4to4;
   array returnws returnw1-returnw3;
   array prices price1-price3;
   if(state~=.) then do;
      if(state=1 | state=2 | state=3) then do;
         returnws[1] = returnw; prices[1] = price;
         marketState="Bull Market      ";
      end;
      if(state=4 | state=5 | state=6) then do;
         returnws[2] = returnw; prices[2] = price;
         marketState="Transition Market";
      end;
```

```
      if(state=7 | state=8) then do;
         returnws[3] = returnw; prices[3] = price;
         marketState="Bear Market      ";
      end;
   end;
   label marketState="Market State";
run;
proc sgplot data=decodeMaketStates;
   styleattrs datacontrastcolors=(red yellow green);
   scatter x=date y=marketState / group=marketState nomissinggroup;
   yaxis label="Market State" offsetmin=0.1 offsetmax=0.1;
   keylegend / sortorder=reverseauto;
run;
```

Output 20.1.24 Decoded Market States

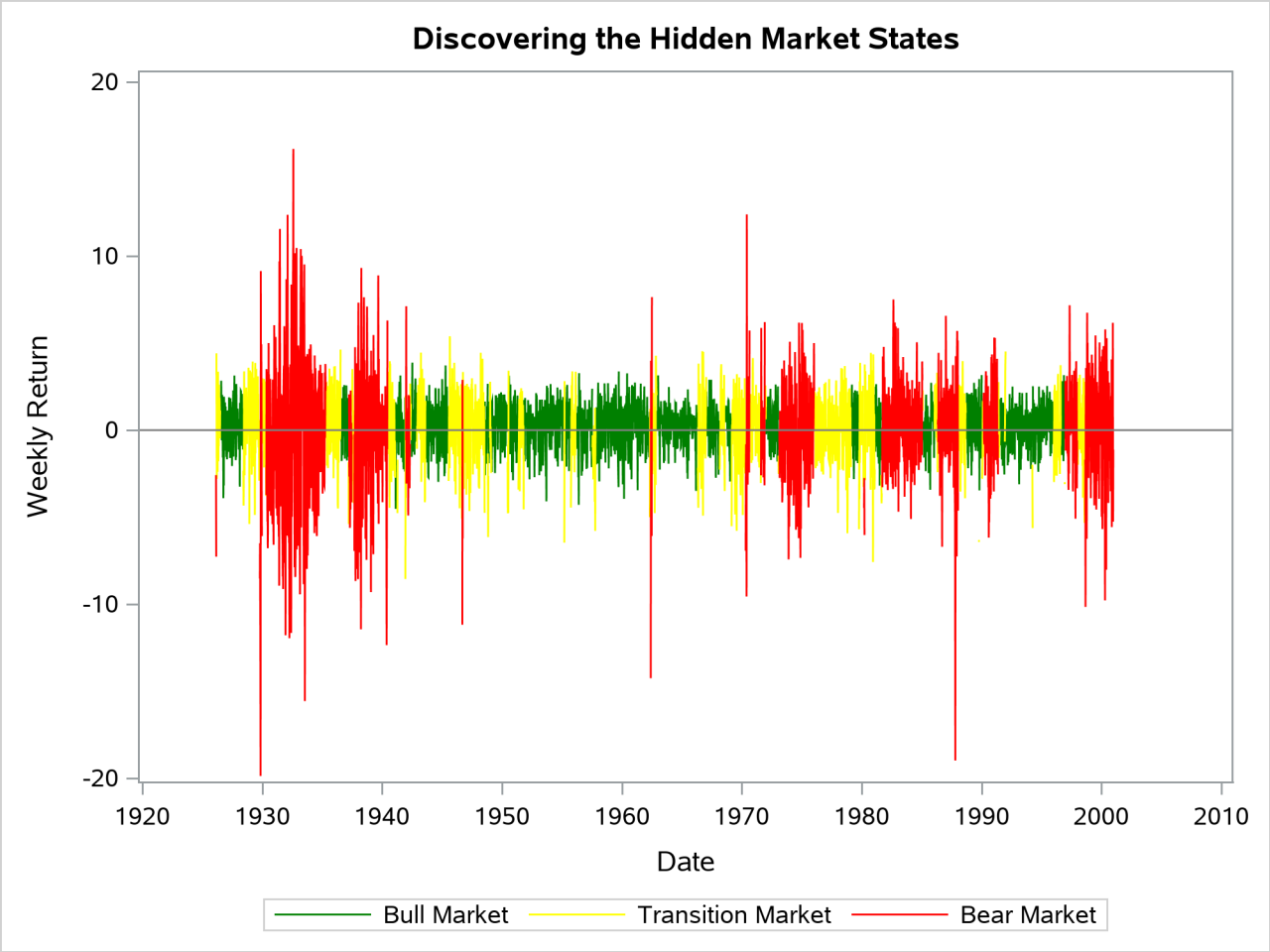


The following statements plot the decoded weekly returns, which are shown in Output 20.1.25:

```
proc sgplot data=decodeMaketStates;
   series x=date y=returnw1 / break lineattrs=(color=green)
                              legendlabel="Bull Market";
   series x=date y=returnw2 / break lineattrs=(color=yellow)
                              legendlabel="Transition Market";
```

```
   series x=date y=returnw3 / break lineattrs=(color=red)
                              legendlabel="Bear Market";
   refline 0 / axis=y lineattrs=(color=gray thickness=1);
   yaxis max=20 min=-20 label="Weekly Return";
run;
```

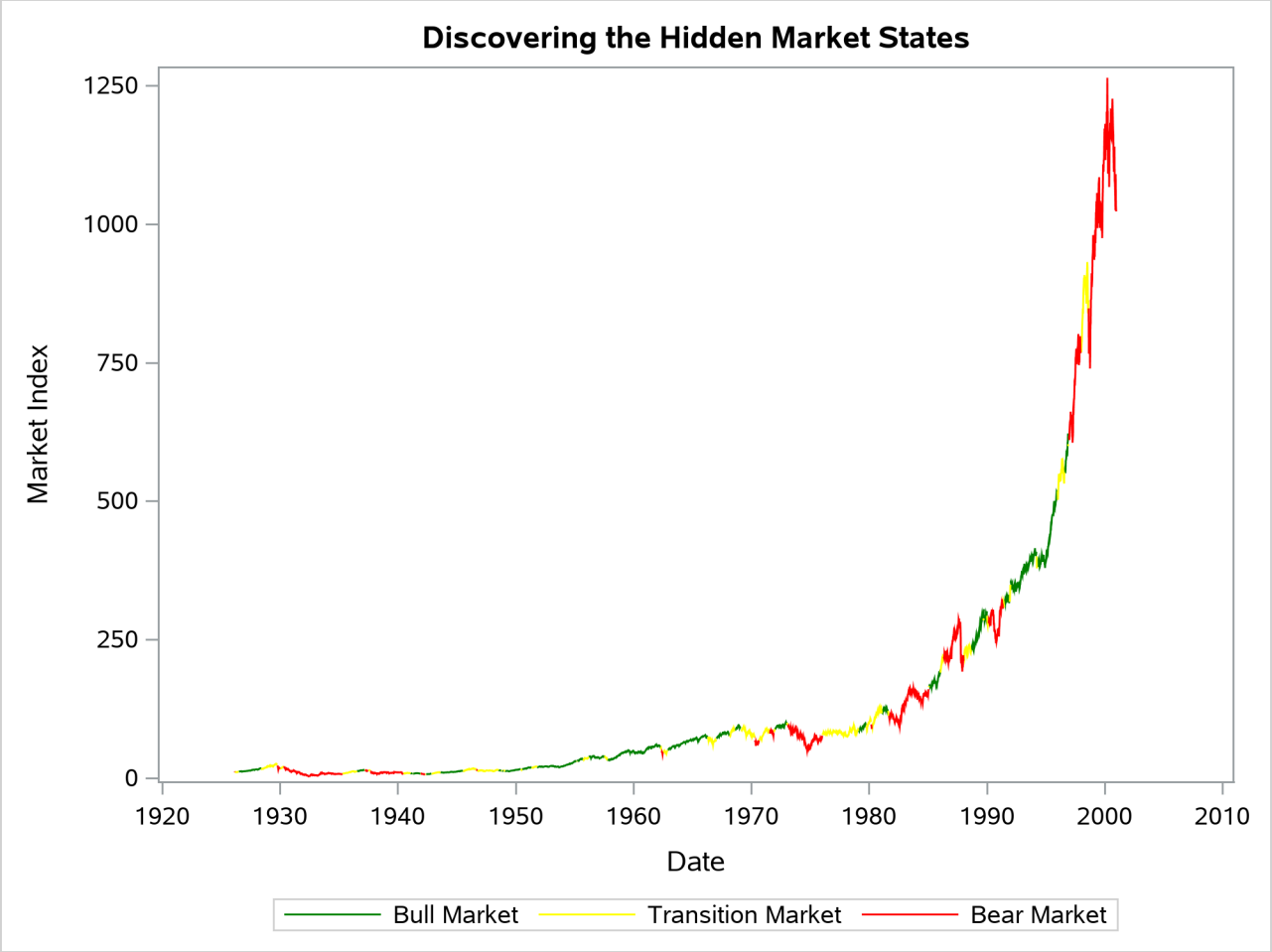
Output 20.1.25 Decoded Weekly Returns



The following statements plot the decoded stock prices, which are shown in Output 20.1.26:

```
proc sgplot data=decodeMaketStates;
   series x=date y=price1 / break lineattrs=(color=green)
                            legendlabel="Bull Market";
   series x=date y=price2 / break lineattrs=(color=yellow)
                            legendlabel="Transition Market";
   series x=date y=price3 / break lineattrs=(color=red)
                            legendlabel="Bear Market";
   yaxis label="Market Index";
run;
```

Output 20.1.26 Decoded Stock Prices



The Out-of-Sample Testing

In the model selection process, the AIC, but not the in-sample goodness of fit (that is, the log likelihood), is used as the criterion for selecting the best model because a bias-variance trade-off exists. In this section, three types of out-of-sample forecasts are tested to check whether the selected model has a good forecast performance.

The Value-at-Risk (VaR) Forecast

You can specify the INMODEL= option in the SCORE statement to forecast the VaRs at different levels, such as 1%, 5%, and 10%, as shown in Output 20.1.27. The following macros can predict the VaRs for the out-of-sample period and evaluate the predictive performance by the likelihood ratio (LR) test for the unconditional coverage of the VaR forecast (Kuester, Mittnik, and Paolella 2006):

```
* print Predictive Performance of VaR Forecasts;
%let alpha1 = 0.80;
%let alpha2 = 0.90;
%let alpha3 = 0.98;
%macro forecastVaR(myLib, dsData, dsModel, timeId, varReturn,
   k, p, iStart, iEnd, oosStart, oosEnd,
```

```
   dsForecastPrefix);
   %do i = &iStart. %to &iEnd.;
      proc hmm data=&myLib..&dsData.;
         score inmodel=&myLib..&dsModel.;
         forecast out=&myLib..&dsForecastPrefix.&i. alpha=&&alpha&i. online;
      run;
      data &dsForecastPrefix.&i.;
         set &myLib..&dsForecastPrefix.&i.;
      run;
      proc sort data=&dsForecastPrefix.&i.; by &timeId.; run;
      data &dsForecastPrefix.&i.;
         set &dsForecastPrefix.&i. (FIRSTOBS=&oosStart. OBS=&oosEnd.
            keep=&timeId. &varReturn._Q1
            rename=(&varReturn._Q1=&varReturn._Q1_&k.&p.&i.));
         time=_N_; nState = &k; yLag = &p;
      run;
   %end;
%mend forecastVaR;

%macro evaluateVaR(dsData,dsForecastPrefix,varReturn,k,p,iStart,iEnd,n,dsOut);
   data &dsData.out;
      set &dsData.;
      if (date>&cutDate.) then output;
   run;
   data forecastData;
      set &dsData.out;
      time=_N_;
   run;
   data forecastData;
      merge &dsForecastPrefix.: forecastData;
      by time;
   run;
   data &dsOut.;
      set forecastData;
      retain %do i = &iStart. %to &iEnd.; cviol&k.&p.&i. 0 %end; ;
      retain %do i = &iStart. %to &iEnd.; c&varReturn.&k.&p.&i. 0 %end; ;
      %do i = &iStart. %to &iEnd.;
         if &varReturn. le &varReturn._Q1_&k.&p.&i. then do;
            cviol&k.&p.&i.=cviol&k.&p.&i.+1;
         end;
         c&varReturn.&k.&p.&i. = c&varReturn.&k.&p.&i.
                                 + &varReturn._Q1_&k.&p.&i.;
      %end;
      if _N_ = &n. then do;
         %do i = &iStart. %to &iEnd.;
            Norminal = (1-&&alpha&i.)/2;
            Viol = cviol&k.&p.&i. / &n.;
            LR = 2*(cviol&k.&p.&i.*log(Viol)+(&n.-cviol&k.&p.&i.)*log(1-Viol)
               -(cviol&k.&p.&i.*log(Norminal)+(&n.-cviol&k.&p.&i.)
                *log(1-Norminal)));
            pValue = 1 - cdf("CHISQUARE", LR, 1);
            meanVaR = c&varReturn.&k.&p.&i. / &n.;
            output;
         %end;
```

```
      end;
      label Norminal='Target Prob.' Viol='Violation Ratio' LR='LR Stat.'
            pValue='Pr > ChiSq' meanVaR='Avg. of VaR' nState='Number of States'
            yLag='AR Lag';
      keep nState yLag Norminal Viol LR pValue meanVaR;
   run;
   proc print data=&dsOut. noobs label;  format Viol LR pValue meanVaR 6.4; run;
%mend evaluateVaR;

