

RATS Handbook for Switching Models and Structural Breaks

Thomas A. Doan
Estima

2nd Edition
January 18, 2026

Copyright © 2026 by Thomas A. Doan

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Contents

Preface	vii
1 Structural Breaks at Known Locations	1
1.1 Structural Breaks: Introduction	1
1.2 Breaks in Static Models	2
1.3 Breaks in Dynamic Models	5
1.4 RATS Tips and Tricks	9
1.1 Models with Exogenously-determined Trend Breaks	11
2 Fluctuation Tests	13
2.1 ICSS Test for Variance Changes	19
2.2 Rolling Sample Estimates	26
2.1 Simple Fluctuation Test	28
2.2 Fluctuation Test for GARCH	28
2.3 ICSS Analysis vs GARCH model	29
3 Parametric Tests	31
3.1 Full Coefficient Vector	31
3.1.1 Linear Least Squares	33
3.1.2 GMM	35
3.2 Outliers and Shifts	37
3.2.1 Linear Least Squares	37
3.2.2 ARIMA models	38
3.2.3 GARCH models	40
3.3 Fixed Regressor Bootstrap	43
3.1 Break Analysis for GMM	47
3.2 ARIMA Model with Outlier Handling	48
3.3 GARCH Model with Outlier Handling	49
3.4 Fixed Regressor Bootstrap	51

4	TAR Models	53
4.1	Estimation.	54
4.2	Testing	55
4.2.1	Arranged Autoregression Test	56
4.2.2	Direct Threshold Tests	56
4.3	Forecasting	58
4.4	Non-linear Impulse Responses.	61
4.5	Tips and Tricks	65
4.1	TAR Model for Unemployment	66
4.2	TAR Model for Interest Rate Spread	68
5	Threshold VAR/Cointegration	72
5.1	Threshold Error Correction	72
5.2	Threshold VAR	76
5.2.1	Tsay(1998) interest rates example	77
5.2.2	Balke(2000) Credit Regimes	80
5.2.3	Impulse Response Functions	86
5.3	Threshold Cointegration	94
5.1	Threshold Error Correction Model	97
5.2	Threshold Error Correction Model: Forecasting	99
5.3	Threshold VAR, Multiple Breaks	102
5.4	Threshold VAR, Testing and Estimation	104
5.5	Threshold VAR, Impulse Responses	108
6	STAR Models	114
6.1	Testing	115
6.2	Estimation.	117
6.3	Forecasts and Impulse Responses	118
6.4	More Complicated Models.	125
6.1	LSTAR Model: Testing and Estimation	126
6.2	LSTAR Model: Impulse Responses	127

7 Mixture Models	131
7.1 Maximum Likelihood	133
7.2 EM Estimation.	135
7.3 Bayesian MCMC	138
7.3.1 Label Switching	140
7.1 Mixture Model-Maximum Likelihood	142
7.2 Mixture Model-EM	143
7.3 Mixture Model-MCMC	145
 8 Markov Switching: Introduction	 148
8.1 Markov Switching Models.	148
8.2 Common Concepts	150
8.2.1 Prediction Step	151
8.2.2 Update Step	152
8.2.3 Smoothing	152
8.2.4 Simulation of Regimes	153
8.2.5 Pre-Sample Regime Probabilities	157
8.2.6 Pathologies	158
8.3 Estimation.	159
8.3.1 Simple Example	159
8.3.2 Maximum Likelihood	160
8.3.3 EM	161
8.3.4 MCMC (Gibbs Sampling)	165
8.1 Markov Switching Variances-ML	168
8.2 Markov Switching Variances-EM	169
8.3 Markov Switching Variances-MCMC	171
 9 Markov Switching Regressions	 176
9.1 MSREGRESSION procedures	176
9.2 The example	177
9.2.1 Maximum Likelihood	177
9.2.2 EM	180
9.2.3 MCMC (Gibbs Sampling)	181

9.1	MS Linear Regression: ML Estimation	186
9.2	MS Linear Regression: EM Estimation	187
9.3	MS Linear Regression: MCMC Estimation	188
10	Markov Switching Multivariate Regressions	193
10.1	@MSSYSREGRESSION procedures	193
10.1.1	Impulse Response Functions	194
10.2	The example	196
10.2.1	Maximum Likelihood	196
10.2.2	EM	199
10.2.3	MCMC (Gibbs Sampling)	200
10.3	Systems Regression with Fixed Coefficients.	204
10.1	MS Systems Regression: ML Estimation	207
10.2	MS Systems Regression: EM Estimation	208
10.3	MS Systems Regression: MCMC Estimation	209
11	Markov Switching VAR's	214
11.1	Estimation.	215
11.1.1	MSVARSETUP procedures	215
11.2	The example	217
11.2.1	Maximum Likelihood	217
11.2.2	EM	219
11.2.3	MCMC (Gibbs Sampling)	220
11.1	Hamilton Model: ML Estimation	224
11.2	Hamilton Model: EM Estimation	226
11.3	Hamilton Model: MCMC Estimation	227
12	Markov Switching State-Space Models	231
12.1	The Examples	232
12.2	The Kim Filter	233
12.2.1	Lam Model by Kim Filter	236
12.2.2	Time-Varying Parameters Model by Kim Filter	239
12.3	Estimation with MCMC.	241
12.3.1	Lam Model by MCMC	242

12.3.2 Time-varying parameters by MCMC	246
12.1 Lam GNP Model-Kim Filter	250
12.2 Time-Varying Parameters-Kim Filter	254
12.3 Lam GNP Model-MCMC	258
12.4 Time-Varying Parameters-MCMC	263
 13 Markov Switching ARCH and GARCH	 268
13.1 Markov Switching ARCH models	269
13.1.1 Estimation by ML	271
13.1.2 Estimation by MCMC	274
13.2 Markov Switching GARCH	280
13.2.1 The Example	283
13.2.2 Dueker Filter	284
13.2.3 Gray Filter	290
13.1 MS ARCH Model-Maximum Likelihood	295
13.2 MS ARCH Model-MCMC	297
13.3 MS GARCH Model-Approximate ML, Dueker filter	302
13.4 MS GARCH Model-Approximate ML, Gray filter	303
 A A General Result on Smoothing	 307
 B The EM Algorithm	 308
 C Hierarchical Priors	 310
 D Gibbs Sampling and Markov Chain Monte Carlo	 313
 E Time-Varying Transition Probabilities	 317
E.1 EM Algorithm	317
 F Probability Distributions	 319
F.1 Univariate Normal	319
F.2 Beta distribution	320
F.3 Dirichlet distribution	321
F.4 (Scaled) Inverse Chi-Squared Distribution	322

F.5	Gamma Distribution	323
F.6	Inverse Gamma Distribution	324
F.7	Bernoulli Distribution	325
F.8	Multivariate Normal	326
F.9	Wishart Distribution	327
F.10	Inverse Wishart Distribution	328
 G GNU Free Documentation License		330
1.	APPLICABILITY AND DEFINITIONS	330
2.	VERBATIM COPYING	332
3.	COPYING IN QUANTITY.	332
4.	MODIFICATIONS	333
5.	COMBINING DOCUMENTS	335
6.	COLLECTIONS OF DOCUMENTS	336
7.	AGGREGATION WITH INDEPENDENT WORKS	336
8.	TRANSLATION	336
9.	TERMINATION	337
10.	FUTURE REVISIONS OF THIS LICENSE	337
11.	RELICENSING	338
 Bibliography		339
 Index		341

Preface

This workbook is based upon the content of the RATS e-course on *Switching Models and Structural Breaks*, offered in fall of 2010. It covers a broad range of topics for models with various types of breaks or regime shifts.

In some cases, models with breaks are used as diagnostics for models with fixed coefficients. If the fixed coefficient model is adequate, we would expect to reject a similar model that allows for breaks, either in the coefficients or in the variances. For these uses, the model with the breaks isn't being put forward as a model of reality, but simply as an alternative for testing purposes. Chapters 2 and 3 provide several examples of these, with Chapter 2 looking at "fluctuation tests" and Chapter 3 examining parametric tests.

Increasingly, however, models with breaks are being put forward as a description of the process itself. There are two broad classes of such models: those with observable regimes and those with hidden regimes. Models with observable criteria for classifying regimes are covered in Chapters 4 (Threshold Autoregressions), 5 (Threshold VAR and Cointegration) and 6 (Smooth Threshold Models). In all these models, there is a threshold trigger which causes a shift of the process from one regime to another, typically when an observable series moves across an (unknown) boundary. There are often strong economic arguments for such models (generally based upon frictions such as transactions costs), which must be overcome before an action is taken. Threshold models are generally used as an alternative to fixed coefficient autoregressions and VAR's.

The remaining seven chapters cover models with hidden regimes, that is models where there is no *observable* criterion which determines to which regime a data point belongs. Instead, we have a model which describes the behavior of the observables in each regime, and a second model which describes the (unconditional) probabilities of the regimes, which we combine using Bayes rule to infer the posterior probability of the regimes. Chapter 7 starts off with the simple case of time independence of the regimes, while the remainder use the (more realistic) assumption of Markov switching. The sequence of chapters 8 to 11 look at increasingly complex models based upon linear regressions, from univariate, to systems, to VAR's with complicated restrictions. All of these demonstrate the three main methods for estimating these types of models: maximum likelihood, EM and Bayesian MCMC.

The final two chapters look at Markov switching in models where exact likelihoods can't be computed, requiring approximations to the likelihood. Chapter 12 examines state-space models with Markov switching, while Chapter 13 is devoted to switching ARCH and GARCH models.

The second edition adds over 100 pages, with new coverage of the ICSS test for variance breaks (Section 2.1), the “fixed regressor bootstrap” (Section 3.3), increased coverage of computation of non-linear impulse response functions in various threshold models (Sections 4.4, 5.2.3 and 6.3) and a completely rewritten section on Threshold VAR’s (5.2). We’ve reworked the various Markov switching support procedures, and the updated chapters on Markov Switching models and their examples have been revised to reflect that. In particular, Chapter 10 on Markov Switching Multivariate Regressions now has a (very) detailed description of the process of computing regime-specific impulse response functions with error bands.

Finally, the section on Markov Switching GARCH models (Section 13.2) has been completely rewritten to explain the difference between the more accurate “Dueker filter” and the more commonly used “Gray filter” with application to the same data set to demonstrate that. It’s also shown that each of the filters at times may fail (rather badly) to provide an adequate approximation to the log likelihood, producing possibly misleading results.

Style of this Workbook

We use bold-faced Courier (for instance, **DLM**) for any use of RATS instruction or procedure names within the main text, and non-bolded Courier %SCALAR) for any other pieces of code, such as function and variable names. For easy reference, the full text of each example is included. The running examples are also available as separate files.

Structural Breaks at Known Locations

1.1 Structural Breaks: Introduction

The term *structural break* is generally used to mean an (identifiable) change in the behavior of time series data (or more properly in a time series *model*) at a specific point in time, due perhaps to a change in law or policy. It's become increasingly common to let the location(s) of break(s) be determined by the data (an “endogenous break”) rather than being chosen by the researcher. A potential problem with an exogenously chosen break is that it can lead to misleading inference if there *is* a break, but it's at a different location for a different reason.

For instance, Figure 1.1 is of the data from Example 2.1, which are independent draws with a hard break in the mean after entry 100 (from $-.25$ for entries 1 to 100 to $+.25$ for entries 101 to 200).