%macro VaR(k,p);
   %forecastVaR(myLib=mylib, dsData=&ds.,
      dsModel=&ds.inMLModel_k&k.To&k._p&p.To&p.,
      timeId=date, varReturn=returnw,
      k=&k., p=&p., iStart=1, iEnd=3,
      oosStart=%eval(3999-(5-&p.)), oosEnd=%eval(4854-1-(5-&p.)),
      dsForecastPrefix=&ds.Forecastk&k._p&p.);
   %evaluateVaR(dsData=&ds.,dsForecastPrefix=&ds.Forecastk&k._p&p.,
      varReturn=returnw,k=&k.,p=&p.,iStart=1,iEnd=3,n=855,
      dsOut=VaR_outputk&k._p&p.);
%mend VaR;

%VaR(8,4);
```

Output 20.1.27 Predictive Performance of VaR Forecasts

Discovering the Hidden Market States

Number of States	AR Lag	Target Prob.	Violation Ratio	LR Stat.	Pr > ChiSq	Avg. of VaR
8	4	0.10	0.1053	0.2592	0.6107	-2.718
8	4	0.05	0.0538	0.2541	0.6142	-3.691
8	4	0.01	0.0129	0.6503	0.4200	-5.974

According to the *p*-values in the Pr > ChiSq column in Output 20.1.27, at the 5% significance level, no tests can reject the null hypothesis that the number of violations is correct. Hence, the eight-state RS-AR(4) model has the correct unconditional coverage for the 1%, 5%, and 10% VaR forecasts.

The Out-of-Sample Log Likelihood

In fact, besides the good predictability of the tail of the distribution of weekly returns as shown in the VaR forecast analysis, the eight-state RS-AR(4) model also provides a very good prediction of the whole distribution of weekly returns, which can be seen by comparing the average weekly log likelihoods of the in-sample period and the out-of-sample period. As shown in Output 20.1.28, compared to both the simplest model (the two-state RS-AR(0) model, which has the fewest parameters and the worst in-sample fit) and the most complex model (the nine-state RS-AR(5) model, which has the most parameters and the best in-sample fit), the eight-state RS-AR(4) model has the best out-of-sample forecast ability (that is, the largest average weekly log likelihood in the out-of-sample period).

```
* print in-sample and out-of-sample average weekly log likelihood;
proc hmm data=mylib.&ds.
         outstat=mylib.&ds.Stat_k2to9_p0to5;
   score inmodel=mylib.&ds.inmlmodel_k2to9_p0to5;
```

```
   evaluate out=mylib.&ds.Eval_k2to9_p0to5;
run;
data inLL;
   set mylib.&ds.Eval_k2to9_p0to5(where=(date='26DEC2000'd)
      rename=(logLikelihood=isLL));
run;
data fullLL;
   set mylib.&ds.Eval_k2to9_p0to5(where=(date='22DEC2017'd)
      rename=(logLikelihood=fullLL));
run;
data oosLL;
   merge inLL(in=a) fullLL(in=b);
   by modelIndex;
   oosLL = (fullLL - isLL)/855;
   if(a=b);
   drop date isLL fullLL;
run;

data inLL;
   set mylib.&ds.InMLStat_k2to9_p0to5;
   isLL = logLikelihood / (3999-5);
   keep modelIndex nState yLag isLL;
run;
proc sort data=inLL; by modelIndex; run;

data avgLL;
   merge inLL(in=a) oosLL(in=b);
   by modelIndex;
   if(a=b and (modelIndex=1 or modelIndex=41 or modelIndex=48));
   drop modelIndex;
run;

proc print data=avgLL noobs label;
   label nState='k' yLag='p' isLL='In-Sample' oosLL='Out-Of-Sample';
run;
```

Output 20.1.28 Comparison of Average Weekly Log Likelihoods

Discovering the Hidden Market States

k	p	In-Sample	Out-Of-Sample
2	0	-2.12513	-2.14688
8	4	-2.05835	-2.10316
9	5	-2.05494	-2.12056

Trading Strategies and Portfolio Values

As mentioned at the beginning of this example, if you knew it would be a bull market, you might choose to buy the stock; if you knew it would be a bear market, you might choose to sell the stock. In this section, the market states are forecasted and then several trading strategies are compared.

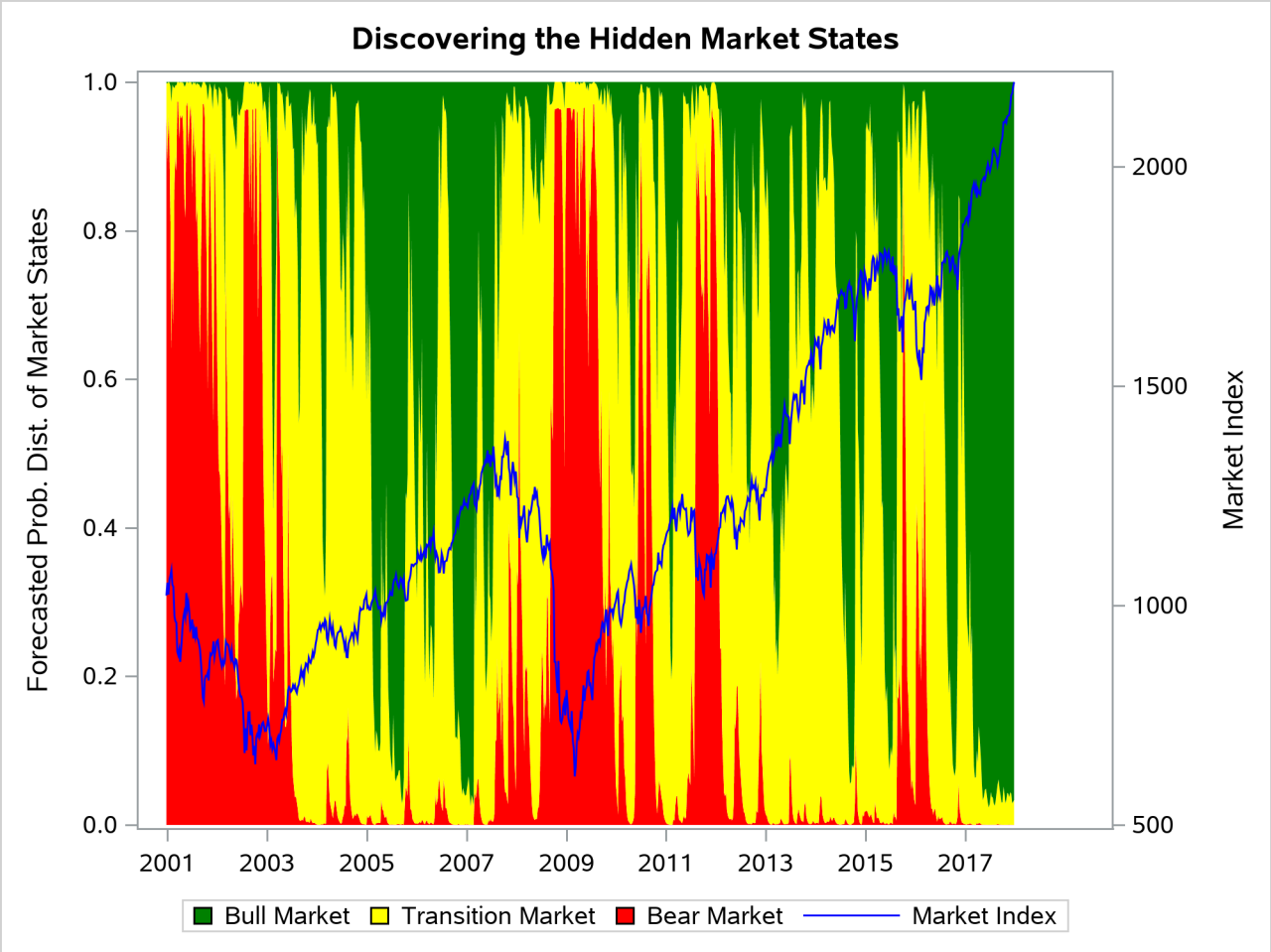
The following statements forecast and plot the probability distribution of market states in Output 20.1.29. The forecast of market states is in probabilities, which is similar to a weather forecast. For example, a forecast of

60% chance of bear market, 39% chance of transition market, and 1% chance of bull market is comparable to a forecast of 60% chance of raining, 39% chance of being cloudy, and 1% chance of being sunny.

```
proc hmm data=mylib.&ds.;
   score inmodel=mylib.&ds.InMLModel_k8To8_p4To4;
   forecast out=mylib.&ds.Forecast_k8to8_p4to4 online;
   display / excludeall;
run;

%let forecastStartDate = '26DEC2000'd;
%let forecastEndDate = '22DEC2017'd;
data ForecastProb;
   set mylib.&ds.Forecast_k8to8_p4to4
          (where=(date>=&forecastStartDate. and date<=&forecastEndDate.));
run;
proc sort data=ForecastProb; by date; run;
data ForecastProb;
   set ForecastProb(keep=state1 - state8) end=last;
   set &ds.(where=(date>=&forecastStartDate. and date<=&forecastEndDate.)
            keep=date price);
   * probability distribution of market states;
   bullProb = state1+state2+state3;
   transitionProb = state4+state5+state6;
   bearProb = state7+state8;
   bearTransitionProb = bearProb + transitionProb;
run;
* plot forecasted prob. dist. of market states and trading strategies;
%macro plotForecastAndTradingStrategies(ds, bearThreshold, bullThreshold,
                                        oneMinusBullThreshold);
   proc sgplot data=&ds.;
      band x=date lower=0 upper=bearProb /
         fillattrs=(color=red) name="bear" legendlabel="Bear Market";
      band x=date lower=bearProb upper=bearTransitionProb /
         fillattrs=(color=yellow)
         name="transition" legendlabel="Transition Market";
      band x=date lower=bearTransitionProb upper=1 /
         fillattrs=(color=green) name="bull" legendlabel="Bull Market";
      series x=date y=price / y2axis lineattrs=(color=blue)
         name="index" legendlabel="Market Index";
      %if %length(&bearThreshold.)>0 %then %do;
         refline &bearThreshold. / axis=y lineattrs=(color=red thickness=3)
            label="Bear Threshold=&bearThreshold." labelloc=inside
            labelattrs=(weight=bold);
      %end;
      %if %length(&bearThreshold.)>0 %then %do;
         refline &oneMinusBullThreshold. / axis=y
            lineattrs=(color=green thickness=3)
            label="Bull Threshold=&bullThreshold." labelloc=inside
            labelattrs=(weight=bold);
      %end;
      xaxis values=("01JAN2001"d to "31DEC2017"d by year) display=(nolabel)
         tickvalueformat=year4. offsetmin=0.03 offsetmax=0.15;
      yaxis label="Forecasted Prob. Dist. of Market States";
      y2axis label="Market Index";
```

```
      keylegend "bull" "transition" "bear" "index"/
         location=outside position=bottom;
   run;
%mend;

%plotForecastAndTradingStrategies(ForecastProb);
```

Output 20.1.29 Forecasted Probabilities of Market States



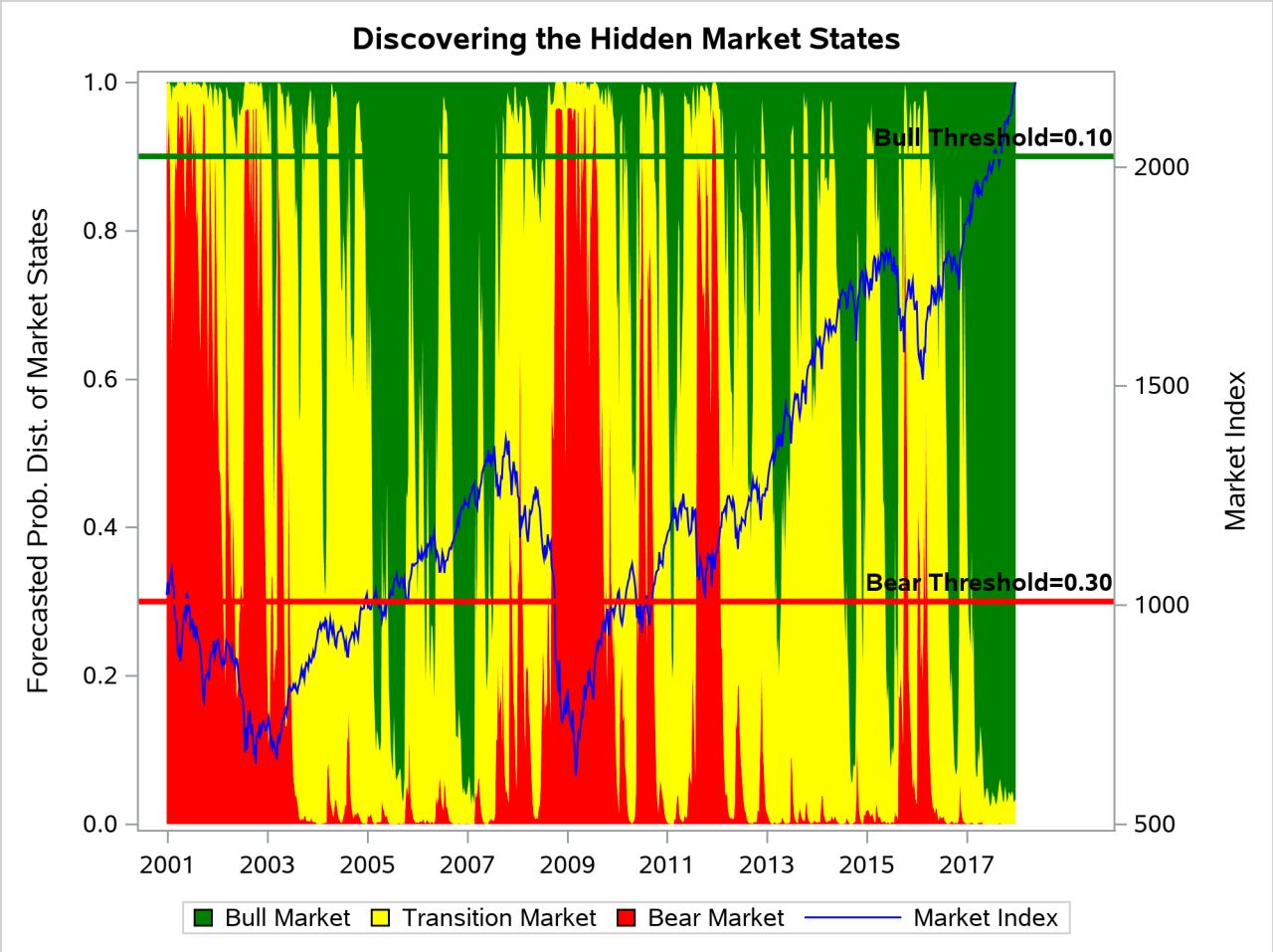
In this example, four simple trading strategies are considered:

- Risk-free strategy: hold the bond for the entire time period. This strategy does not depend on any forecast, and it is the first benchmark strategy.
- Market strategy: hold the stock for the entire time period. This strategy does not depend on any forecast, either. It reflects the market performance, and it is the second benchmark strategy.
- Bear strategy: sell the stock if the chance of a bear market in the next week is no less than 30%; otherwise, buy the stock.
- Bull strategy: buy the stock if the chance of a bull market in the next week is no less than 10%; otherwise, sell the stock.

The thresholds in bear and bull strategies, 30% and 10% respectively, are ad hoc, which reflect the investor's risk attitude. The trading strategy of switching between bond and stock can also be analogous to the decision of traveling with or without umbrella. The risk-free strategy is an analogous to a decision to always travel with an umbrella; the market strategy is to analogous to never traveling with umbrella; the bear strategy is analogous to traveling with an umbrella if the chance of raining tomorrow is no less than 30%; and the bull strategy is analogous to traveling without an umbrella if the chance of being sunny tomorrow is no less than 10%. The following statements plot the thresholds for the bear and bull trading strategies:

```
%let bearThreshold=0.30;
%let bullThreshold=0.10;
%let oneMinusBullThreshold=0.90;
%plotForecastAndTradingStrategies(ForecastProb, &bearThreshold.,
                                  &bullThreshold., &oneMinusBullThreshold.);
```

Output 20.1.30 Thresholds for Bull and Bear Trading Strategies



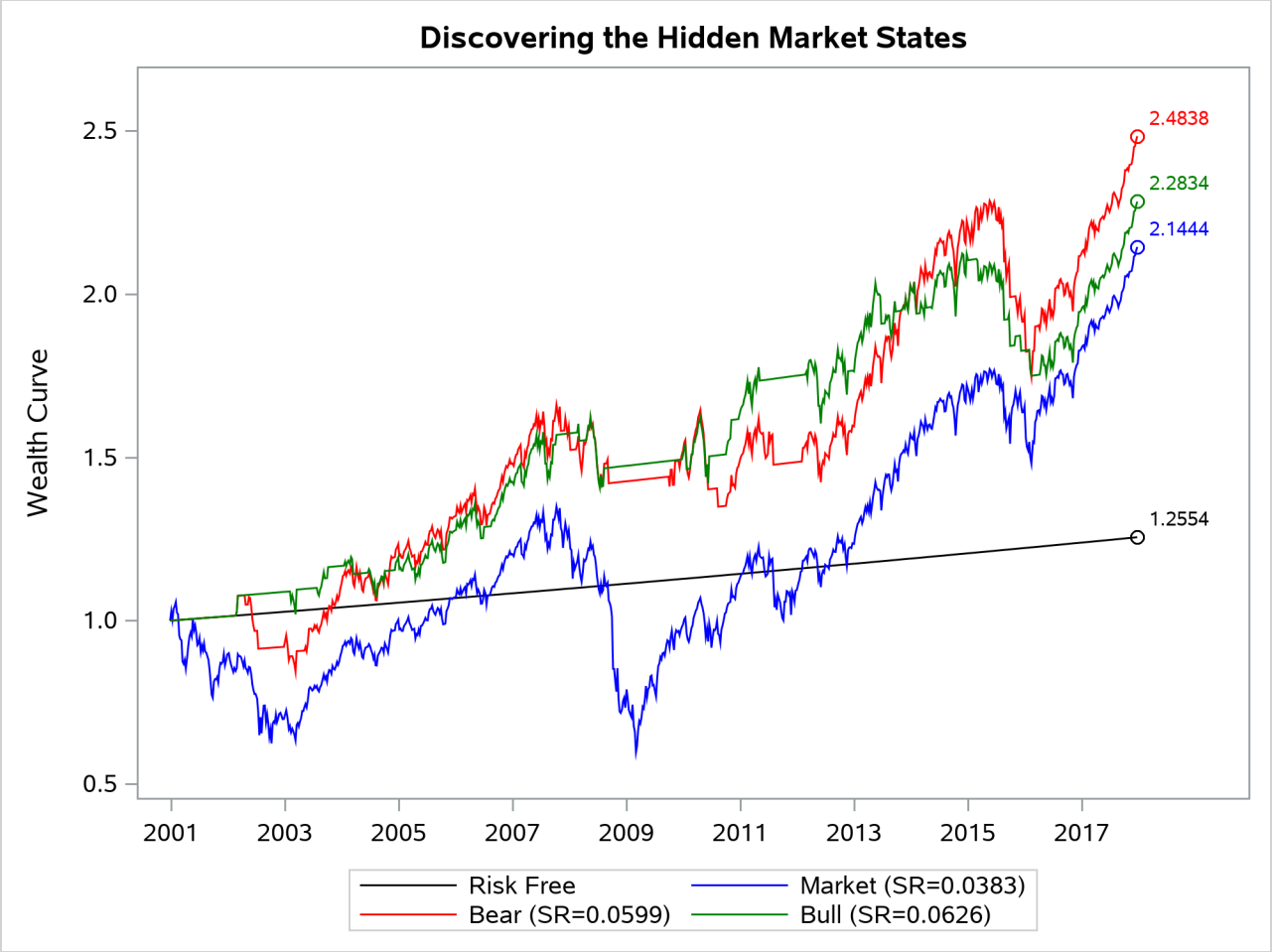
The following statements calculate the portfolio values and plot the wealth curves of four trading strategies:

```
%let strategy1=RiskFree;
%let strategy2=Market;
%let strategy3=Bear;
%let strategy4=Bull;
```

```
%let strategy1Label=Risk Free;
%let strategy2Label=Market;
%let strategy3Label=Bear;
%let strategy4Label=Bull;
* assume that 1 year = 252 trading days = 50.4 weeks;
* weekly risk free rate = exp(ln(1+APY)/50.4)-1;
%let riskFreeRate = 0.0002661; * APY=1.35%;
%let initialMoney=1;

%macro port_val_cal(stockPosition, stockPrice, riskFreeRate, strategy,
                    initialMoney, rowNo);
   if (&rowNo. = 1) then do;
      &strategy.CashValue = &initialMoney.-&initialMoney.*&stockPosition.;
      &strategy. = &initialMoney.;
      &strategy.StockAmount = &strategy.*&stockPosition./&stockPrice.;
      &strategy.CurrentCashValue = &strategy.CashValue;
  end;
  else do;
      &strategy.CurrentCashValue = &strategy.CashValue*(1+&riskFreeRate.);
      &strategy. = &strategy.CurrentCashValue
                 + &strategy.StockAmount*&stockPrice.;
      &strategy.StockAmount = &strategy.*&stockPosition./&stockPrice.;
      &strategy.CashValue = &strategy.*(1-&stockPosition.);
  end;
  &strategy.ExcessReturn = &strategy./lag(&strategy.)-1-&riskFreeRate.;
  &strategy.ExcessReturn2 = &strategy.ExcessReturn**2;
%mend port_val_cal;

%macro port_val_gen(ForecastProb, bearThreshold, bullThreadhold, riskFreeRate,
                    initialMoney, startDate, endDate);
   data PortValPath;
      set &ForecastProb. end=last;
      array stockPos[4] %do i = 1 %to 4; &&strategy&i..StockPos %end; ;
      retain
         %do i = 1 %to 4;
            &&strategy&i..CashValue 0 &&strategy&i..StockAmount 0
            &&strategy&i..CurrentCashValue 0 &&strategy&i..CumExcessReturn 0
            &&strategy&i..CumExcessReturn2 0
         %end; ;
      * stock position for each strategy;
      stockPos[1] = 0;
      stockPos[2] = 1;
      if(bearProb>=&bearThreshold.) then stockPos[3]=0;
      else stockPos[3]=1;
      if(bullProb>=&bullThreshold.) then stockPos[4]=1;
      else stockPos[4]=0;
      * wealth for each strategy;
      %do i = 1 %to 4;
         %port_val_cal(stockPosition=stockPos[&i.], stockPrice=price,
                       riskFreeRate=&riskFreeRate., strategy=&&strategy&i..,
                       initialMoney=&initialMoney., rowNo=_N_);
         &&strategy&i..CumExcessReturn+&&strategy&i..ExcessReturn;
         &&strategy&i..CumExcessReturn2+&&strategy&i..ExcessReturn2;
      %end;
```

```
      * Sharpe ratio and portfolio final wealth;
      if last then do;
         &strategy1.LastPorVal = &strategy1.;
         %do i = 2 %to 4;
            &&strategy&i..LastPorVal = &&strategy&i..;
            &&strategy&i..ExcessReturnMean =
               &&strategy&i..CumExcessReturn/(_N_-1);
            &&strategy&i..ExcessReturnStd =
               sqrt(&&strategy&i..CumExcessReturn2/(_N_-1)
                  - &&strategy&i..ExcessReturnMean**2);
            &&strategy&i..SharpeRatio = round(&&strategy&i..ExcessReturnMean
               /&&strategy&i..ExcessReturnStd, 0.0001);
            call symputx("&&strategy&i..SharpeRatio",
               &&strategy&i..SharpeRatio, 'G');
         %end;
      end;
   run;
%mend;

* plot portfolio values of different trading strategies;
%let color1=black;
%let color2=blue;
%let color3=red;
%let color4=green;
%macro perf_comp_plot_gen(nStrategies);
   proc sgplot data=PortValPath cycleattrs;
      %do i = 1 %to &nStrategies.;
         %let strategy=&&strategy&i..;
         series x=date y=&&strategy&i.. / name="&strategy."
            lineattrs=(color=&&color&i..)
            %if (&i.=1) %then %do;
               legendlabel="&&strategy&i.Label.";
            %end;
            %else %do;
               legendlabel="&&strategy&i.Label. (SR=&&&strategy.SharpeRatio.)";
            %end;
         scatter x=date y=&&strategy&i..LastPorVal / datalabelpos=topright
            markerattrs=(color=&&color&i..)
            datalabel=&&strategy&i..LastPorVal;
      %end;
      xaxis values=("01JAN2001"d to "31DEC2017"d by year) display=(nolabel)
         tickvalueformat=year4. offsetmin=0.03 offsetmax=0.15;
      yaxis max=2.6 min=0.5 label="Wealth Curve";
      keylegend %do i = 1 %to &nStrategies.; "&&strategy&i.." %end; /
         location=outside position=bottom;
   run;
%mend perf_comp_plot_gen;

%port_val_gen(ForecastProb=ForecastProb,
              bearThreshold=&bearThreshold., bullThreadhold=&bullThreshold.,
              riskFreeRate=&riskFreeRate., initialMoney=&initialMoney.,
              startDate=&forecastStartDate., endDate=&forecastEndDate.);
%perf_comp_plot_gen(nStrategies=4);
```

Output 20.1.31 Wealth Curves of Four Trading Strategies



As shown in Output 20.1.31, thanks to the forecast of market states by the eight-state RS-AR(4) model, both bear and bull strategies beat the market: compared to the market strategy, they increase the Sharpe ratio more than 50% and get higher portfolio values at the end of out-of-sample period.

Example 20.2: Clustering Time Series by Using the Gaussian Hidden Markov Model

Clustering time series is a very common method of analyzing cross-sectional time series data. For more information, see Liao (2005) and Rani and Sikka (2012) and references therein. Especially for methods that use hidden Markov models (HMMs), see Smyth (1997), Oates, Firoiu, and Cohen (1999), and Ghassempour, Girosi, and Maeder (2014).

This example illustrates an algorithm for quickly clustering time series. With prior knowledge of the number of clusters and the number of states for each cluster, the algorithm creates the initial clusters according to the distribution of the likelihoods on an arbitrary HMM. Then, one HMM on each cluster is estimated and the time series are moved between clusters according to their likelihoods on those HMMs. The process is repeated until the algorithm converges or the maximum number of iterations is reached (Liao 2005).

In this example, the data generating process (DGP) is the same one that is described in Smyth (1997). There are two bistate Gaussian HMMs. For the first Gaussian HMM, the parameters are

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}, \mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

For the second Gaussian HMM, the parameters are

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.4 & 0.6 \\ 0.6 & 0.4 \end{pmatrix}, \mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

The only difference between these two Gaussian HMMs is that they have different transition probability matrices, so the time series that are generated from the first HMM have slower dynamics than those from the second HMM. Hence, the clustering time series algorithm should assign the time series that are generated from the first HMM to a cluster, and assign those from the second HMM to another cluster, if there is prior knowledge that the number of clusters is two and the number of states for each cluster is two.

The following macro generates the cross-sectional time series data table for analysis:

```
title "Clustering Time Series";

* create a data table dgpTable
  consisting of nSections sections and T observations for each section;
* then copy the data table dgpTable to the data table trainingTable
  and the data table libTrainingTable;
%macro createClustersTable(nSections,T,
                           dgpTable,trainingTable,libTrainingTable);
   %let seed = 12345;
   %let nClustersDGP = 2;
   %let c1portion = 0.5;

   %let c1pi1 = 0.5;
   %let c1a11 = 0.6;
   %let c1a22 = 0.6;
   %let c1mu1 = 0;
   %let c1sigma1 = 1;
   %let c1mu2 = 3;
   %let c1sigma2 = 1;

   %let c2pi1 = 0.5;
   %let c2a11 = 0.4;
   %let c2a22 = 0.4;
   %let c2mu1 = 0;
   %let c2sigma1 = 1;
   %let c2mu2 = 3;
   %let c2sigma2 = 1;
   data &dgpTable.;
```

```
      do sec = 1 to &nSections.;
         * which cluster;
         * (1) randomly assign clusters;
         * u = uniform(&seed.);
         * if(u<=&c1portion.) then cluster = 1;
         * else cluster = 2;
         * (2) all time series for one cluster are in one block;
         if(sec/&nSections.<=&c1portion.) then cluster = 1;
         else cluster = 2;
         %do i = 1 %to &nClustersDGP.;
            if(cluster=&i.) then do;
               do t = 1 to &T.;
                  if(t=1) then p = &&c&i.pi1;
                  else do;
                     if(lagState=1) then p = &&c&i.a11;
                     else p = 1-&&c&i.a22;
                  end;
                  u = uniform(&seed.);
                  if(u<=p) then state = 1;
                  else state = 2;
                  if(state=1) then
                     y = &&c&i.mu1. + sqrt(&&c&i.sigma1.)*normal(&seed.);
                  else
                     y = &&c&i.mu2. + sqrt(&&c&i.sigma2.)*normal(&seed.);
                  output;
                  lagState = state;
               end;
            end;
         %end;
      end;
   run;

   * trainingTable is sorted by sec and t;
   data &trainingTable.;
      set &dgpTable.;
      keep sec t y;
   run;

   data &libTrainingTable.;
      set &trainingTable.;
   run;
%mend;
```

The following macro initializes the clusters:

```
%macro estSection(libTrainingTable,casSectionTable,selectedSec,i,T,
                  evalTable,libModelTable);
   data &casSectionTable.&selectedSec.;
      set &libTrainingTable.;
      if(sec=&selectedSec.);
   run;

   proc hmm data=&casSectionTable.&selectedSec.;
      id section=sec time=t;
```

```
      model y  / method=ml type=gaussian nstate=&k.;
      optimize ALGORITHM=interiorpoint printlevel=3 printIterFreq=1;
      score outmodel=&libModelTable.&i.;
   run;

   proc hmm data=&libTrainingTable.;
      evaluate out=mylib.tempEval;
      score inmodel=&libModelTable.&i.;
   run;

   data &evalTable.&i.;
      set mylib.tempEval;
      logLikelihood&i. = logLikelihood;
      if(t=&T.) then output;
      keep sec logLikelihood&i.;
   run;

   proc sort data=&evalTable.&i.; by sec; run;
%mend;

%macro assignSeriesToClusters(evalTable,nClusters);
   data &evalTable.;
      merge &evalTable.1 - &evalTable.&nClusters.;
      by sec;
      retain sumLogLikelihood 0;
      array arr[*] logLikelihood1 - logLikelihood&nClusters.;
      sumLogLikelihood = sumLogLikelihood + max(of arr[*]);
      pCluster = whichn(max(of arr[*]), of arr[*]);
      keep sec pCluster sumLogLikelihood;
   run;
%mend;

* initialize clusters;
* each section has T observations;
%macro ctsInitClusters(libTrainingTable,nClusters,nSections,T,
                       evalTable,libModelTable);
   * as a start, use first section or randomly select a section;
   %let selectedSec = 1;
   %let i = 0;
   %estSection(&libTrainingTable.,mylib.tempSection,
               &selectedSec.,&i.,&T.,
               &evalTable.,&libModelTable.);
   proc sort data=&evalTable.&i.; by descending logLikelihood&i.; run;

   %do i = 1 %to &nClusters.;
      data _null_;
         set &evalTable.0;
         pos = floor(1+(&nSections.-1)*(&i.-1)/(&nClusters.-1)+0.5);
         if(_N_=pos) then
            call symput('selectedSec',trim(left(put(sec,12.))));
      run;
      %estSection(&libTrainingTable.,mylib.tempSection,
                  &selectedSec.,&i.,&T.,
                  &evalTable.,&libModelTable.);
```

```
   %end;

   %assignSeriesToClusters(&evalTable.,&nClusters.);
%mend;
```

The following macro repeatedly moves the time series between clusters until no movement is necessary:

```
* estimate HMM for cluster i;
* each section has T observations;
%macro estCluster(libClusteredTable,libTrainingTable,i,T,
                  evalTable,libModelTable);
   data &libClusteredTable.&i.;
      set &libClusteredTable.;
      if(pCluster=&i.);
   run;

   proc hmm data=&libClusteredTable.&i.;
      id section=sec time=t;
      model y  / method=ml type=gaussian nstate=&k.;
      optimize ALGORITHM=interiorpoint printlevel=3 printIterFreq=1;
      score outmodel=&libModelTable.&i.;
   run;

   proc hmm data=&libTrainingTable.;
      evaluate out=mylib.tempEval;
      score inmodel=&libModelTable.&i.;
   run;

   data &evalTable.&i.;
      set mylib.tempEval;
      logLikelihood&i. = logLikelihood;
      if(t=&T.) then output;
      keep sec logLikelihood&i.;
   run;

   proc sort data=&evalTable.&i.; by sec; run;
%mend;

* moves time series between clusters based on their likelihoods;
* each section has T observations;
* (1) &libModelTable.&i. has the model estimates for cluster i,
* and (2) &evalTable. has the clustering result;
%macro ctsClustering(trainingTable,libTrainingTable,nClusters,T,
                     libClusteredTable,evalTable,libModelTable);
   * save last clustering result;
   data prev&evalTable.; set &evalTable.; run;

   * clustering data;
   data &libClusteredTable.;
      merge &trainingTable.(in=a) &evalTable.(in=b);
      by sec;
   run;

   %do i = 1 %to &nClusters.;
```

```
      %estCluster(&libClusteredTable.,&libTrainingTable.,&i.,&T.,
                  &evalTable.,&libModelTable.);
   %end;

   %assignSeriesToClusters(&evalTable.,&nClusters.);

   * compare current and previous clustering results;
   proc compare data=&evalTable. compare=prev&evalTable.; run;
   %let converged=%eval(&sysinfo.=0);
   %let nIter=%eval(&nIter.+1);
   %let done=%eval(&converged. or &nIter.>=&maxIter.);

   %put nIter     = &nIter.;
   %put converged = &converged.;
   %put done      = &done.;
%mend;

* cluster time series;
* each section has T observations;
%macro ctsLoopClustering(trainingTable,libTrainingTable,nClusters,T,
                         libClusteredTable,evalTable,libModelTable);
   %let done = 0;     * terminate clustering;
   %do %while(&done.=0);
         %ctsClustering(&trainingTable.,&libTrainingTable.,&nClusters.,&T.,
                     &libClusteredTable.,&evalTable.,&libModelTable.);
   %end;
%mend;
```

Because the DGP is known, you can use the following macro to evaluate the performance of the clustering algorithm:

```
* compare the clustering results with the true clusters in the DGP;
* it is applicable only when the DGP is available;
%macro evaluateClusteringResults(dgpTable,evalTable,nSections,T);
   data true&evalTable.;
      set &dgpTable.;
      if(t=&T.);
      keep sec cluster;
   run;

   data ctsCheck;
      merge true&evalTable.(in=a) &evalTable.(in=b);
      by sec;
      if(cluster=pCluster) then do; correct=1; end;
      else do; correct=0; end;
      if(a=b);
   run;

   data ctsAccuracy;
      set ctsCheck;
      retain count 0 correctCount 0;
      count = count + 1;
      correctCount = correctCount + correct;
      if(count>&nSections.-0.5) then do;
```

```
         accuracy = correctCount / count;
         if(accuracy<0.5) then accuracy = 1 - accuracy;
         nIterations = &nIter.;
         converged = &converged.;
         output;
      end;
      keep nIterations converged accuracy;
   run;

   proc print data = ctsAccuracy noobs;
      var nIterations converged accuracy;
      format accuracy 6.4;
   run;
%mend;
```

As data become easy to access, the number of sections might become large. In this example, the number of sections is set to 10,000. As a start, the sample size for each time series is set to 200, which is the same as the sample size in Smyth (1997) and Oates, Firoiu, and Cohen (1999). In fact, the time series might have different lengths; if so, you should replace the log likelihood with the log likelihood per observation when you compare the time series.

The following macro calls simulate 10,000-section time series, apply the clustering algorithm, and then evaluate its performance:

```
%let N = 10000;
%let T = 200;
%createClustersTable(&N.,&T.,ctsDGP,ctsTrain,mylib.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;           * number of states in the HMM for each cluster;
%ctsInitClusters(mylib.ctsTrain,&nClusters.,&N.,&T.,
    evalCluster,mylib.modelCluster);
%let maxIter = 100;* maximum number of iterations;
%let nIter = 0;    * number of iterations;
%let converged = 0;* converge status;
%ctsLoopClustering(ctsTrain,mylib.ctsTrain,&nClusters.,&T.,
    mylib.ctsTrainClustered,evalCluster,mylib.modelCluster);
%evaluateClusteringResults(ctsDGP,evalCluster,&N.,&T.);
```

As shown in Output 20.2.1, the accuracy of the clustering algorithm is very high.

Output 20.2.1 Clustering Accuracy ($N = 10{,}000, T = 200$)

Clustering Time Series

nIterations	converged	accuracy
5	1	0.9868

The following statements display the parameter estimates for cluster 1 and cluster 2 in Output 20.2.2 and Output 20.2.3, respectively:

```
* print parameter estimates for cluster 1;
proc hmm data=mylib.ctsTrain;
   score inmodel=mylib.modelCluster1;
run;
```

```
* print parameter estimates for cluster 2;
proc hmm data=mylib.ctsTrain;
   score inmodel=mylib.modelCluster2;
run;
```

As shown in Output 20.2.2 and Output 20.2.3, the parameter estimates of the Gaussian HMMs for the two clusters are very close to the true parameters in the DGP.

Output 20.2.2 Parameter Estimates for Cluster 1 ($N = 10{,}000, T = 200$)

Clustering Time Series

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.598877	0.000977	612.98	<.0001
TPM1_2	0.401123	0.000977	410.57	<.0001
TPM2_1	0.399884	0.000979	408.35	<.0001
TPM2_2	0.600116	0.000979	612.82	<.0001
MU1_1	-0.000368	0.002626	-0.14	0.8885
MU2_1	2.997355	0.002636	1137.11	<.0001
SIGMA1_1_1	0.994514	0.003457	287.69	<.0001
SIGMA2_1_1	1.002381	0.003487	287.43	<.0001

Output 20.2.3 Parameter Estimates for Cluster 2 ($N = 10{,}000, T = 200$)

Clustering Time Series

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.399589	0.001164	343.20	<.0001
TPM1_2	0.600411	0.001164	515.69	<.0001
TPM2_1	0.600027	0.001161	516.61	<.0001
TPM2_2	0.399973	0.001161	344.37	<.0001
MU1_1	-0.000618	0.002597	-0.24	0.8119
MU2_1	2.999476	0.002603	1152.48	<.0001
SIGMA1_1_1	0.995491	0.003433	290.00	<.0001
SIGMA2_1_1	0.999145	0.003447	289.87	<.0001

In the second scenario, the sample size of one time series is increased to 400:

```
%let N = 10000;
%let T = 400;
%createClustersTable(&N.,&T.,ctsDGP,ctsTrain,mylib.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;           * number of states in the HMM for each cluster;
%ctsInitClusters(mylib.ctsTrain,&nClusters.,&N.,&T.,
    evalCluster,mylib.modelCluster);
%let maxIter = 100;* maximum number of iterations;
%let nIter = 0;    * number of iterations;
```

```
%let converged = 0;* converge status;
%ctsLoopClustering(ctsTrain,mylib.ctsTrain,&nClusters.,&T.,
    mylib.ctsTrainClustered,evalCluster,mylib.modelCluster);
%evaluateClusteringResults(ctsDGP,evalCluster,&N.,&T.);
```

As the sample size increases, more information becomes available for each time series. As expected, the accuracy increases (to almost 100% in this case). Output 20.2.4 shows the accuracy.

Output 20.2.4 Clustering Accuracy ($N = 10{,}000, T = 400$)

Clustering Time Series

nIterations	converged	accuracy
4	1	0.9989

The following statements print the parameter estimates for both clusters, which also become closer to the true parameter values, as shown in Output 20.2.5 and Output 20.2.6:

```
* print parameter estimates for cluster 1;
proc hmm data=mylib.ctsTrain;
   score inmodel=mylib.modelCluster1;
run;

* print parameter estimates for cluster 2;
proc hmm data=mylib.ctsTrain;
   score inmodel=mylib.modelCluster2;
run;
```

Output 20.2.5 Parameter Estimates for Cluster 1 ($N = 10{,}000, T = 400$)

Clustering Time Series

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.599184	0.000689	869.96	<.0001
TPM1_2	0.400816	0.000689	581.95	<.0001
TPM2_1	0.400637	0.000690	580.78	<.0001
TPM2_2	0.599363	0.000690	868.85	<.0001
MU1_1	-0.000181	0.001851	-0.10	0.9219
MU2_1	2.999973	0.001855	1617.37	<.0001
SIGMA1_1_1	0.996500	0.002443	407.90	<.0001
SIGMA2_1_1	0.999350	0.002452	407.56	<.0001

Output 20.2.6 Parameter Estimates for Cluster 2 ($N = 10{,}000, T = 400$)

Clustering Time Series

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.399302	0.000827	482.79	<.0001
TPM1_2	0.600698	0.000827	726.30	<.0001
TPM2_1	0.600445	0.000827	725.85	<.0001
TPM2_2	0.399555	0.000827	483.00	<.0001
MU1_1	2.998405	0.001852	1619.15	<.0001
MU2_1	-0.000477	0.001852	-0.26	0.7966
SIGMA1_1_1	1.001306	0.002449	408.85	<.0001
SIGMA2_1_1	1.001528	0.002449	408.92	<.0001

In the last scenario, the sample size for each time series decreases to 20 (see the following statements), which is a very small number. This case is common when the sampling frequency cannot be high (for example, yearly tax return data from millions of families). The information in one time series is very limited; this might even hinder the estimation of the hidden Markov models. However, as shown in this example, by clustering the time series, you condense the common information in time series, and then the true DGP is recovered.