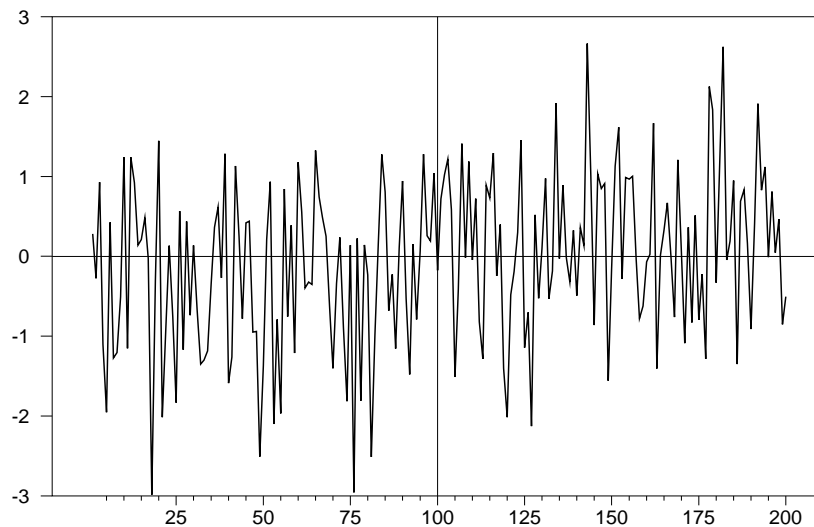


Figure 1.1: Data with Break in Mean

If we make a test at the *correct* time period:

```
set dummy = t>=101
linreg eps
# constant dummy
```

we get a solidly significant t -statistic of 3.56. However, the quite *incorrect* break point at 50 still gives a t -statistic of 2.33, which is significant at the .02 level as a two-tailed test and at the .04 level even as a one-tailed test, so if you hypothesized $t_0 = 50$ as the location of a break, you would reject constancy in favor of the break at your chosen date, despite the fact that that's *not* the actual break point. Instead, a test with an endogenous break point would probably find the break closer to 100, which would create strong doubt in your hypothesis.

Note, however, that there are many ways to misuse structural break analysis. In particular, it should not be taken as a good sign for a “model” to have breaks, particularly *many* breaks. If you run a multiple breakpoint analysis on a linear regression and find it has three breaks, then you probably have a poorly chosen regression (an important omitted variable or a neglected non-linearity could create that, for instance). The good sign is for a model to *not* show breaks—if it does have breaks, then your job should really just be starting (to determine why the simpler model fails).

Even though exogenously-determined structural breaks are much less commonly used than in the past, the basic building block for finding an unknown break point is the analysis with a *known* break, since we will generally need to estimate with different test values for the break point. We divide this into “static” and “dynamic” models, since the two have quite different properties.

1.2 Breaks in Static Models

We're talking here about models of the form $y_t = X_t\beta + u_t$, where X_t doesn't include lags of either y_t or u_t and u_t is serially uncorrelated. Eliminating models with time series dynamics makes it simpler to examine the effect of shifts.

Outliers

The simplest “break” in a model is a single-period outlier, say at t_0 . The conventional way to deal with a outlier at a known location is to “dummy out” the data point, by adding a dummy variable for that point only. This is equivalent to running weighted least squares with the variance at t_0 being ∞ . Since it's fairly rare that we're sure that a data point requires such extreme handling, we'll look at other, more flexible ways to handle this later in the course. We can create the required dummy with:

```
set dpoint = t==t0
```

Broken Trends

If X includes constant and trend, a broken trend function is frequently of interest. This can take one of several forms, as shown in Figure 1.2.

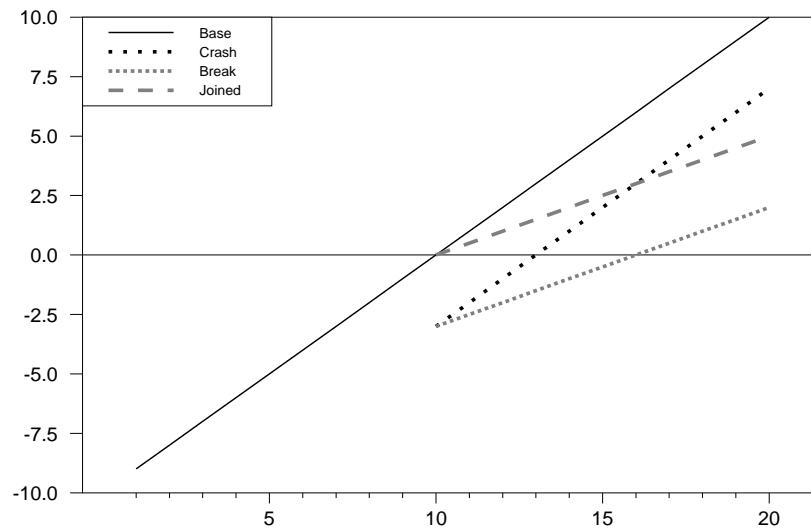


Figure 1.2: Broken Trends

If the break point is at the entry T_0 , the two building blocks for the three forms of break are created with:¹

```
set dlevel = t>t0
set dtrend = %max(t-t0, 0)
```

The *crash* model keeps the same trend rate, but has an immediate and permanent level shift (for variables like GDP, generally a shift down, hence the name). It is obtained if you add the `DLEVEL` alone. The *join* model has a change in the growth rate, but no immediate change in the level. You get it if you add `DTREND` alone. The *break* model has a change in both level and rate—in effect, you have two completely different trend functions before and after. You get this by adding *both* `DLEVEL` and `DTREND`.

Full Coefficient Breaks

If the entire coefficient vector is allowed to break, we have two equivalent ways to write the model:²

$$y_t = X_t\beta + X_tI_{t>t_0}\delta + u_t \quad (1.1)$$

$$y_t = X_tI_{t\leq t_0}\beta^{(1)} + X_tI_{t>t_0}\beta^{(2)} + u_t \quad (1.2)$$

The two are related with $\beta^{(1)} = \beta$ and $\beta^{(2)} = \beta + \delta$. Each of these has some uses for which it is most convenient.

If the residuals aren't serially correlated, (1.2) can be estimated by running separate regressions across the two subsamples. The following, for instance, is from Greene's 5th edition:

¹Standard practice is for structural breaks to apply one period *after* the quoted break date, that is, the earlier segment includes T_0 , while the later segment starts with T_0+1 .

²We'll use the notation $I_{condition}$ as a 1-0 indicator for the condition described.

```
linreg loggpop 1960:1 1973:1
# constant logypop logpg logpnc logpuc trend
compute rsspre=%rss
linreg loggpop 1974:01 1995:01
# constant logypop logpg logpnc logpuc trend
compute rsspost=%rss
```

If you don't need the coefficients themselves, an even more convenient way to do this is to use the **SWEEP** instruction (see page 9 for more). The calculations from Greene can be done with

```
sweep(group=(t<=1973:1),series=resv)
# loggpop
# constant logypop logpg logpnc logpuc trend
```

However, if you want to allow for HAC standard errors (like Newey-West), there's no choice but to estimate the full regression with dummied-out regressors since one sample can lag or lead into the other in computing the covariance matrix.³ If you have only a few regressors, generating the dummied-out regressors isn't that hard, but it can be tedious to do manually if there are many. The original program provided by Stock and Watson for the `ONEBREAK.RPF` example had 20 lines like this:

```
set dc = t > t3
set d0 = dc*fdd
set d1 = dc*fdd{1}
set d2 = dc*fdd{2}
set d3 = dc*fdd{3}
```

Our rewrite of this uses the `%EQNXVECTOR` function to generate the full list of dummied regressors (see page 10 for more). For a break at a specific time period (called `TIME`), this is done with

```
compute k=%nreg
dec vect[series] dummies(k)
*
do i=1,k
  set dummies(i) = %eqnxvector(baseeqn,t)(i)*(t>time)
end do i
```

The expression on the right side of the **SET** first extracts the time `T` vector of variables, takes element `I` out of it, and multiplies that by the relational `T>TIME`. The `DUMMIES` set of series is used in a regression with the original set of variables, and an exclusion restriction is tested on the dummies, to do a HAC test for a structural break:

³Eicker-White standard errors, to correct for heteroscedasticity alone, can be done using separate subsamples because they don't require lags or leads of the residuals.

```
linreg(noprint, lags=7, lwindow=newey) drpoj lower upper
# constant fdd{0 to 18} dummies
exclude(noprint)
# dummies
```

1.3 Breaks in Dynamic Models

This is a *much* more complicated situation, because the effect of various interventions is now felt in subsequent periods. That's clear from looking at the simple model

$$y_t = y_{t-1} + \alpha + \delta I_{t>t_0} + u_t \quad (1.3)$$

The δ , rather than shifting the mean of the process as it would in a static model, now shifts its trend rate.

There are two basic ways of incorporating shifts, which have been dubbed *Additive Outliers* (AO) and *Innovational Outliers* (IO), though they aren't always applied just to outlier handling. The simpler of the two to handle (at least in linear regression models) is the IO, in which shifts are done by adding the proper set of dummies to the regressor list. They are called “innovational” because they are equivalent to adjusting the mean of the u_t process. For instance, we could write (1.3)

$$y_t = y_{t-1} + u_t^*$$

where u_t^* has mean α for $t \leq t_0$ and mean $\alpha + \delta$ for $t > t_0$. The effect of an IO is usually felt gradually, as the change to the shock process works into the y process itself.

The AO are more realistic, but are more complicated—they directly shift the “mean” of the y process itself. The drifting random walk $y_t = y_{t-1} + \alpha + u_t$ will look something like the Base series in Figure 1.2 with added noise. How would we create the various types of interventions? The easiest way to look at this is to rewrite the process as $y_t = y_0 + \alpha t + z_t$, where $z_t = z_{t-1} + u_t$. This breaks the series down into the systematic trend and the “noise”.

The crash model needs the trend part to take the form $y_0 + \alpha t + \delta I_{t>t_0}$. We *could* just go ahead and estimate the parameters using

$$y_t = y_0 + \alpha t + \delta I_{t>t_0} + z_t$$

treating z_t as an unmodeled error process (which is done in Example 1.1 with a more complicated trend model). That would give consistent, but highly inefficient, estimates of α and δ with a Durbin-Watson hovering near zero. Instead, we can difference to eliminate the unit root in z_t , producing

$$y_t - y_{t-1} = \alpha(t - (t-1)) + \delta(I_{t>t_0} - I_{t-1>t_0}) + u_t$$

which reduces to

$$y_t - y_{t-1} = \alpha + \delta I_{t=(t_0+1)} + u_t$$

so to get the permanent shift in the process, we need a one-shot change at the intervention point.

The join model has a trend of

$$y_0 + \alpha t + \gamma \max(t - t_0, 0)$$

Differencing that produces $\alpha + \gamma I_{t>t_0}$. The full break model needs both intervention terms, so the trend is

$$y_0 + \alpha t + \delta I_{t>t_0} + \gamma \max(t - t_0, 0)$$

which differences to

$$\alpha + \delta I_{t=t_0+1} + \gamma I_{t>t_0}$$

so to handle both effects we need both the “spike” dummy and the level shift dummy.

Now suppose we have a *stationary* model (AR(1) for simplicity) where we want to incorporate a once-and-for-all change in the process mean. Let’s use the same technique of splitting the representation into mean and noise models:

$$y_t = \alpha + \delta I_{t>t_0} + z_t$$

where now $z_t = \rho z_{t-1} + u_t$. If we quasi-difference the equation to eliminate the z_t , we get

$$y_t - \rho y_{t-1} = \alpha(1 - \rho) + \delta(I_{t>t_0} - \rho I_{t-1>t_0}) + u_t$$

The δ term is no longer as simple as it was when we were first differencing. It’s $\delta(1 - \rho)$ for $t > t_0 + 1$ (terms which are zero when $\rho = 1$), and δ when $t = t_0 + 1$. There is no way to unravel these (plus the intercept) to give just two terms that we can use to estimate the model by (linear) least squares; in other words, unlike the IO, the AO interventions don’t translate into a model that can be estimated by simple techniques.

With constants and simple polynomial trends for the deterministic parts, there is no (effective) difference between estimating a “reduced form” stationary AR(p) process using a **LINREG**, and doing the same thing with the mean + noise arrangement used by **BOXJENK** (estimated with conditional least squares)—the AR parameters will match exactly, and the deterministic coefficients can map to each other. That works because the filtered versions of the polynomials in t are still just polynomials of the same order. Once you put in any intervention terms, this is no longer the case—the filtered intervention terms generally produce two (or more) terms that aren’t already included. **BOXJENK** is designed to do the AO style of intervention, so if you need to estimate this type of intervention model, it’s the instruction to use. We do that in Example 1.1.

Example 1.1 looks at a broken trend model for annual data for log U.S. GNP over the period from 1909 to 1970. It does an AO handling (thus the process mean changes) with joined trend breaks in 1929 and 1932. The base trend and the two dummied trends are created with:

Table 1.1: Broken Trend by Linear Regression

Linear Regression - Estimation by Least Squares					
Dependent Variable LOGGNP					
Annual Data From 1909:01 To 1970:01					
Sum of Squared Residuals	4852.3483				
Log Likelihood	-223.1368				
Durbin-Watson Statistic	0.4054				
Variable	Coeff	Std Error	T-Stat	Signif	
1. Constant	470.3119	4.0937	114.8862	0.0000	
2. TREND	2.6268	0.3190	8.2357	0.0000	
3. DTREND1	-7.0961	1.8151	-3.9094	0.0002	
4. DTREND2	8.4875	1.6550	5.1283	0.0000	

```

set trend      = t
set dtrend1    = %max(t-1929:1, 0)
set dtrend2    = %max(t-1932:1, 0)

```

This estimates the model using **LINREG** (Table 1.1, with some summary statistics deleted) and computes the estimated trend function into **LINTREND**.

```

linreg loggnp
# constant trend dtrend1 dtrend2
prj lintrend

```

Because the trends are cumulative, the trend rate is estimated to be 2.62% in 1909-1929, 2.62-7.10 (roughly -4.47%) from 1930-1932 and 2.62-7.10+8.49 (4.02%) from 1933 on.⁴

While this estimate may be adequate for many purposes, the Durbin-Watson is really low, suggesting that a better model might be a trend with a random walk “noise” model. As described above, that can be done by differencing the trend model and running a linear regression on the differenced regressors, but it’s simpler to use **BOXJENK** with the **GLS** option (Table 1.2), as that is what it’s designed to do.

```

boxjenk(diffs=1,glsl,meaneq=bjtredeq) loggnp
# trend dtrend1 dtrend2

```

Note that there is no **CONSTANT** in this—the intercept in the trend model is eliminated in applying the differences.⁵

⁴If you want, you can use **SUMMARIZE** to compute those last two with estimated standard errors. There’s no simple way to parameterize a joined trend like this (or the related but more complicated splines) which allows you to read off separate estimates from each subsample because the joining creates a restriction between segments.

⁵You *could* include **CONSTANT** in the list, but it will have a zero coefficient with zero standard error which means that it doesn’t affect the results.

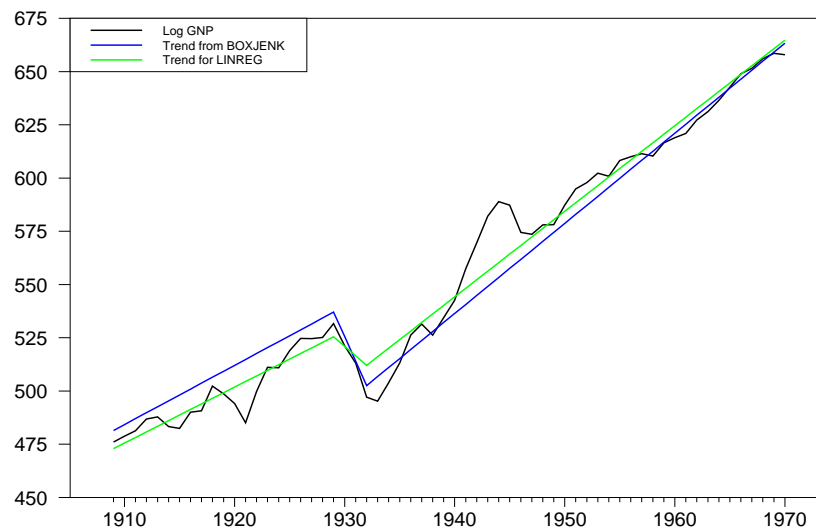
Table 1.2: Broken Trend by **BOXJENK**

Box-Jenkins - Estimation by ML Gauss-Newton				
Dependent Variable LOGGNP				
Annual Data From 1910:01 To 1970:01				
Sum of Squared Residuals	1816.7637			
Log Likelihood	-190.0704			
Durbin-Watson Statistic	1.5539			
Q(15-0)	20.2093			
Significance Level of Q	0.1640			
Variable	Coeff	Std Error	T-Stat	Signif
1. TREND	2.7785	1.2515	2.2202	0.0303
2. DTREND1	-14.2770	3.4652	-4.1202	0.0001
3. DTREND2	15.7302	3.3564	4.6866	0.0000

Because differencing causes the trend to lose its level, this computes the variable part of the trend and uses the mean deviation of that to re-center the trend.

```
set bjtrend = %eqnprj(bjtrendeq,t)
sstats(mean) / loggnp-bjtrend>>bjmean
set bjtrend = bjtrend+bjmean
```

The two estimates of the trend are shown in Figure 1.3. Note that the **BOXJENK** estimates a more severe decline in the growth rate at the depths of the Depression than the simple regression.

**Figure 1.3:** Estimates of Broken Trends

The equivalent code for differencing the model algebraically is

```
diff loggnp / dy
set ddtrend1 = t>1929:1
set ddtrend2 = t>1932:1
linreg dy
# constant ddtrend1 ddtrend2
```

which gives identical results to Table 1.2.

Finally, the example does a residual analysis, which finds that an ARIMA(0,1,1) model is probably the most parsimonious fit to the error process. That's re-estimated using **BOXJENK** by adding an **MA=1** option to the earlier instruction:

```
@regcorrs
*
boxjenk(diffs=1,glsl,ma=1) loggnp
# trend dtrend1 dtrend2
```

1.4 RATS Tips and Tricks

The SWEEP Instruction

SWEEP is a very handy instruction when you need to do a set of linear regressions which take the same set of regressors, whether the set of regressions are different samples, or different dependent variables or both. For analyzing sample splits, the key option is **GROUP**, which gives a formula which takes distinct values for the sets of entries which define the sample splits. In the example in this chapter:

```
sweep(group=(t<=1973:1),series=resv)
# loggpop
# constant logypop logpg logpnc logpuc trend
```

the **GROUP** expression is 1 for $t \leq 1973:1$ and 0 afterwards. A three-way split could be done with something like $\text{GROUP} = (t \leq 1973:1) + (t \leq 1991:4)$, which will have value 2 for time periods through 1973:1 (since both inequalities are true there), 1 for 1973:2 to 1991:4 and 0 afterwards.

The instruction above will estimate the regressions separately over the two samples, and provide the combined residuals (in the **VECTOR[SERIES]** named **RESV**⁶), the 1×1 covariance matrix of the residuals in **%SIGMA**, and the variables **%NOBS**, **%NREG** and **%NREGSYSTEM**, where **%NREG** is the number of regressors in each regression (here 6) and **%NREGSYSTEM** is the number across groups (and targets), in this case, 12. These provide all the information needed for the split sample part of a Chow test, since the combined sum of squared residuals will just be $\%NOBS * \%SIGMA(1,1)$, and the degrees of freedom will

⁶It's a **VECT[SERIES]** because there could be more than one target.

be `%NOBS-%NREGSYSTEM`. If you want to compute a likelihood ratio instead, it also produces the variable `%LOGL` as the (Gaussian) log likelihood and `%NFREE` as the full count of estimated parameters including any variance parameters. For instance, here `%NFREE` would be 13 (12 regressions plus the one common variance).

An additional option which will be useful in structural breaks analysis is `VARIANCES=[HOMOGENEOUS]/HETEROGENEOUS`. In the default of `VARIANCES=HOMOGENEOUS`, the same variance is assumed for each subsample, while with `VARIANCES=HETEROGENEOUS` separate variances are estimated. Note that this only affects the log likelihood (and free parameter count) as the coefficient estimates in the subsamples don't depend upon the variances as long as the variance is constant *within* each subsample.

%EQNXVECTOR function

`%EQNXVECTOR(eqn, t)` returns the `VECTOR` of explanatory variables for equation `eqn` at entry `t`. You'll see this used quite a bit in the various procedures used for break analysis since it's often necessary to look at an isolated data point. An `eqn` of 0 can be used to mean the last regression run.

In this chapter, this is used in the following:

```
compute k=%nreg
dec vect[series] dummies(k)
*
do i=1,k
    set dummies(i) = %eqnxvector(baseeqn,t)(i)*(t>time)
end do i
```

The expression on the right side of the **SET** first extracts the time `T` vector of variables, takes element `I` out of it, and multiplies that by the relational `T>TIME`. The `DUMMIES` set of series is used in a regression with the original set of variables, and an exclusion restriction is tested on the dummies, to do a HAC test for a structural break:

There are several related functions which can also be handy. In all cases, `eqn` is either an equation name, or 0 for the last estimated (linear) regression. These also evaluate at a specific entry `T`.

- `%EQNPRJ(eqn, t)` evaluates the fitted value $X_t\beta$ for the current set of coefficients for the equation.
- `%EQNVALUE(eqn, t, beta)` evaluates $X_t\beta$ for an input set of coefficients.
- `%EQNRESID(eqn, t)` evaluates the residual $y_t - X_t\beta$ for the current set of coefficients, where y_t is the dependent variable of the equation.
- `%EQNRVALUE(eqn, t)` evaluates the residual $y_t - X_t\beta$ for an input set of coefficients.