```
%let N = 10000;
%let T = 20;
%createClustersTable(&N.,&T.,ctsDGP,ctsTrain,mylib.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;           * number of states in the HMM for each cluster;
%ctsInitClusters(mylib.ctsTrain,&nClusters.,&N.,&T.,
    evalCluster,mylib.modelCluster);
%let maxIter = 100;* maximum number of iterations;
%let nIter = 0;    * number of iterations;
%let converged = 0;* converge status;
%ctsLoopClustering(ctsTrain,mylib.ctsTrain,&nClusters.,&T.,
    mylib.ctsTrainClustered,evalCluster,mylib.modelCluster);
%evaluateClusteringResults(ctsDGP,evalCluster,&N.,&T.);
```

As expected and as shown in Output 20.2.7, the algorithm needs many more iterations to converge, and the accuracy of the clustering algorithm is not high.

Output 20.2.7 Clustering Accuracy ($N = 10{,}000, T = 20$)

Clustering Time Series

nIterations	converged	accuracy
26	1	0.7552

The following statements print the parameter estimates for both clusters, which are shown in Output 20.2.8 and Output 20.2.9. Although each time series might contain little information, the parameter estimates of the Gaussian HMMs for the two clusters are very close to the true parameters in the DGP.

```
* print parameter estimates for cluster 1;
proc hmm data=mylib.ctsTrain;
   score inmodel=mylib.modelCluster1;
run;

* print parameter estimates for cluster 2;
proc hmm data=mylib.ctsTrain;
   score inmodel=mylib.modelCluster2;
run;
```

Output 20.2.8 Parameter Estimates for Cluster 1 ($N = 10{,}000, T = 20$)

Clustering Time Series

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.357167	0.003560	100.32	<.0001
TPM1_2	0.642833	0.003560	180.56	<.0001
TPM2_1	0.638448	0.003510	181.91	<.0001
TPM2_2	0.361552	0.003510	103.02	<.0001
MU1_1	-0.002996	0.007409	-0.40	0.6860
MU2_1	2.989537	0.007619	392.36	<.0001
SIGMA1_1_1	0.965620	0.009808	98.45	<.0001
SIGMA2_1_1	1.011633	0.010366	97.59	<.0001

Output 20.2.9 Parameter Estimates for Cluster 2 ($N = 10{,}000, T = 20$)

Clustering Time Series

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.638344	0.002875	222.00	<.0001
TPM1_2	0.361656	0.002875	125.78	<.0001
TPM2_1	0.365361	0.002894	126.23	<.0001
TPM2_2	0.634639	0.002894	219.26	<.0001
MU1_1	3.010552	0.007647	393.69	<.0001
MU2_1	0.011987	0.007950	1.51	0.1316
SIGMA1_1_1	0.981346	0.010152	96.66	<.0001
SIGMA2_1_1	1.029845	0.010807	95.30	<.0001

Example 20.3: Analysis of the Business Cycle by Using the Regime-Switching Mean-Adjusted Autoregression Model

This example revisits the model in Hamilton (1989) to illustrate how the regime-switching mean-adjusted autoregression model is used to estimate the business cycle with quarterly US real gross national product (GNP) data from 1951 to 1984.

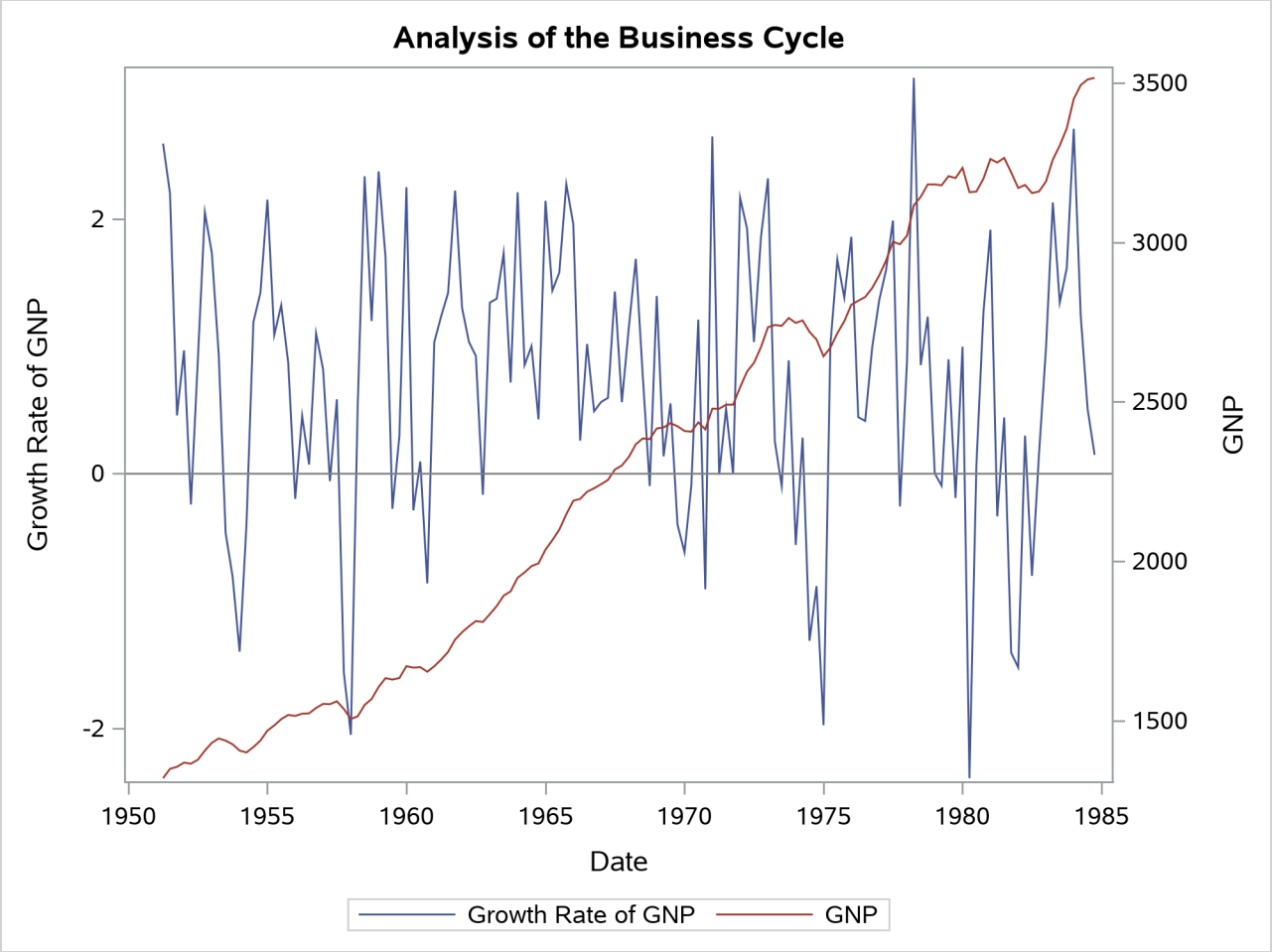
The following statements import the data and calculate the GNP growth rate (in percentage) to which the model applies:

```
title 'Analysis of the Business Cycle';
data gnpHamilton;
   input date: anydtdte. gnp;
   format date yyq6.;
   dgnp = 100*(log(gnp)-log(lag(gnp)));
   if cmiss(of _all_) then delete;
   label date="Date" gnp="GNP" dgnp="Growth Rate of GNP";
datalines;
1951q1  1286.6
1951q2  1320.4
1951q3  1349.8

   ... more lines ...

1982q4  3159.3
1983q1  3190.6
1983q2  3259.3
1983q3  3303.4
1983q4  3357.2
1984q1  3449.4
1984q2  3492.6
1984q3  3510.4
1984q4  3515.6
;
```

The following statements plot the data in Output 20.3.1:

```
proc sgplot data=gnpHamilton;
   series x=date y=dgnp;
   series x=date y=gnp / y2axis;
   refline 0 / axis=y lineattrs=(color=gray thickness=1);
run;
```

Output 20.3.1 GNP and Growth Rate of GNP



Hamilton's model is a bistate regime-switching mean-adjusted AR(4) model which can be described as

$$y_t = c^{(s_t)} + \sum_{\tau=1}^{4} \alpha_\tau^{(s_t)}(y_{t-\tau} - c^{(s_{t-\tau})}) + \varepsilon_t,\ \varepsilon_t \sim N(0, \Sigma^{(s_t)})$$

where $s_t \in \{1, 2\}$ is the unobserved state.

In Hamilton (1989), the autoregressive parameters α's and variance parameters Σ's are assumed to be state-independent, which means $\alpha_\tau^{(1)} = \alpha_\tau^{(2)}, \tau = 1, \ldots, 4$, and $\Sigma^{(1)} = \Sigma^{(2)}$. The following statements re-create Hamilton's result by setting the initial parameter values as the parameter estimates in Table 1 in Hamilton (1989) and also setting the MAXITER= option in the OPTIMIZE statement to zero. The FILTER statement conducts the probabilities of state on the basis of data available up to the current period. The DECODE statement outputs the best possible path of states on the basis of all observations. The SMOOTH statement finds the probabilities of states on the basis of all observations.

```
data mylib.gnpHamilton; set gnpHamilton; run;
proc hmm data=mylib.gnpHamilton labelswitch=(sort=desc(const));
   id time=date;
   model dgnp / armean=adjusted type=ar ylag=4 method=ml nstate=2
                stateIndependent=(ar cov);
   optimize maxiter=0;
   initial tpm={0.9049 0.0951, 0.2450 0.7550},
      ar={0.014 -0.058 -0.247 -0.213, 0.014 -0.058 -0.247 -0.213},
      const={1.1643, -0.3577},
      cov={0.5914, 0.5914};
   filter out=mylib.filter;
   decode out=mylib.decode;
   smooth out=mylib.smooth;
run;
```

The number of observations and model information are shown in Output 20.3.2.

Output 20.3.2 Number of Observations and Model Information

Analysis of the Business Cycle

Observation Information	
Number of Observations	135
Number of Missing Observations	0

Model Information	
Type of Model	Regime-Switching Mean-adjusted Autoregression Model
Stationary	Yes
Number of States	2
Number of Dependent Variables	1
Number of Exogenous Variables	0

The fit statistics are shown in Output 20.3.3.

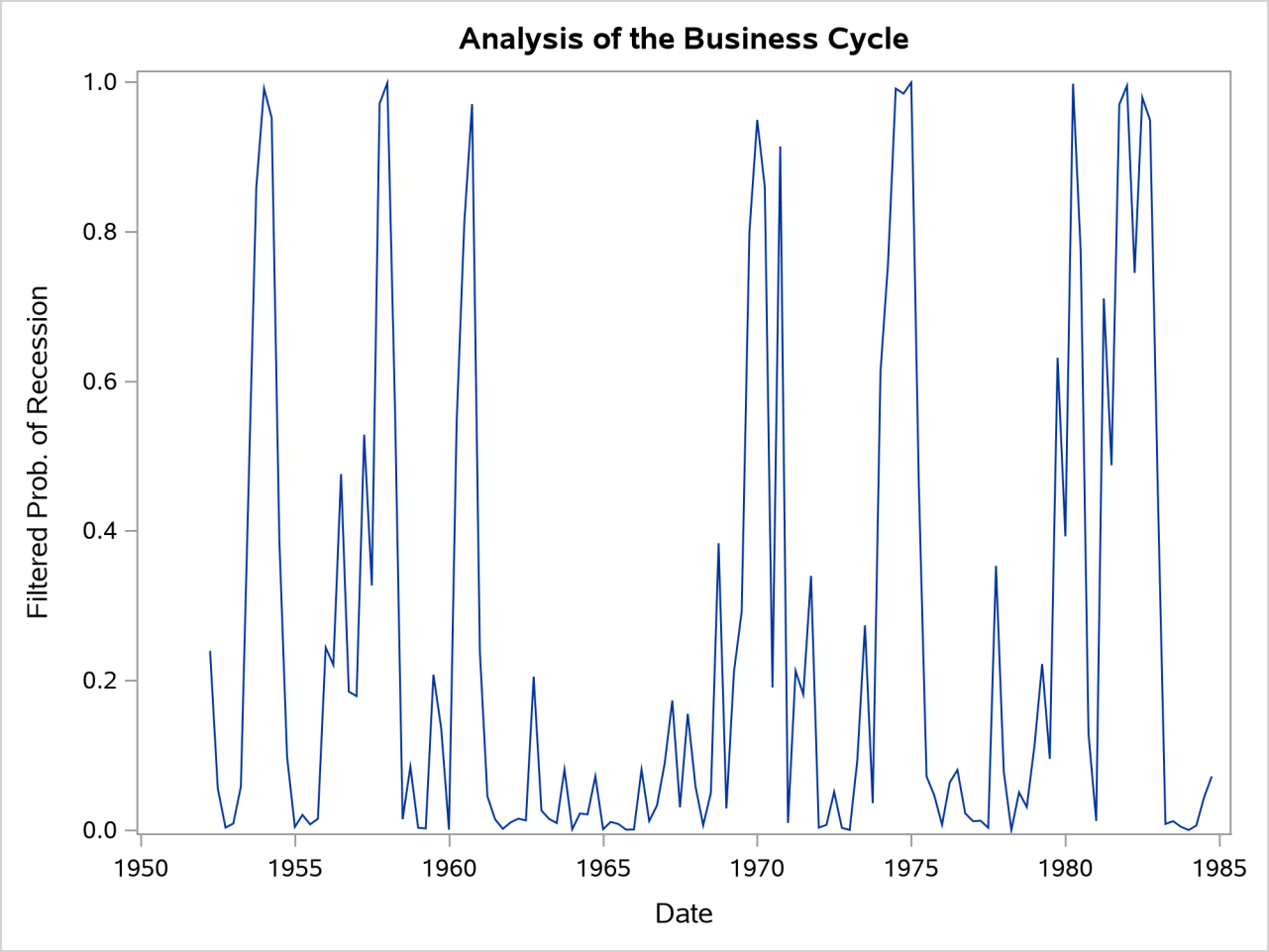
Output 20.3.3 Model Information

Fit Statistics	
Log Likelihood	-183.1751626
AIC	384.35032521
AICC	385.79032521
BIC	410.49779822
HQC	394.97592522

The following statements plot the filtered probabilities of state 2 in Output 20.3.4, which is comparable to Figure 1 in Hamilton (1989). State 2 corresponds to recession because the intercept (and then the expected mean) in state 2 is negative.