Example 1.1 Models with Exogenously-determined Trend Breaks

This estimates a broken-trend model for log U.S. GNP with joined trend breaks in at 1929 and 1932. The trend rate of growth through 1929 is thus allowed to be different than the rate of growth after 1932, with the deepest part of the Depression handled using a separate time segment.

```
open data nelsonplosser.rat
calendar(a) 1909:1
data(format=rats) 1909:01 1970:01 realgnp
set loggnp = 100.0*log(realgnp)
*
graph
# loggnp
*
* We'll allow for joined trends at 1929 and 1932.
*
set trend = t
set dtrend1 = %max(t-1929:1,0)
set dtrend2 = %max(t-1932:1,0)
*
* Estimating with LINREG
*
linreg loggnp
# constant trend dtrend1 dtrend2
prj lintrend
*
* Estimating with BOXJENK (with no ARMA parameters). The constant washes
* out of this.
*
boxjenk(diffs=1,glis,meaneq=bjtrendeql) loggnp
# trend dtrend1 dtrend2
*
* The trend is missing its level, so this computes the variable part of
* the trend and uses the mean deviation of that to center the trend.
*
set bjtrend = %eqnprj(bjtrendeql,t)
sstats(mean) / loggnp-bjtrend>>bjmean
set bjtrend = bjtrend+bjmean
*
graph(key=upleft,klabels=$
  ||"Log GNP","Trend from BOXJENK","Trend for LINREG"||) 3
# loggnp
# bjtrend
# lintrend
*
* Equivalent model done using algebraically differenced equation
*
diff loggnp / dy
set ddtrend1 = t>1929:1
set ddtrend2 = t>1932:1
```

```
linreg dy
# constant ddtrend1 ddtrend2
*
@regcorrs
*
* Based upon the residual autocorrelations, it looks as if an MA(1)
* noise model is appropriate.
*
boxjenk(diffs=1, gls, ma=1) loggnp
# trend dtrend1 dtrend2
```

Fluctuation Tests

A Brownian Bridge (or tied-down Brownian Motion) is a variation of Brownian Motion which starts at 0 at $t = 0$, ends at 0 at $t = 1$ and fluctuates in between. If $W(x)$ is a Brownian Motion, the Brownian Bridge (BB for short) is $W(x) - xW(1)$. It's the limit process for the (centered) partial sums

$$B\left(\frac{t}{T}\right) = \frac{1}{\sqrt{T}} \left(\sum_{s=1}^t (\varepsilon_s - \bar{\varepsilon}) \right) \quad (2.1)$$

for i.i.d. $N(0, 1)$ process ε where $\bar{\varepsilon}$ is the average across the full set of T observations. The Brownian Bridge has quite a few uses in statistics; for instance, the (scaled) distance between a (true) distribution function and an empirical distribution function for an i.i.d. sample from it forms a BB. The maximum vertical distance between the two is the Kolmogorov-Smirnov statistic—in the K-S test, we reject that the sample came from the hypothesized distribution if the K-S statistic is seen to be too large to represent the maximum absolute value of a BB.

The usefulness in testing for structural breaks can be seen from looking at (2.1). This, rather clearly, has the same distribution if ε has a (fixed) non-zero mean μ . Suppose that instead of a fixed mean, ε has one mean in the early part of the sample, and another, higher, mean later. Then, with removal of a full-sample mean, most of the early values of $(\varepsilon_s - \bar{\varepsilon})$ would be negative, and most of the later ones would be positive. We would thus expect to see the BB process drift well into negative territory for the first part of the sample, then eventually climb back to the forced zero at the end. Figure 2.1 shows an example of the sample values of (2.1) generated from 200 data points, with data drawn from $N(-.25, 1)$ for the first 100 data points and $N(.25, 1)$ for the last 100. (The actual data are graphed in Figure 1.1).

We would expect something similar to this even if the alternative weren't so sharply defined—if the mean of the process were generally lower at the start than at the end, you would still expect to see this type of behavior. If we allow for the more general assumption that the true DGP is i.i.d. (not necessarily Normal) with unknown mean μ and variance σ^2 , then the process

$$B\left(\frac{t}{T}\right) = T^{-1/2}(\hat{\sigma})^{-1/2} \left(\sum_{s=1}^t (\varepsilon_s - \bar{\varepsilon}) \right) \quad (2.2)$$

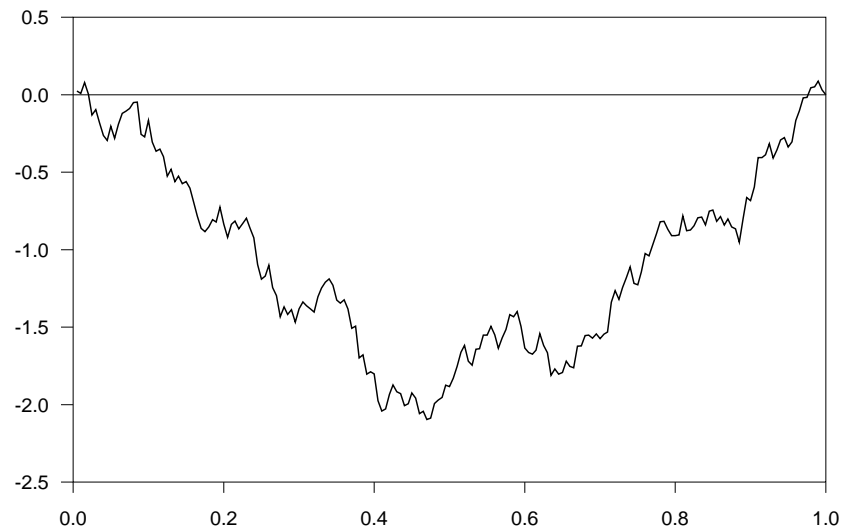


Figure 2.1: Fluctuations with Break in Level

also converges to a Brownian Bridge where $\hat{\sigma}$ is any conventional estimate of the sample standard deviation. Here, we can see the possibility of a test for a break in the *variance*: if instead of being fixed, the variance is systematically lower at the start than at the end, the divisor will be too big at the start, and too small at the end, so the fluctuations will be too small at the start for a standard BB and too large at the end. Figure 2.2 is an example of (2.1) generated using 200 Normals, with mean 0 and standard deviation .5 for the first 100 and mean 0 and standard deviation 1.5 for the second 100.

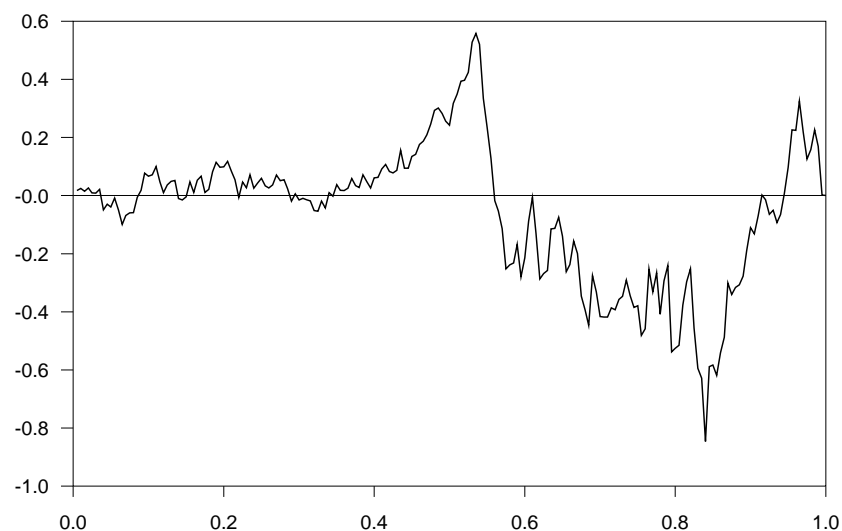


Figure 2.2: Fluctuations with Break in Variance

So we have a description of the behavior of partial sums of a fairly general process under the assumption that there is a single DGP valid across the sample, and we can see that for several important types of breaks in the structure (of unknown form), that they will behave differently. The question then becomes,

what functional(s) of these will be useful for detecting instability in the process. The K-S maximum gap is one possibility, which would likely work for the break in level, but wouldn't work well for the break in variance. The proposal from Nyblom (1989) is to use the sample equivalent of

$$\int_0^1 B(x)^2 dx \quad (2.3)$$

This should be quite good at detecting instability in the mean, since $B(x)^2$ will be unnaturally large due to the drift in $B(x)$. It's not as useful for detecting breaks in the variance (in this form), since the higher and lower zones will tend to roughly cancel—a function of higher powers than 2 will be required for that.

The same basic idea can be applied to problems much broader than an i.i.d. univariate process. Suppose that

$$\sum_{t=1}^T f(y_t | X_t, \theta)$$

is the log likelihood for y given X and the parameters θ . Then, for the maximizing value $\hat{\theta}$, the sequence of derivatives

$$g_t \equiv \frac{\partial f(y_t | X_t, \theta)}{\partial \theta} (\hat{\theta})$$

has to sum to zero over the sample for each component. Under standard regularity conditions, and with the assumption that the underlying model is stable, we have the result that

$$\frac{1}{\sqrt{T}} \sum_{t=1}^T g_t \xrightarrow{d} N(0, \mathcal{I}^{-1})$$

where \mathcal{I} is $1/T$ times the information matrix. For purposes of deriving a Brownian motion, the sample g_t can be treated as if they were individually mean zero and variance \mathcal{I}^{-1} . This is the principal result of the Nyblom paper. The individual components of the gradient can be tested as above, and you can also apply a joint test, as the whole vector of gradients becomes a multi-dimensional Brownian Bridge with the indicated covariance matrix. If we have an estimate of $V \equiv \mathcal{I}^{-1}$, the fluctuation statistic for component i of the gradient is computed as

$$T^{-2} v_{ii}^{-1} \sum_{t=1}^T \left(\sum_{s=1}^t g_{s,i} \right)^2$$

and the joint test is

$$T^{-2} \sum_{t=1}^T \left\{ \left(\sum_{s=1}^t g_s \right) V^{-1} \left(\sum_{s=1}^t g_s \right)' \right\} \quad (2.4)$$

The trickiest part of working out these formulas often is getting the power of T correct. If we look at the univariate case, the Brownian Bridge construction is

$$B\left(\frac{t}{T}\right) = T^{-1/2}v_{ii}^{-1/2} \left(\sum_{s=1}^t g_{s,i} \right) \quad (2.5)$$

We need a set of Riemann sums to approximate (2.3). When we square B , we get a multiplier of $T^{-1}v_{ii}^{-1}$. With T data points, the interval length is $1/T$; since we have to multiply the sum by the interval length to get the discrete integral, our final scale factor is $T^{-2}v_{ii}^{-1}$.

We now have formulas for computing the test statistics, so we need to know what the critical values are for (2.3) and its multivariate extensions. Functionals of Brownian Motion almost always have highly non-standard distributions, and the usual procedure is to approximate critical values by simulation. This one happens to be simple enough that, while there is no straightforward way to approximate the entire distribution analytically, you *can* get fairly accurate tail-probabilities, which is all that really matters for hypothesis testing.¹ In RATS, this can be done with the function `%NYBLOMTEST(x,p)`. This returns an approximate significance level for test statistic x computed with p components. The test statistic for the example shown in Figure 2.1 is 1.51680. `%NYBLOMTEST(1.51680,1)` returns .0002, so this is clearly way out in the tails. The test statistic for the example in Figure 2.2 is 0.06018, which has a significance level of .8029.

To do this type of fluctuations test, you can apply the RATS procedure `@FLUX` to the `VECTOR[SERIES]` returned by the `DERIVES` option on any of the instructions `MAXIMIZE`, `GARCH`, `NLLS`, `NLSYSTEM` and `BOXJENK`. For instance, the following does a stability test on a `GARCH(1,1)` model (from Example 2.2)

```
garch(p=1,q=1,derives=dd) / dlogdm
@flux
# dd
```

The results are in Table 2.1. This gives a joint test for all four coefficients, and separate tests on each coefficient. It would be surprising for a `GARCH` model to give us a problem with the coefficients of the *variance* model (here coefficients 2-4) simply because if there's a failure, it isn't likely to be systematic in the time direction. The one coefficient that seems to have a stability problem is the process mean, which is much less surprising.

You can check deeper into a parameter that shows a problem by accumulating its gradient series. Here, the one red flag is on parameter 1 (the process mean), so

¹This uses a *saddlepoint approximation*, which, in effect, expands the distribution function at ∞ , so it's accurate in the tails, but not as accurate closer to 0.

Table 2.1: Fluctuation Statistics for GARCH Model

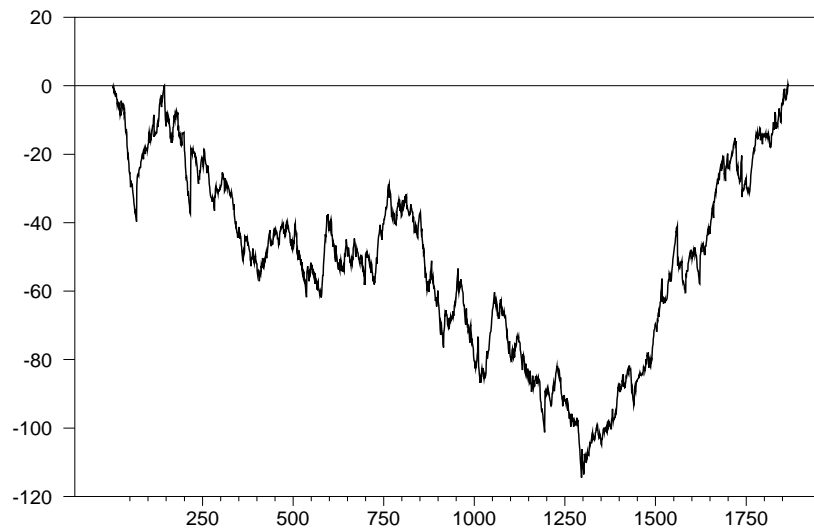
Joint	1.25426531	0.05
1	0.71267640	0.01
2	0.35762640	0.09
3	0.13834512	0.41
4	0.18261092	0.29

```

set cdd = dd(1)
acc cdd
graph(footer="Accumulated Gradients on mean in GARCH(1,1)")
# cdd

```

which creates Figure 2.3. The shape of this is what matters, not the scale,² and the shape indicates a likely break in the parameter around entry 1300. The largely downward trend in the cumulated gradient prior to 1300 would tend to indicate that the mean should be lower there, and the rather sharply upward trend after it shows that the mean should be higher there.

**Figure 2.3:** Accumulated Gradients on mean in GARCH(1,1)

The `@FLUX` procedure uses the BHHH estimate for the information matrix, since that can be computed from the gradients. The calculations themselves are relatively simple:

```

cmom(smpl=smpl1,equation=fluxeq) start1 end1
compute vinv=inv(%nobs*%cmom)
compute n=%ncmom
*
dim %hstats(n) s(n)
compute %hjoint=0.0,%hstats=%const(0.0),s=%const(0.0)

```

²This graphs the sum in parentheses in (2.5) without the scaling factors.

```

do time=start1,end1
  if .not.smp11(time)
    next
  compute x=%eqnxvector(fluxeq,time)
  compute s=s+x
  ewise %hstats(i)=%hstats(i)+s(i)**2
  compute %hjoint=%hjoint+%qform(vinv,s)
end do time
ewise %hstats(i)=%hstats(i)/(%nobs*%cmom(i,i))

```

The **CMOM** instruction computes the cross product matrix of the gradients, which will be the estimate for the sample information matrix. The calculation for **VINV** gets the factors of T correct for computing the joint statistic (2.4). The vector **S** is used to hold the partial sums of the entire vector of gradients. Through the time loop, **%HSTATS** has the individual component sums of squares for the partial sums; that gets scaled at the end. (We can't use elements of **VINV** for those, since that's from inverting the joint matrix, and we need the inverse just of the diagonal elements). **%HJOINT** takes care of the running sum for the joint test.

Independently (and apparently unaware) of Nyblom, Hansen (1992) derived similar statistics for partial sums of moments for a linear regression. As with the gradients for the likelihood, we have (under certain regularity conditions), the result that, for the linear regression $y_t = X_t\beta + u_t$

$$\frac{1}{\sqrt{T}} \sum_{t=1}^T X'_t u_t \xrightarrow{d} N(0, E(X' u^2 X))$$

The least squares solution for $\hat{\beta}$ forces the sum of the sample moments to zero:

$$\sum_{t=1}^T X'_t \hat{u}_t = 0$$

so again, properly scaled, we can generate a (multivariate) Brownian Bridge process. Hansen also shows that you can test for stability in the variance by using partial sums of $\hat{u}_t^2 - \hat{\sigma}^2$ —if you use the maximum likelihood estimate of $\hat{\sigma}^2$ (T divisor rather than $T - k$), that sums to zero over the sample. He uses an empirical estimate of the variance of $\hat{u}_t^2 - \hat{\sigma}^2$ to avoid having to make too many assumptions about the underlying distribution. Hansen's test can be done in RATS using the procedure **@STABTEST**. That's one of the test methods included in the example file **CONSTANT.RPF**, where it's done with

```

@stabtest y 1959:1 1971:3
# constant x2 x3

```

Table 2.2: Hansen Stability Test from CONSTANT.RPF

Hansen Stability Test		
Test	Statistic	P-Value
Joint	4.06952983	0.00
Variance	0.12107450	0.47
Constant	0.92208827	0.00
X2	0.91741025	0.00
X3	0.91379801	0.00

Basically, you just use the same setup as a **LINREG**, but with **@STABTEST** as the command instead. The results for that are in Table 2.2, which shows an overwhelming rejection of stability.³

Since estimating the mean is just regression on a constant, we can get a better test for stability of the variance in the initial example. This is done with

```
@stabtest eps
# constant
```

The results are in Table 2.3. As we would expect, that doesn't find instability in the mean (which was zero throughout), but overwhelmingly rejects stability in the variance.

Table 2.3: Hansen Stability Test for Breaking Variance Example

Hansen Stability Test		
Test	Statistic	P-Value
Joint	2.76585353	0.00
Variance	2.70348328	0.00
Constant	0.17120168	0.32

2.1 ICSS Test for Variance Changes

Another test procedure based upon Brownian Bridge fluctuations is the ICSS (Iterative Cumulative Sums of Squares) algorithm from Inlan & Tiao (1994). In their case, the BB is formed by

$$D_t = \frac{C_t}{C_T} - \frac{t}{T} \quad (2.6)$$

where

$$C_t = \sum_{s=1}^t \varepsilon_s^2 \quad (2.7)$$

is the cumulated sum of squares. This is (effectively) the same BB as is used in the variance stability test in Hansen (1992). The differences between the two procedures are:

³which is corroborated by the other tests in the example.

1. Hansen uses a sample estimate of the variance of the summands to normalize the Brownian motion, while Inclan and Tiao assume a constant variance (they assume i.i.d. ε).
2. Hansen uses the integral of the square as the functional of the BB, while Inclan and Tiao use the maximum absolute deviation.

Because Inclan and Tiao use the maximum deviation, the procedure identifies a particular entry as the location which produces the biggest break.⁴ The “iterated” part of ICSS is that a sequence of tests is used to identify (potentially) multiple locations of apparent changes in variance, that is, the main purpose of the article is not to show a test for constant variance, but to use that test to provide a (feasible) method for determining variance change points. If there are J variance breaks at unknown locations, then there are $O(T^J)$ possible combinations, which quickly becomes infeasible for an exhaustive search. For instance, in the example in the Inclan and Tiao paper with 369 data points, the number of possible sets of 5 break points is around 5×10^{10} . The iterative procedure both locates apparent change points and determines whether they are statistically significant. The “feasible” part of this is that once it decides upon the number of breaks, it runs through the break points, choosing a new T_j as the best break point between T_{j-1} and T_{j+1} taking the neighbors as given, stopping the refinement of the breaks once the change from one iteration to the next is minimal. There’s no guarantee that this will give precisely the same set of breaks as an exhaustive search, but it should be close.

This procedure assumes that the series has a common fixed mean across the data set—it should either be a series for which a fixed mean is a reasonable assumption, or it should be residuals from a preliminary set of estimates which eliminates any structural breaks in the *mean* process.

ICSS can be seen as a direct competitor to a GARCH model—in fact, the empirical example in the paper is daily log returns of IBM stock price, which is the type of series that would generally be the subject of some type of GARCH model. If the true DGP is just a broken variance process (such as the second case in Example 2.1), then the standard LM test for ARCH/GARCH (**@ARCHTEST**) will probably reject homoscedasticity in favor of ARCH, and GARCH model estimates will show fairly typical GARCH coefficients. This is because the volatility is, in fact, “clustered”, just not in the way that GARCH envisions. Conversely, if the true DGP is (a fairly persistent) GARCH process, then ICSS will likely locate several variance breaks.

If this is the case, how does one choose? You really can’t directly use the log likelihood or information criteria derived from them to compare the two estimates because ICSS is doing a nearly exhaustive search for high variance locations. If

⁴The article shows how the maximum deviation in the BB relates to the F test for variance break point.

you think about how a GARCH process works, a high variance segment will typically be triggered by an outlier or at least several large residuals over a short time period. However, the GARCH variance calculation only reacts to the outlier *with a lag*, so the outlier itself will have a low likelihood. ICSS will almost certainly put the variance break *before* the outlier, so to the variance break model, the outlier will *not be* a major surprise, thus biasing the log likelihood in favor on the variance break model. That, of course, is one of the major problems with endogenous breakpoint analysis—it doesn’t really provide a method to predict out-of-sample since the “model” is picked solely on the basis of the observed data and you don’t know what regime will be in place into the future.

Perhaps the worst of all possible approaches is to do an ICSS analysis and add dummies generated based upon them as variance shifts in a GARCH model. Structural breaks are breaks in a model, not breaks in a series. The “model” in ICSS has the variance constant over segments and the breaks are chosen based upon that. That’s very different from how the GARCH model looks at volatility and you can’t transfer information from one to the other—the variance dummies would generally be expected to be strongly significant in the combined model by conventional tests, but because they were chosen by search the conventional significance levels will be wrong (probably *very* wrong). It would also be expected to considerably reduce any GARCH-related effects. For instance, if you transfer univariate ICSS break information to a multivariate GARCH model, you will probably make it impossible to identify GARCH-related cross-effects like spillovers because the ICSS shift dummies will simply say that variance is high or low in a set of entries because...that’s what it’s observed to be.

Example 2.3 applies the ICSS procedure to the data set used in Example 2.2. In that example, we found an apparent break in the mean at roughly entry 1300, so we will include a mean shift dummy for the entries after that. First, we will re-estimate the GARCH model with the mean dummy included. This also include the `LIKELY` option (which saves the series of log-likelihood elements) which we’ll discuss shortly..

```
set meanshift = t>=1300
*
garch(reg,hseries=hh,derives=dd,likely=llgarch) / dlogdm
# constant meanshift
```

Not surprisingly (given the previous example) the mean shift dummy is quite significant (Table 2.4).