```
data filter; set mylib.filter; run;
proc sort data=filter; by date; run;
proc sgplot data=filter;
   series x=date y=state2;
   yaxis label="Filtered Prob. of Recession";
run;
```

Output 20.3.4 Filter Plot of Economic Recession

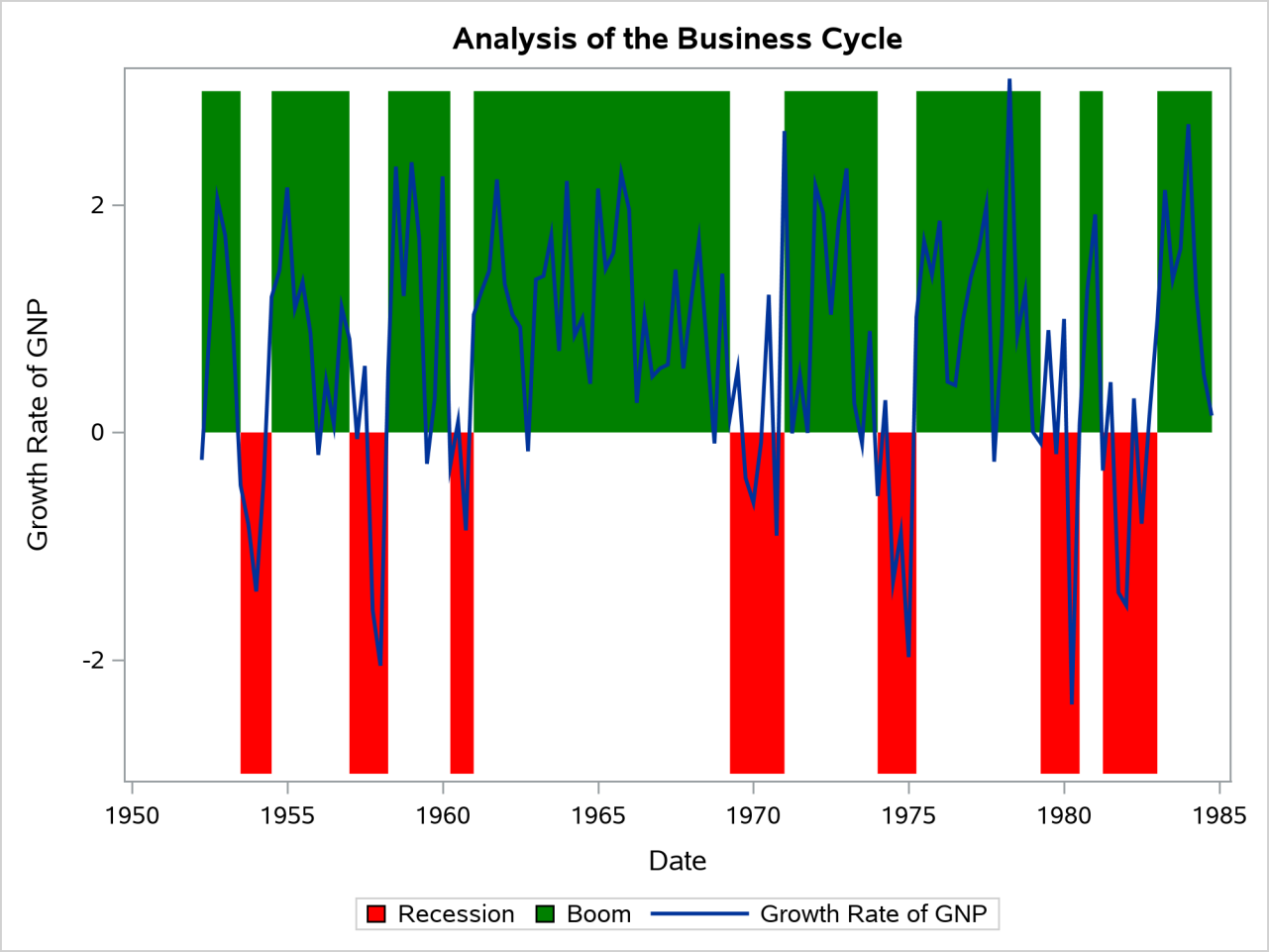


In order to show how states correspond to the signs of GNP growth rate, the following statements overlay the decoded states on the GNP growth rate. Recession is from peak to trough, and boom is from trough to peak. As shown in Output 20.3.5, the recession includes most periods that have a negative GNP growth rate.

```
data plot2state;
   merge mylib.decode mylib.gnpHamilton;
   if state=2 then do; recession=1; boom=0; end;
   else if state=1 then do; recession=0; boom=1; end;
   rec = recession*(-3);
   boo = boom*3;
   by date;
   if cmiss(of _all_) then delete;
run;

proc sgplot data=plot2state;
```

```
   band x=date upper=0 lower=rec / legendlabel='Recession' type=step
                                   fillattrs=(color=red);
   band x=date upper=boo lower=0 / legendlabel='Boom' type=step
                                   fillattrs=(color=green);
   series x=date y=dgnp / lineattrs=(thickness=2);
run;
```

Output 20.3.5 Decoded Plot of Business Cycle



Hamilton (1989) also compares his choice of business-cycle dates with the historical business-cycle dates that are reported from the National Bureau of Economic Research (NBER). The following data step inputs the business-cycle dates from NBER:

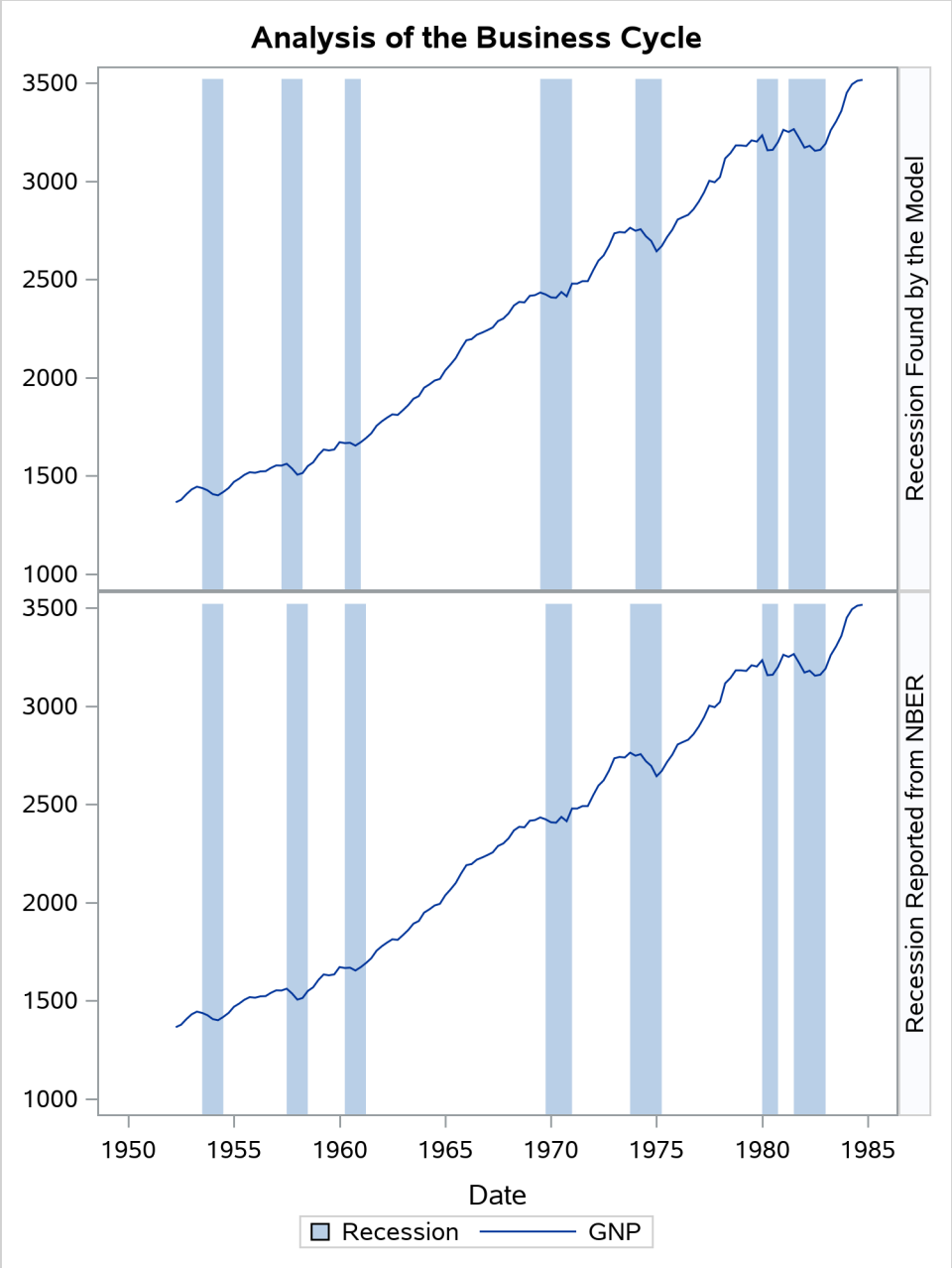
```
data mylib.nber;
   input date: anydtdte. recessionNBER;
   format date yyq6.;
   label date="Date" recessionNBER="Recession Reported from NBER";
datalines;
1951q1  0
1951q2  0
1951q3  0
1951q4  0
1952q1  0
```

```
1952q2  0

   ... more lines ...

1982q4  1
1983q1  0
1983q2  0
1983q3  0
1983q4  0
1984q1  0
1984q2  0
1984q3  0
1984q4  0
;
```

The following statements plot together the recession dates that are found by the model and those that are reported from NBER. As shown in Output 20.3.6, the recession dates that are found by Hamilton's model (the upper part) is extremely close to the dates that are reported from NBER (the lower part).