Because we’re now interested in the variance (not the mean), Figure 2.4 looks at the fluctuation BB on the variance constant in the GARCH model, generated with

Table 2.4: GARCH model with mean shift dummy

GARCH Model - Estimation by BFGS					
Convergence in 19 Iterations. Final criterion was 0.0000046 <= 0.0000100					
Dependent Variable DLOGDM					
Usable Observations		1866			
Log Likelihood		-2059.6016			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	-0.0527	0.0173	-3.0500	0.0023
2.	MEANSHIFT	0.1544	0.0364	4.2374	0.0000
3.	C	0.0166	0.0048	3.4661	0.0005
4.	A	0.1103	0.0159	6.9574	0.0000
5.	B	0.8672	0.0182	47.7490	0.0000

```

set cdd = dd(3)
acc cdd
graph(footer="Fluctuations on Variance Constant in GARCH")
# cdd

```

**Figure 2.4:** Fluctuation Statistic on Variance Constant

(With the added mean-model regressor, the variance constant is the 3rd coefficient). The Nyblom statistic on this is not statistically significant at conventional levels (the significance level is .10), but the two fairly rapid moves between 100 and 200 and between 800 and 1000 may indicate some possible problems.⁵

The example then does an ICSS analysis (Table 2.5) on mean-corrected residuals. This saves a `RECT[INTEGER]` with the subsample ranges identified by ICSS.

⁵The Nyblom test has much better power against single breaks than multiple breaks.

Table 2.5: ICSS Analysis of Residuals

ICSS Analysis of Series EPS
Using significance level 0.05
Breakpoints found at entries
50
93
214
466
814
1009
1296
1458

```

linreg dlogdm
# constant meanshift
*
set eps = %resids
*
@icss(subsamples=ss) eps

```

The easiest way to estimate the model with the variance breaks detected by ICSS is to use **GARCH** with no GARCH parameters ($P=0, Q=0$ options) using the **XREG** option to put the variance shift dummies in. ICSS found eight breaks, hence nine subsamples, but we only need dummies for eight of those (since there's already a overall variance constant). This leaves off the final subsample dummy:

```

dec vect[series] xshifts(%cols(ss)-1)
do i=1,%size(xshifts)
    set xshifts(i) = t>=ss(1,i).and.t<=ss(2,i)
end do i
*
garch(p=0,q=0,reg,xreg,hseries=hhicss,likely=llicss) / dlogdm
# constant meanshift
# xshifts

```

The results are in Table 2.6. As mentioned above, the substantial improvement in the log likelihood versus the simple GARCH model is misleading because the ICSS search procedure is using a search across the data range. We can compare the estimated variances (Figure 2.5) with

```

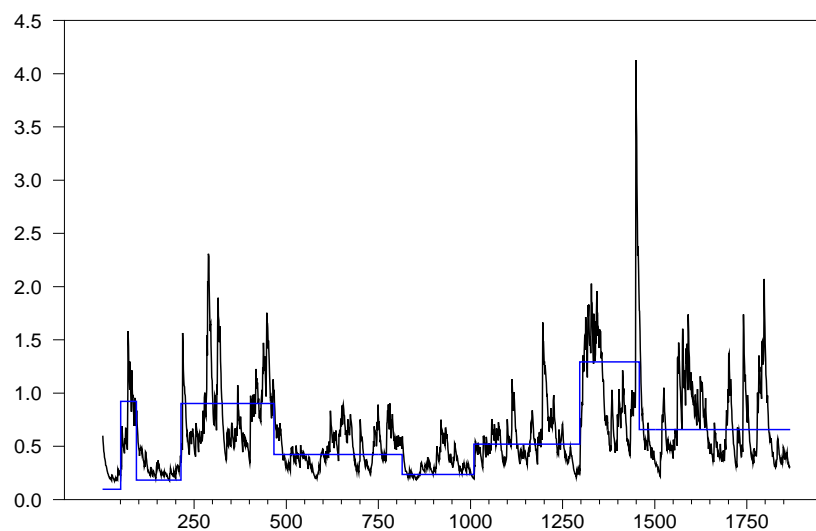
graph(footer="Volatility Estimates from GARCH and ICSS", $
    overlay=step,ovsamescale) 2
# hh
# hhicss

```

One thing that sticks out (literally) about the GARCH volatility estimates is the apparent overreaction at entry 1449 after a 5 standard deviation outlier at

Table 2.6: Variance Estimates with ICSS-Generated Dummies

GARCH Model - Estimation by BFGS					
Convergence in 58 Iterations. Final criterion was 0.0000098 <= 0.0000100					
Dependent Variable DLOGDM					
Usable Observations		1866			
Log Likelihood		-2024.7699			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	-0.0605	0.0156	-3.8783	0.0001
2.	MEANSHIFT	0.1656	0.0371	4.4595	0.0000
3.	C	0.6573	0.0446	14.7370	0.0000
4.	XSHIFTS(1)	-0.5610	0.0487	-11.5230	0.0000
5.	XSHIFTS(2)	0.2655	0.1847	1.4377	0.1505
6.	XSHIFTS(3)	-0.4735	0.0493	-9.6078	0.0000
7.	XSHIFTS(4)	0.2448	0.0894	2.7389	0.0062
8.	XSHIFTS(5)	-0.2334	0.0539	-4.3270	0.0000
9.	XSHIFTS(6)	-0.4205	0.0507	-8.2989	0.0000
10.	XSHIFTS(7)	-0.1366	0.0611	-2.2359	0.0254
11.	XSHIFTS(8)	0.6349	0.1420	4.4708	0.0000

**Figure 2.5:** Volatility Estimates from GARCH and ICSS

1448 (we don't graph the residuals here). These models were estimated using a Normal, and a t instead would reduce the log likelihood hit at the actual outlier (for *both* sets of estimates—it's about a 4.5 standard deviation outlier in the ICSS model), but won't really change the subsequent spike in the GARCH volatility estimates.⁶

Another graph which will be helpful in examining the difference between these two models uses the cumulated log likelihoods of the models. The `LIKELY` option on `GARCH` saves the log likelihood value for each entry, so the cumulated value will be the sample log likelihood through a particular entry. The gap (Figure 2.6) between the log likelihoods of the two models will help us see where the “look-ahead” estimates of ICSS help it the most.

```
acc llgarch / cumllgarch
acc llicss / cumllicss
set gap = cumllicss-cumllgarch
graph(footer="Log Likelihood Gap Between ICSS and GARCH")
# gap
```

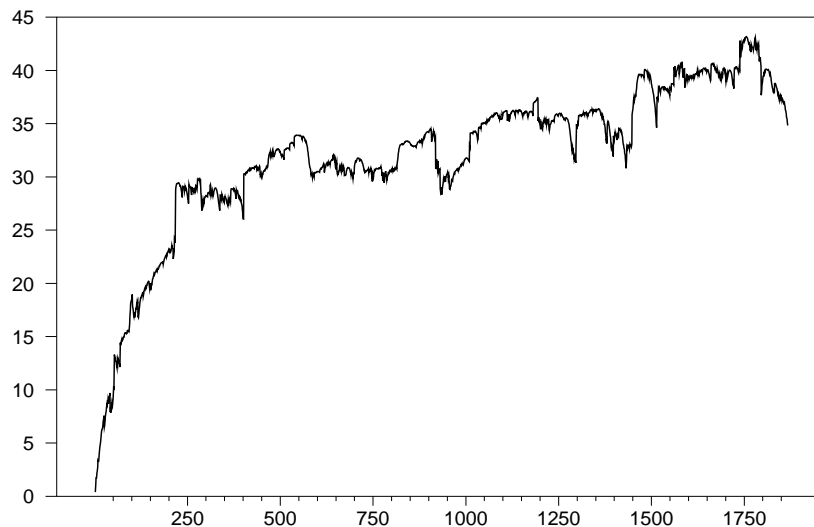


Figure 2.6: Log Likelihood Gap between ICSS and GARCH

What we can see is that virtually the entire difference between the two log likelihoods is in roughly the first 225 data points. The data are extraordinarily quiet for most of that time period and other than right at the start of the sample (where the GARCH variance estimates are still trying to work off the effects of the pre-sample values) the two models actually pretty closely agree. However, there's a lower bound on the variance for the GARCH (because of the variance constant), which is being estimated using a full sample which generally has

⁶There are alternative models (such as jump GARCH) which better handle these types of outliers, allowing for a mixture of outliers which affect the subsequent variance, and those that don't.

much higher volatility than that early phase. Particularly if the record on monetary or trade policies would indicate a likely change at around that time, you could justify simply cutting off the early part of the sample as not being representative of the process later, and in this case, both ways of looking at the data tend to agree—ICSS finds three breaks in the first 214 entries and only five in the last 1600+.

2.2 Rolling Sample Estimates

An increasingly common addition to many recent empirical papers is a “rolling” sample analysis, where the full sample analysis is redone in moving windows across the data set. While at least superficially this seems to be a test for model stability, in fact, it is very rare for anyone producing such an analysis to make any attempt to see whether the differing results across various windows are an actual sign of a structural break. The reality is that each of those sub-sample estimates is subject to (sometimes considerable) sampling error. As a quick example, consider the following, which generates a set of 500 i.i.d. $N(0, 1)$ numbers and does a moving window estimate of the variance using windows of width 50 (Figure 2.7):

```
seed 53030
set random 1 500 = %ran(1.0)
mvstats(width=50,variances=vrandom) random
graph(footer="Moving Average Variance Estimates")
# vrandom
```

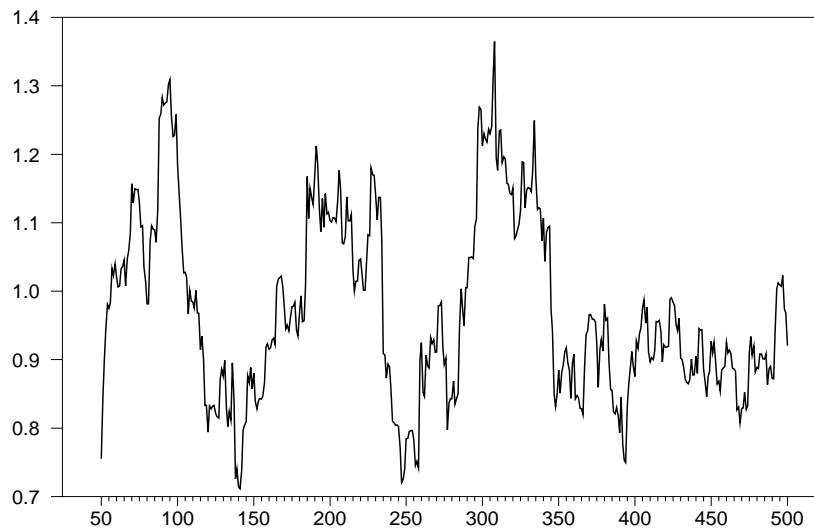


Figure 2.7: Rolling Sample Variances on Random Data

One could probably write a five page paper describing the zigs and zags of this, but *all* of it is simply a demonstration of sampling error—the process is exactly the same throughout. A far too common error is for an author to treat

the rolling estimates as data. We’ve even seen a published paper which did a unit root test on a set of rolling estimates—it’s not at all clear what that would even be testing since even if the underlying model *had* a stability problem, the rolling statistics, basically by construction, will have a near-unit root as consecutive window estimates are based upon almost entirely the same data.