```
data gnpsmooth;
   merge mylib.smooth mylib.nber mylib.gnpHamilton;
   if state2>=0.5 then recession=1; else recession=0;
   by date;
   gnp2=gnp;
   if cmiss(of _all_) then delete;
   label recession="Recession Found by the Model" gnp2="GNP";
run;
proc transpose data=gnpsmooth  out=gnpsmooth_t;
   by date;
   var recessionNBER recession;
run;
proc transpose data=gnpsmooth  out=gnpsmooth_t2;
   by date;
   var gnp gnp2;
run;
data gnpplot;
   merge gnpsmooth_t(rename=(_name_=name1 _label_=label1 col1=recession))
         gnpsmooth_t2(rename=(_name_=name2 _label_=label2 col1=gnp));
   by date;
   rec=recession*3520;
run;
proc sgpanel data=gnpplot;
   panelby label1 / layout=rowlattice uniscale=column novarname;
   band x=date upper=rec lower=0 / legendlabel='Recession' type=step;
   series x=date y=gnp / legendlabel='GNP';
   rowaxis display=(nolabel) values=(1000 to 3520 by 500);
run;
```

Output 20.3.6 Smoothed Plot of Economic Recession



The following statements use PROC HMM to estimate Hamilton's model while using the STATEINDEPENDENT=(AR COV) option in the MODEL statement so that the autoregression parameter and the variance are fixed across the states:

```
proc hmm data=mylib.gnpHamilton labelswitch=(sort=desc(const));
   id time=date;
   model dgnp / armean=adjusted type=ar ylag=4 method=ml nstate=2
            stateIndependent=(ar cov);
   optimize  printIterFreq=1 printlevel=3 ;
   filter out=mylib.filter;
```

```
   decode out=mylib.decode;
   smooth out=mylib.smooth;
run;
```

The parameter estimates are shown in Output 20.3.7, and the fit statistics are shown in Output 20.3.8. The log likelihood value and the information criterion in Output 20.3.8 show that this solution is better than the one in Output 20.3.3 when MAXITER=0.

Output 20.3.7 Parameter Estimates

Analysis of the Business Cycle

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.904414	0.037629	24.03	<.0001
TPM1_2	0.095586	0.037629	2.54	0.0123
TPM2_1	0.254962	0.099402	2.56	0.0115
TPM2_2	0.745038	0.099402	7.50	<.0001
CONST1_1	1.160773	0.077552	14.97	<.0001
CONST2_1	-0.392035	0.275951	-1.42	0.1580
AR1_1_1_1	0.016338	0.121529	0.13	0.8933
AR1_2_1_1	-0.088765	0.152015	-0.58	0.5604
AR1_3_1_1	-0.219363	0.114233	-1.92	0.0572
AR1_4_1_1	-0.184025	0.111810	-1.65	0.1024
AR2_1_1_1	0.016338	0.121529	0.13	0.8933
AR2_2_1_1	-0.088765	0.152015	-0.58	0.5604
AR2_3_1_1	-0.219363	0.114233	-1.92	0.0572
AR2_4_1_1	-0.184025	0.111810	-1.65	0.1024
COV1_1_1	0.609788	0.108633	5.61	<.0001
COV2_1_1	0.609788	0.108633	5.61	<.0001

Output 20.3.8 Fit Statistics

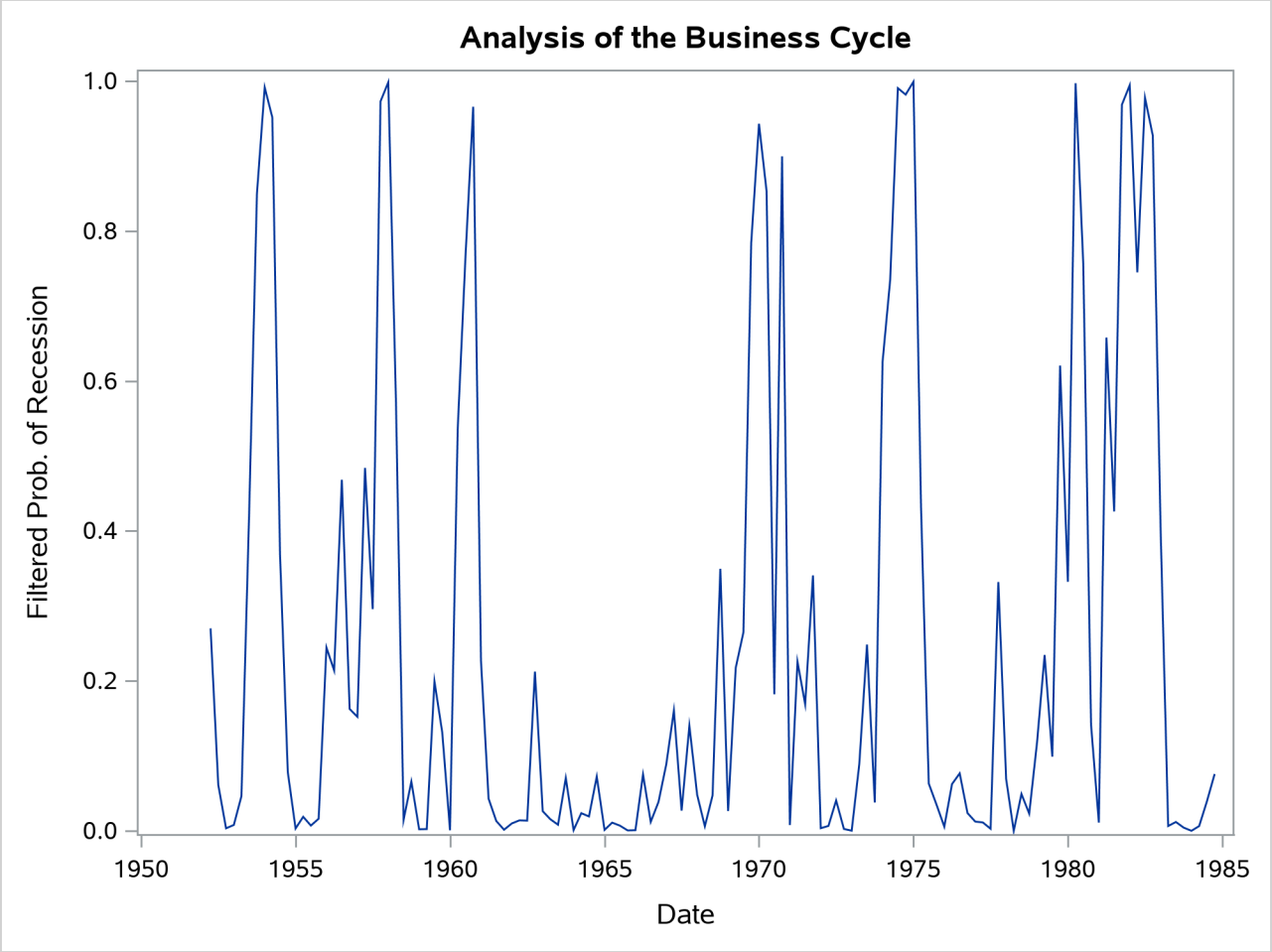
Fit Statistics

Log Likelihood	-183.0222956
AIC	384.04459127
AICC	385.48459127
BIC	410.19206427
HQC	394.67019127

The following statements plot the filtered probabilities of state 2 (recession). Output 20.3.9 shows the results.

```
data filter; set mylib.filter; run;
proc sort data=filter; by date; run;
proc sgplot data=filter;
   series x=date y=state2;
   yaxis label="Filtered Prob. of Recession";
run;
```

Output 20.3.9 Filter Plot of Economic Recession

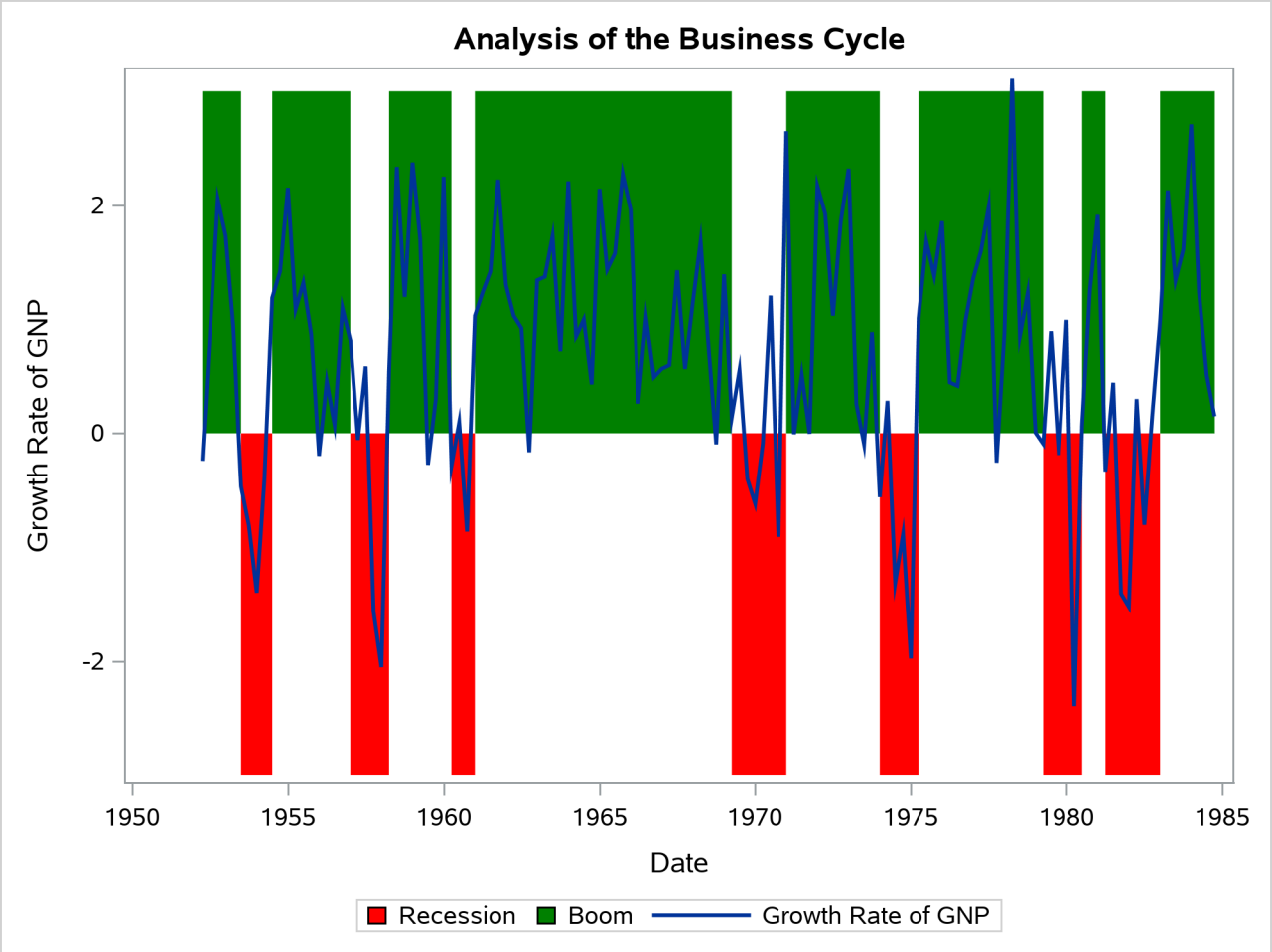


The following statements overlay the decoded states on the GNP growth rate. Recession is from peak to trough, and boom is from trough to peak.

```
data plot2state;
   merge mylib.decode mylib.gnpHamilton;
   if state=2 then do; recession=1; boom=0; end;
   else if state=1 then do; recession=0; boom=1; end;
   rec = recession*(-3);
   boo = boom*3;
   by date;
   if cmiss(of _all_) then delete;
run;

proc sgplot data=plot2state;
   band x=date upper=0 lower=rec / legendlabel='Recession' type=step
                                   fillattrs=(color=red);
   band x=date upper=boo lower=0 / legendlabel='Boom' type=step
                                   fillattrs=(color=green);
   series x=date y=dgnp / lineattrs=(thickness=2);
run;
```

Output 20.3.10 shows that more states switch than are shown in Hamilton's decoded plot. The reason is that the estimates indicate less persistence of each state.

Output 20.3.10 Decoded Plot of Business Cycle



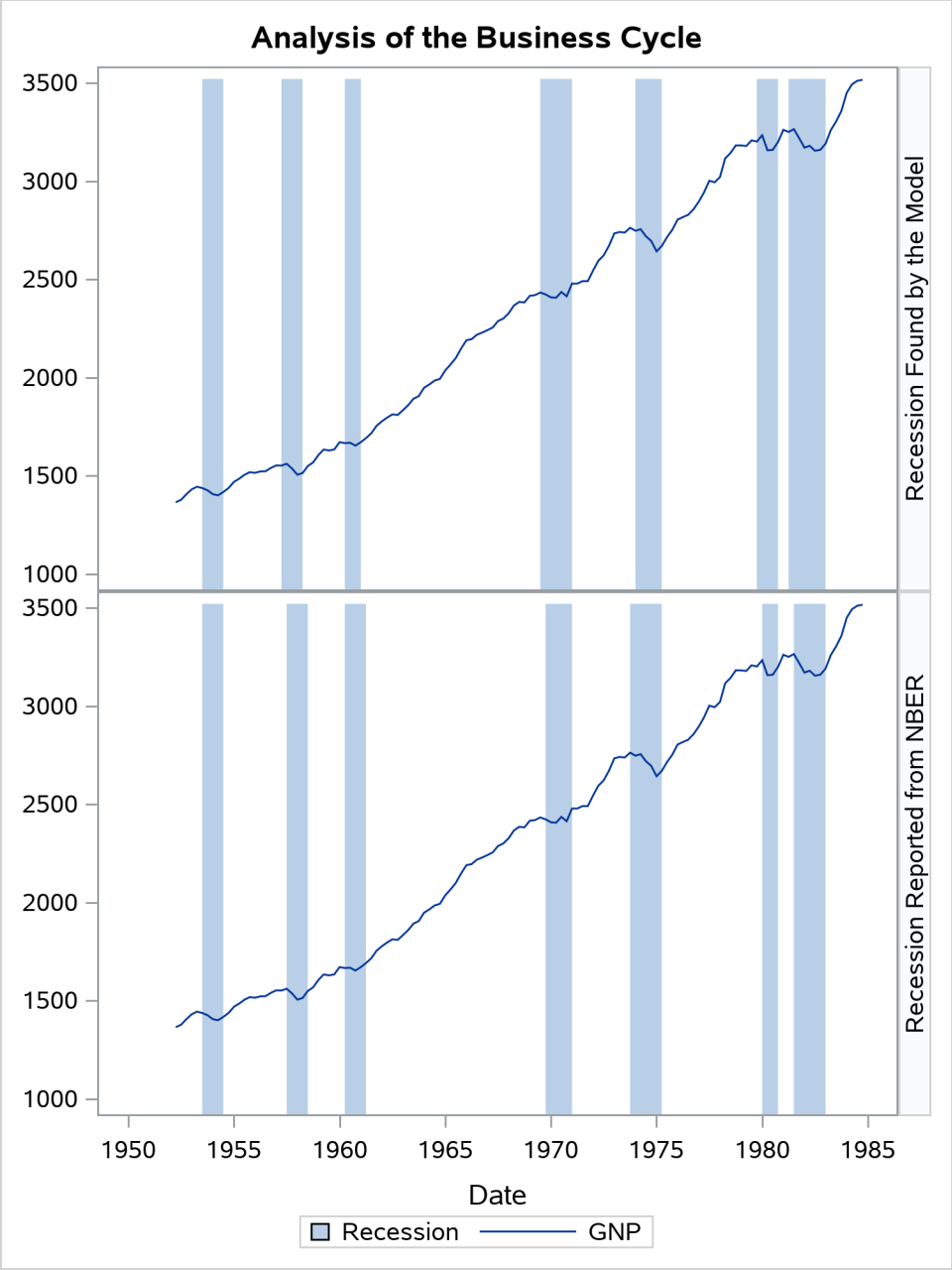
The following statements plot the recession dates that are found by the model and those that are reported from NBER:

```
data gnpsmooth;
   merge mylib.smooth mylib.nber mylib.gnpHamilton;
   if state2>=0.5 then recession=1; else recession=0;
   by date;
   gnp2=gnp;
   if cmiss(of _all_) then delete;
   label recession="Recession Found by the Model" gnp2="GNP";
run;
proc transpose data=gnpsmooth  out=gnpsmooth_t;
   by date;
   var recessionNBER recession;
run;
proc transpose data=gnpsmooth  out=gnpsmooth_t2;
   by date;
```

```
   var gnp gnp2;
run;
data gnpplot;
   merge gnpsmooth_t(rename=(_name_=name1 _label_=label1 col1=recession))
         gnpsmooth_t2(rename=(_name_=name2 _label_=label2 col1=gnp));
   by date;
   rec=recession*3520;
run;
proc sgpanel data=gnpplot;
   panelby label1 / layout=rowlattice uniscale=column novarname;
   band x=date upper=rec lower=0 / legendlabel='Recession' type=step;
   series x=date y=gnp / legendlabel='GNP';
   rowaxis display=(nolabel) values=(1000 to 3520 by 500);
run;
```

As shown in Output 20.3.11, the recession dates that are found by the model (the upper part) are extremely close to the dates that are reported from NBER (the lower part).

Output 20.3.11 Smoothed Plot of Economic Recession



Example 20.4: Analysis of English Characters by Using a Finite HMM

In natural language processing (NLP), practitioners want to learn the structure of sentences and predict the sequences of words. HMMs have been widely applied for characterizing the statistical features of natural languages; for more information, see Jurafsky and Martin (2019). This example runs the model from Cave and Neuwirth (1980) to illustrate how to use the finite HMM to analyze the English language. The data set used in this example was extracted and converted from the classic novel *Vanity Fair*. To process the raw text, first remove all punctuation and numbers and convert all letters to lowercase.

The following statements create the table for the first 5,000 symbols. The table has two columns: column t is the position of the symbol, and column character is the value at time *t*.

```
data text;
   input t character $8. ;
datalines;
1   c
2   h
3   a
4   p
5   t
6   e
7   r
8

   ... more lines ...

4992    c
4993    k
4994
4995    o
4996    n
4997
4998    h
4999    e
5000    r
;
```

When you get data set text, you can translate the column value into integers. The following statements relabel the character variables as the unique integers that the finite HMM procedure uses:

```
data tbltemp;
   set text;
   _temp_count_ = 1;
run;

proc sql;
   create table f0 as
     SELECT tbl.character, /* any other columns to be grouped */
            sum(tbl._temp_count_) as count
     FROM tbltemp tbl
     group by tbl.character
   ;
quit;
```

```
data f0Id;
   set f0;
   y = _n_;
   keep character y;
run;

proc print data=f0Id; run;

proc sort data=text out=textSort;
   by character;
run;

PROC SQL;
   CREATE TABLE textLeft AS
   SELECT A.*, B.y
   FROM textSort A LEFT JOIN f0Id B
   ON A.character = B.character
   ORDER BY character;
QUIT;

proc sort data=textLeft out=textLeft;
   by t;
run;

data text2;
   set textLeft;
run;

data mylib.text;;
   set text2;
run;
```

This converts the novel into a sequence of 26 distinct letters and the word space, for a total of 27 symbols. Then, translate the characters to numbers, a to 2, b to 3, . . . , z to 27, and word space to 1.

The conversion matching table is shown in Figure 20.4.1.

Output 20.4.1 Matching Table

Obs	character	y
1		1
2	a	2
3	b	3
4	c	4
5	d	5
6	e	6
7	f	7
8	g	8
9	h	9
10	i	10
11	j	11
12	k	12
13	l	13
14	m	14
15	n	15
16	o	16
17	p	17
18	q	18
19	r	19
20	s	20
21	t	21
22	u	22
23	v	23
24	w	24
25	x	25
26	y	26
27	z	27

The following statements print the table after value conversion. The table, which is shown in Figure 20.4.2, contains three columns: column t is the position of the symbol, column character is the original symbol, and column y is the translated value of the symbol at position t:

```
proc print data=text2(obs=10) noobs;
run;
```

Output 20.4.2 Converted Data of *Vanity Fair*

t	character	y
1	c	4
2	h	9
3	a	2
4	p	17
5	t	21
6	e	6
7	r	19
8		1
9	i	10
10	c	4

Cave and Neuwirth's model is assumed to have an underlying Markov chain with two hidden states. For each of these hidden states, assume that the 27 symbols appear according to a fixed probability distribution. The following statements estimate a finite HMM in which NSTATE=2 and NCATEGORY=27. There might be a lot of local maximums for the finite HMM; hence, you specify the multistart mode by specifying MULTISTART=1 in the OPTIMIZE statement.

```
data mylib.text;
   set text;
run;
proc hmm data=mylib.text ;
   id time=t;
   model y / type=finite nstate=2 nCategory=27;
   optimize algorithm=interiorpoint multistart=1;
run;
```

The number of observations and model information are shown in Output 20.4.3.

Output 20.4.3 Number of Observations and Model Information

Analysis of English Text

Observation Information	
Number of Observations	5000
Number of Missing Observations	0

Model Information	
Type of Model	Finite HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	1

The parameter estimates are shown in Output 20.4.4, and the fit statistics are shown in Output 20.4.5.

Output 20.4.4 Parameter Estimates

Parameter Estimates

Parameter	Estimate	Standard Error	t Value	Pr > |t|
TPM1_1	0.308898	0.012144	25.44	<.0001
TPM1_2	0.691102	0.012144	56.91	<.0001
TPM2_1	0.758620	0.016042	47.29	<.0001
TPM2_2	0.241380	0.016042	15.05	<.0001
CPM1_1	0.007426	0.009917	0.75	0.4540
CPM1_2	0.000841	0.004412	0.19	0.8489
CPM1_3	0.024076	0.003013	7.99	<.0001
CPM1_4	0.046623	0.004166	11.19	<.0001
CPM1_5	0.063820	0.004851	13.16	<.0001
CPM1_6	1.473516E-10	0.000000380	0.00	0.9997
CPM1_7	0.034394	0.003591	9.58	<.0001
CPM1_8	0.026369	0.003151	8.37	<.0001
CPM1_9	0.092864	0.005802	16.01	<.0001
CPM1_10	1.425877E-10	0.000000368	0.00	0.9997
CPM1_11	0.008407	0.001788	4.70	<.0001
CPM1_12	0.008347	0.003475	2.40	0.0163
CPM1_13	0.058470	0.004650	12.57	<.0001
CPM1_14	0.055031	0.004515	12.19	<.0001
CPM1_15	0.110443	0.006295	17.54	<.0001
CPM1_16	2.677046E-10	0.000000703	0.00	0.9997
CPM1_17	0.035541	0.003649	9.74	<.0001
CPM1_18	0.004204	0.001266	3.32	0.0009
CPM1_19	0.104711	0.006140	17.05	<.0001
CPM1_20	0.128023	0.006742	18.99	<.0001
CPM1_21	0.111687	0.008254	13.53	<.0001
CPM1_22	6.076342E-10	0.000001795	0.00	0.9997
CPM1_23	0.014140	0.002315	6.11	<.0001
CPM1_24	0.030190	0.003368	8.96	<.0001
CPM1_25	0.003057	0.001080	2.83	0.0047
CPM1_26	0.029808	0.003347	8.91	<.0001
CPM1_27	0.001529	0.000764	2.00	0.0455
CPM2_1	0.332555	0.012576	26.44	<.0001
CPM2_2	0.132087	0.008225	16.06	<.0001
CPM2_3	1.950648E-10	0.000000521	0.00	0.9997
CPM2_4	9.941711E-10	0.000002891	0.00	0.9997
CPM2_5	3.358829E-10	0.000000924	0.00	0.9997
CPM2_6	0.221124	0.009073	24.37	<.0001
CPM2_7	1.189225E-10	0.000000315	0.00	0.9997
CPM2_8	2.0441364E-9	0.000006289	0.00	0.9997
CPM2_9	3.0078461E-9	0.000009440	0.00	0.9997
CPM2_10	0.133430	0.007223	18.47	<.0001
CPM2_11	7.496966E-11	0.000000197	0.00	0.9997
CPM2_12	0.005101	0.003602	1.42	0.1568
CPM2_13	9.85672E-10	0.000002959	0.00	0.9997
CPM2_14	8.918453E-11	0.000000236	0.00	0.9997
CPM2_15	1.481432E-10	0.000000400	0.00	0.9997
CPM2_16	0.123360	0.006965	17.71	<.0001

Output 20.4.4 *continued*

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > |t|
CPM2_17	2.135985E-10	0.000000633	0.00	0.9997
CPM2_18	8.173257E-11	0.000000225	0.00	0.9997
CPM2_19	1.372299E-10	0.000000384	0.00	0.9997
CPM2_20	4.145852E-10	0.000001234	0.00	0.9997
CPM2_21	0.011222	0.006957	1.61	0.1068
CPM2_22	0.041120	0.004110	10.00	<.0001
CPM2_23	3.942147E-12	4.8333296E-8	0.00	0.9999
CPM2_24	6.578474E-12	8.3343256E-8	0.00	0.9999
CPM2_25	5.142148E-12	5.9097634E-8	0.00	0.9999
CPM2_26	1.0991882E-9	0.000005356	0.00	0.9998
CPM2_27	4.567458E-12	4.7271039E-8	0.00	0.9999

Output 20.4.5 Fit Statistics

Fit Statistics	
Log Likelihood	-13913.63033
AIC	27939.260663
AICC	27940.552186
BIC	28304.223482
HQC	28067.17439

The most interesting part of this result is the result of the category probability matrix (CPM). In state 1, the probabilities of containing 3, 4, 5, . . . , 26, which correspond to the consonants, are significant. In state 2, the probabilities of containing 1, 2, 6, 10, 16, 22, which correspond to the vowels a, e, i, o, and u and the space, are significant. Without your having made any assumptions about the two hidden states, the CPM shows that one hidden state contains the vowels and the other hidden state contains the consonants. Interestingly, the space is treated more like a vowel in the context of English. The PROC HMM result shows that the distinction between vowels and consonants is a statistically significant feature inherent in the language. And thanks to HMMs, this could easily be deduced by someone who has no background knowledge of the language.

References

Cappé, O., Moulines, E., and Rydén, T. (2010). *Inference in Hidden Markov Models*. New York: Springer.

Cave, R. L., and Neuwirth, L. P. (1980). "Hidden Markov Models for English." In *Hidden Markov Models for Speech*, edited by J. D. Ferguson, 16–57. Princeton, NJ: Institute for Defense Analysis.

Center for Research in Security Prices (2018). "CRSP NYSE/NYSEMKT/Nasdaq/Arca Value-Weighted Market Index." CRSP, University of Chicago Booth School of Business.

Frühwirth-Schnatter, S. (2006). *Finite Mixture and Markov Switching Models*. New York: Springer.

Ghassempour, S., Girosi, F., and Maeder, A. (2014). "Clustering Multivariate Time Series Using Hidden Markov Models." *International Journal of Environmental Research and Public Health* 11:2741–2763. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3968966/.

Hamilton, J. D. (1989). "A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle." *Econometrica* 57:357–384.

Jurafsky, D., and Martin, J. H. (2019). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Third edition, available as unpublished draft at https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf.

Krolzig, H.-M. (1997). *Markov-Switching Vector Autoregressions: Modelling, Statistical Inference, and Application to Business Cycle Analysis*. Berlin: Springer.

Kuester, K., Mittnik, S., and Paolella, M. S. (2006). "Value-at-Risk Prediction: A Comparison of Alternative Strategies." *Journal of Financial Econometrics* 4:53–89.

Liao, T. W. (2005). "Clustering of Time Series Data—a Survey." *Pattern Recognition* 38:1857–1874.

Oates, T., Firoiu, L., and Cohen, P. R. (1999). "Clustering Time Series with Hidden Markov Models and Dynamic Time Warping." In *Proceedings of the IJCAI-99 Workshop on Neural, Symbolic, and Reinforcement Learning Methods for Sequence Learning*, 17–21.

Rabiner, L. R. (1989). "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE* 77:257–286.

Rani, S., and Sikka, G. (2012). "Recent Techniques of Clustering of Time Series Data: A Survey." *International Journal of Computer Applications* 52:1–9.

Smyth, P. (1997). "Clustering Sequences with Hidden Markov Models." In *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, edited by M. C. Mozer, T. Petsche, and M. I. Jordan, 648–654. Cambridge, MA: MIT Press.

Sung, H. G. (2004). "Gaussian Mixture Regression and Classification." Ph.D. diss., Rice University.