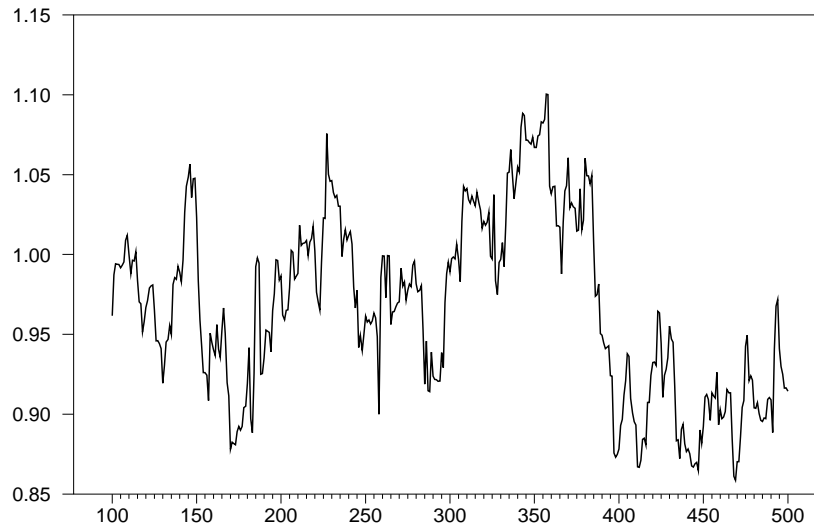


Figure 2.8: Rolling Sample Variances with Wider Window

The difference between run-of-the-mill rolling estimates and the formal tests described earlier in the chapter is that those tests have a null hypothesis and describe the distribution of some specific statistic under the null. Rolling estimates may have the null of stability but looking “different” as a test of that is a bit vague. And how different things can look changes with the width of the window and also with the scale of the graph. Do the same analysis with a window of width 100, and do a standard graph which spreads the data vertically, and you get Figure 2.8, which has a fairly similar appearance to the shorter window size, but rather a different opinion of where the apparent “changes” are. However, put this second graph on the same scale as the first, and the statistics will look almost dead flat.

While one could (and probably should) do simulations or bootstrapping to see how widely the rolling estimates can vary based solely upon sampling error, it might make more sense to think about whether the rolling estimates are actually accomplishing anything in the first place.

Example 2.1 Simple Fluctuation Test

This shows the use of simple fluctuation tests in detecting a break in the mean (in the first part) and a break in the variance (in the second part).

```
seed 53431
*
* Example with break in mean
*
set eps 1 200 = %ran(1.0)+%if(t<=100,-.25,.25)
diff(center) eps / epstilde
acc epstilde / bridge
set bridge = bridge/sqrt(200.0)
set xaxis = t/200.0
scatter(style=lines)
# xaxis bridge
sstats(mean) / bridge^2>>fluxstat
disp "Fluctuation Statistic=" fluxstat $
    "Signif Level" #.#### %nyblomtest(fluxstat,1)
*
* Example with break in variance
*
set eps 1 200 = %if(t<=100,%ran(0.5),%ran(1.5))
diff(standardize) eps / epstilde
acc epstilde / bridge
set bridge = bridge/sqrt(200.0)
scatter(style=lines)
# xaxis bridge
sstats(mean) / bridge^2>>fluxstat
disp "Fluctuation Statistic=" fluxstat $
    "Signif Level" #.#### %nyblomtest(fluxstat,1)
*
* Done with STABTEST procedure
*
@stabtest eps
# constant
```

Example 2.2 Fluctuation Test for GARCH

This shows the use of the FLUX procedure to look for stability problems in a GARCH model.

```
open data garch.asc
data(format=free,org=columns) 1 1867 bp cd dm jy sf
*
set dlogdm = 100*log(dm/dm{1})
*
garch(p=1,q=1,derives=dd) / dlogdm
@flux
# dd
*
```

```

set cdd = dd(1)
acc cdd
graph(footer="Accumulated Gradients on mean in GARCH(1,1)")
# cdd

```

Example 2.3 ICSS Analysis vs GARCH model

This does a comparison of ICSS with GARCH analysis of a set of data and does a combined model using GARCH with identified variance shifts (which is *not* recommended). The details are in Section 2.1.

```

open data garch.asc
data(format=free,org=columns) 1 1867 bp cd dm jy sf
*
set dlogdm = 100*log(dm/dm{1})
*
* Include the mean shift found in looking at fluctuation
* statistics on the original GARCH model
*
set meanshift = t>=1300
*
garch(reg,hseries=hh,derives=dd,likely=llgarch) / dlogdm
# constant meanshift
*
@flux
# dd
set cdd = dd(3)
acc cdd
graph(footer="Fluctuations on Variance Constant in GARCH")
# cdd
*
* Do the linear regression with the mean shift dummy to get
* mean-corrected residuals for @ICSS.
*
linreg dlogdm
# constant meanshift
*
set eps = %resids
*
@icss(subsamples=ss) eps
dec vect[series] xshifts(%cols(ss)-1)
do i=1,%size(xshifts)
    set xshifts(i) = t>=ss(1,i).and.t<=ss(2,i)
end do i
*
garch(p=0,q=0,reg,xreg,hseries=hhicss,likely=llicss) / dlogdm
# constant meanshift
# xshifts
*
graph(footer="Volatility Estimates from GARCH and ICSS",
    overlay=step,ovsamescale) 2
# hh

```

```
# hhicss
*
acc llgarch / cumllgarch
acc llicss / cumllicss
set gap = cumllicss-cumllgarch
graph(footer="Log Likelihood Gap Between ICSS and GARCH")
# gap
```

Parametric Tests

A parametric test for a specific alternative can usually be done quite easily either as a likelihood ratio test or a Wald test. However, if a search over possible breaks is required, it could be quite time-consuming to set up and estimate a possibly non-linear model for every possible break. Instead, looking at a sequence of LM tests is likely to be a better choice.

3.1 Full Coefficient Vector

We are testing for a break at T_0 . Assume first that we're doing likelihood-based estimation. If we can write the log likelihood as:

$$l(\theta) = \sum_{t=1}^T f(\theta|Y_t, X_t) \quad (3.1)$$

then the likelihood with coefficient vector $\theta + \tilde{\theta}$ after the break point is:

$$\sum_{t=1}^{T_0} f(\theta|Y_t, X_t) + \sum_{t=T_0+1}^T f(\theta + \tilde{\theta}|Y_t, X_t)$$

We need an LM test for $\tilde{\theta} = 0$. The first order conditions for $\tilde{\theta}$, evaluated at $\theta = \hat{\theta}$ (the likelihood maximizer for (3.1)) is:

$$\sum_{t=T_0+1}^T \frac{\partial f(\theta|Y_t, X_t)}{\partial \theta}(\hat{\theta}) = 0 \quad (3.2)$$

Similarly, if we are minimizing the least-squares conditions

$$\sum_{t=1}^T u(\theta|Y_t, X_t)^2 \quad (3.3)$$

the first order conditions for $\tilde{\theta}$, evaluated at the solution $\hat{\theta}$ are:

$$\sum_{t=T_0+1}^T \frac{\partial u(\theta|Y_t, X_t)'}{\partial \theta}(\hat{\theta}) u(\hat{\theta}|Y_t, X_t) = 0 \quad (3.4)$$

For the simplest and probably most important case of linear least squares, we have $u_t = Y_t - X_t\theta$, which reduces the condition (3.4) to

$$\sum_{t=T_0+1}^T X'_t u_t = 0$$

where we're using the shorthand $u_t = u(\hat{\theta}|Y_t, X_t)$. As with the fluctuation test, the question is whether the moment conditions which hold (by definition) over the full sample also hold in the partial samples.

In sample, neither (3.2) nor (3.4) will be zero—the LM test is whether they are *close to* zero. If we use the general notation of g_t for the summands in (3.2) or (3.4), then we have the following under the null of no break, with everything evaluated at $\hat{\theta}$:

$$\begin{aligned} \sum_{t=1}^T g_t &= 0 \\ \sum_{t=T_0+1}^T g_t &\approx 0 \end{aligned} \tag{3.5}$$

In order to convert the partial sample sum into a test statistic, we need a variance for it. The obvious choice is the (properly scaled) covariance matrix that we compute for the whole sample under the null. In computing an LM test, we need to take into account the covariance between the derivatives with respect to the original set of coefficients and the test set, that is, the two blocks in (3.5). If we use the full sample covariance matrix, this gives us a very simple form for this. If we assume that

$$\frac{1}{\sqrt{T}} \sum_{t=1}^T g_t \xrightarrow{d} N(0, \mathcal{J})$$

then the covariance matrix of the full (top) and partial (bottom) sums are (approximately):

$$\begin{bmatrix} T\mathcal{J} & (T-T_0)\mathcal{J} \\ (T-T_0)\mathcal{J} & (T-T_0)\mathcal{J} \end{bmatrix} = \begin{bmatrix} T & (T-T_0) \\ (T-T_0) & (T-T_0) \end{bmatrix} \otimes \mathcal{J}$$

The inverse is

$$\begin{bmatrix} \frac{1}{T_0} & -\frac{1}{T_0} \\ -\frac{1}{T_0} & \frac{1}{T_0(T-T_0)} \end{bmatrix} \otimes \mathcal{J}^{-1}$$

The only part of this that matters is the bottom corner, since the full sums are zero at $\hat{\theta}$, so the LM test statistic is:

$$LM = \frac{T}{T_0(T-T_0)} \tilde{g}' J^{-1} \tilde{g} \tag{3.6}$$

where \tilde{g} is the sum of g_t over the tested subsample. This is basically formula (4.4) in Andrews (1993). In practice, we usually have a direct estimator for

$T \times J$, which converts this

$$LM = \frac{T^2}{T_0(T - T_0)} \tilde{g}'(TJ)^{-1} \tilde{g} = \frac{1}{\pi(1 - \pi)} \tilde{g}'(TJ)^{-1} \tilde{g} \quad (3.7)$$

where $\pi = T_0/T$ is the fraction of entries prior to the break entry.

3.1.1 Linear Least Squares

For linear least squares, the LM calculations can give exactly the same results (with much less work) as a series of Wald tests, at least under certain conditions. If we assume that the residuals are conditionally homoscedastic, that is, $E(u_t^2|X_t) = \sigma^2$, then we can use for the joint covariance matrix for (3.5) the finite sample estimate:

$$\sigma^2 \begin{bmatrix} \sum_{t=1}^T X_t' X_t & \sum_{t=T_0+1}^T X_t' X_t \\ \sum_{t=T_0+1}^T X_t' X_t & \sum_{t=T_0+1}^T X_t' X_t \end{bmatrix} \equiv \sigma^2 \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B} & \mathbf{B} \end{bmatrix}$$

The bottom corner of this in the partitioned inverse is $\sigma^{-2}(\mathbf{B} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B})^{-1}$, so the LM test statistic is:

$$\sigma^{-2} \tilde{g}'(\mathbf{B} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B})^{-1} \tilde{g}$$

$\tilde{g}'(\mathbf{B} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B})^{-1} \tilde{g}$ is *exactly* the difference in the sum of squares in adding the dummied-out regressors to the original regressions. The only difference in construction between a (standard) LM test and a Chow test for the break at T_0 is the choice for the sample estimate of σ^2 —the LM test would generally use the estimate under the null, while the Chow test would use the estimate under the unrestricted model. Of course, since we can compute the difference in the sum of squares between the two, we can easily get the sum of squared residuals either way when we do the LM calculation.

This calculation is part of several procedures: `@APBREAKTEST`, `@REHGBREAK` and `@THRESHTEST`. It's a bit more convenient to start the break calculations at the beginning of the data set.¹ The main loop for this will look something like:

```
linreg(noprint) y
# list of explanatory variables
compute dxx =%zeros(%nreg,%nreg)
compute dxs =%zeros(%nreg,1)
```

¹We can get the sequencing above by running the loop backwards.

```

do time=%regstart(),%regend()
  compute x=%eqnxvector(0,time)
  compute dxx =dxx +%outerxx(x)
  compute dxe =dxe +x*%resids(time)
  if time<piStart.or.time>piEnd
    next
  compute ss=(dxx-dxx*%xx*dxx)
  compute rssdiff=%qform(inv(ss),dxe)
  *

  compute fstat=(rssdiff)/((%rss-rssdiff)/(%ndf-%nreg))
  compute wstat(time)=fstat
  if rep==0.and.rssdiff>bestdiff
    compute bestbreak=time,bestdiff=rssdiff
end do time

```

The full-sample **LINREG** gives the residuals (as **%RESIDS**), the inverse of the full sample cross product matrix of the **X** as **%XX** and the sum of squared residuals for the base model as **%RSS**. We need running sums of both $X'X$ (**DXX** in the code) and $X'u$ (**DXE**), which are most easily done using **%EQNXVECTOR** to extract the **X** vector at each entry, then adding the proper functions on to the previous values. **RSSDIFF** is calculated as the difference between the sum of squared residuals with and without the break dummies. If we wanted to do a standard LM test, we could compute that as **RSSDIFF/%SEESQ**. What this² computes is an F times its numerator degrees of freedom, basically the LM test but with the sample variance computed under the alternative—the sum of squared residuals under the alternative is **%RSS-RSSDIFF**, and the degrees of freedom is the original degrees of freedom less the extra **%NREG** regressors added under the alternative. It's kept in the chi-squared form since the tabled critical values are done that way.

Note that this only computes the test statistic for the zone of entries between **piStart** and **piEnd**. It's standard procedure to limit the test only to a central set of entries excluding a certain percentage on each end—the test statistic can't even be computed for the first and last k entries, and is of fairly limited usefulness when only a small percentage of data points are either before or after the break. However, even though the test statistics are only computed for a subset of entries, the accumulation of the subsample cross product and test vector must start at the beginning of the sample.

The procedures **@APBREAKTEST** and **@REGHBREAK** generally do the same thing. The principal difference is that **@APBREAKTEST** is invoked much like a **LINREG**, while **@REGHBREAK** is a “post-processor”—you run the **LINREG** you want first, then **@REGHBREAK**. The example file **ONEBREAK.RPF** uses **@APBREAKTEST** with

²from the **@REGHBREAK** procedure.

```
@apbreaktest(graph) drpoj 1950:1 2000:12
# constant fdd{0 to 18}
```

The same thing done with **@REGHBREAK** would need:

```
linreg drpoj 1950:1 2000:12
# constant fdd{0 to 18}
@reghbreak
```

There are several other differences. One is that **@APBREAKTEST** computes approximate p -values using formulas from Hansen (1997). The original Andrews-Ploberger paper has several pages of lookup tables for the critical values, which depend upon the trimming percentage and number of coefficients—Hansen worked out a set of transformations which allowed the p -value to be approximated using a chi-square. **@REGHBREAK** allows you to compute approximate p -values by using the fixed regressor bootstrap (Section 3.3).

Both procedures calculate what are known as Andrews-Quandt (or just Quandt) statistics, which is the maximum value of the test statistic,³ and the Andrews-Ploberger statistic, which is the log of an average exponential of the statistics.⁴ The A-P test form has certain (minor) advantages in power (Andrews & Ploberger (1994)), but the more “natural” A-Q form seems to be more popular.

3.1.2 GMM

Example 3.1 does a full-break analysis of GMM estimates—this is based upon an example from Greene’s *Econometric Analysis*, 5th edition. The theorems in Andrews (1993) also apply to this, however, the calculations are a bit more complicated than for least squares.

The model being estimated is a money demand function, with log real money as dependent variable and log output and interest rates as the explanatory variables. To avoid problems with simultaneity, the equation is estimated using GMM with lags of the explanatory variables used as instruments:

```
compute sstart=1951:3
compute send=2000:4

instruments constant tbilrate{0 1} logy{1 2}
linreg(inst,optimal,lags=4,lwindow=bartlett) logmp $
    sstart send resids
# constant logy tbilrate
```

³This was proposed originally in Quandt (1960).

⁴It’s the log of the average of $\exp(.5LM)$. Because the LM statistic is an approximation to the log likelihood ratio statistic, which is $2 \times \log(\text{ratio})$, the $\exp(.5LM)$ is roughly equal to the likelihood ratio itself.

To apply Andrew's theorem to this, the same weight matrix (the one from the full sample) gets applied to all the subsamples—varying the weight matrix would require doing separate GMM estimates at each potential break point for both the before and after samples, and the weight matrices on the small subsamples might be very poorly estimated. We pick the full-sample weight matrix off as the `%WMATRIX` variable. We also save the (real-value of) the number of observations.

```
compute wfull=%wmatrix,baset=float(%nobs)
```

For (linear) GMM, the g_t in (3.5) is

$$g_t = \mathbf{X}'\mathbf{Z}[\mathbf{W}]^{-1}\mathbf{Z}'_t u_t$$

where \mathbf{W} is the (full-sample) weight matrix and the corresponding $T \times J$ is

$$\mathbf{X}'\mathbf{Z}[\mathbf{W}]\mathbf{Z}'\mathbf{X}$$

if the \mathbf{W} is the “optimal” weight matrix. Whether it is the optimal one or not, the TJ is the inverse of the `%XX` matrix that **LINREG** computes. To compute (3.7), we need $\mathbf{X}'\mathbf{Z}$, which is a single full-sample calculation, and the sum of $\mathbf{Z}'_t u_t$ over the subsample. The first can be done outside the loop with

```
cmom(zxmatrix=xz,lastreg,nodepvar,instruments) sstart send
```

which is a special form of the **CMOMENT** instruction designed precisely for this particular matrix, computing $\mathbf{X}'\mathbf{Z}$. \mathbf{X} is the set of regressors in the last regression (**LASTREG** option with **NODEPVAR**) and \mathbf{Z} is the set of instruments (with the calculation done in that order). Outside the loop, we can also take care of all the fixed matrices in the final calculation (leaving only the subsample calculation of $\mathbf{Z}'u$) with

```
compute lmv=%mqform(%xx,xz*wfull)
```

which uses the fact that TJ is the inverse of `%XX` and we need $(TJ)^{-1}$ (and thus just `%XX` as the middle of the “sandwich”).

We now can begin the process of computing the LM statistics. This sets the range of potential change points to roughly (.15,.85):

```
compute cstart=1957:3,cend=1993:3
set lms cstart cend = 0
```

Inside the loop over sample split points,

```
compute pi=(splits-sstart)/baset
```

computes the fraction of the sample up to (but not including) the break point and

```
cmom(zxmatrix=m2pi,instruments) splits+1 send
# resids
compute lms(splits)=1.0/(pi*(1-pi))*%qform(lmv,m2pi)
```

computes the moment conditions between the instruments and the residuals (from the full-sample estimates) over the tested sample and computes the LM statistic. This graphs the series of LM statistics and

```
graph(key=upleft,footer="Structural Change Test Statistics") 1
```

and this computes the three principal forms of the break tests: the sup (Andrews-Quandt), mathematical average, and exponential average (Andrews-Ploberger). The A-P version is computed in an “overflow-safe” fashion—while unlikely to be a problem here, the $\exp(.5 LM)$ could, in some situations, overflow if the LM statistic is large enough. Instead, it subtracts the maximum of LM (the `AQTEST` value) from LM while taking the \exp , then adds it back when taking the log of the exponential average.

```
sstats(max) cstart cend lms>>aqtest
sstats(mean) cstart cend lms>>avtest exp(.5*(lms-aqtest))>>aptest
compute aptest=log(aptest)+.5*aqtest
report(action=define,title="Structural Change Statistics")
report(atrow=1,atcol=2,align=center) "Sup" "Arith Avg" "Exp Avg"
report(atrow=2,atcol=1) "LM" aqtest avtest aptest
report(action=format,picture="*.###")
report(action=show)
```

	Sup	Arith Avg	Exp Avg
LM	10.38	4.42	3.19

which aren’t significant at conventional levels (the p -values are a bit above .10).

3.2 Outliers and Shifts

What we examine here are specific changes to the mean of a process.

3.2.1 Linear Least Squares

The “rule-of-thumb” test for outliers is to check the ratio between the absolute value of the residual and the standard error. Anything above a certain limit (2.5 or 3.0) is considered an outlier. The formal LM test, however, is slightly different. The first order conditions are:

$$\sum_{t=1}^T X_t' u_t = 0$$

$$u_{T_0} \approx 0$$

If we assume conditionally homoscedastic residuals, then the covariance matrix of the first order conditions is:

$$\sigma^2 \begin{bmatrix} \sum_{t=1}^T X_t' X_t & X_{T_0}' \\ X_{T_0} & 1 \end{bmatrix}$$

so the LM statistic (writing $\mathbf{X}'\mathbf{X} = \sum_{t=1}^T X_t' X_t$) is

$$\frac{u_{T_0}^2}{\sigma^2 (1 - X_{T_0}(\mathbf{X}'\mathbf{X})^{-1} X_{T_0}')}$$

This is similar to the rule-of-thumb test, except that it uses the *studentized residuals* $u_t/\sqrt{h_{tt}}$, where $h_{tt} = (1 - X_t(\mathbf{X}'\mathbf{X})^{-1} X_t')$. h_{tt} is between 0 and 1. X_t that are quite different from the average will produce a smaller value of this; such data points have more influence on the least squares fit, and this is taken into account in the LM test—a 2.0 “rule of thumb” ratio on a very influential data point could produce a very high LM ratio, since the least squares fit has already adjusted substantially to try to reduce the residual at that entry.

After you do a `LINREG`, you can generate the *leverage statistics* $X_t(\mathbf{X}'\mathbf{X})^{-1} X_t'$ and the studentized residuals with

```
prj(xvx=h)
set istudent = %resids/sqrt(%seesq*(1-h))
```

These are known as the *internally* studentized residuals as the estimate σ^2 is computed including the data point being tested. There are also externally studentized residuals which use an estimate which excludes it. The example file `BALT3P193.RPF` from the textbook examples for Baltagi (2002) computes these and several related statistics.

3.2.2 ARIMA models

The X11 seasonal adjustment algorithm is designed to decompose a series into three main components: trend-cycle, seasonal and irregular. The seasonally adjusted data are what you get if you take out the seasonal, which means that the irregular is an important part of the final data product. If you have major outliers due to strikes or weather, you can't just ignore them. However, they *will* contaminate the estimates of both the trend-cycle and seasonal if they're allowed to. The technique used by Census X12, and by Statistics Canada's X11-ARIMA before that, is to compute a separate pre-adjustment component that takes out the identifiably irregular parts, applies X11 to the preliminarily adjusted data, then puts the extracted components back at the end.

The main focus in this preliminary adjustment is on two main types of additive outliers, as defined in Section 1.3. One (unfortunately) is *itself* known as

an additive outlier, which is a single period shift. The other is a permanent level shift. In the standard set, there's also a "temporary change", which is an impulse followed by an geometric decline back to zero, but that isn't used much.⁵ The shifts are analyzed in the context of an ARIMA model. Seasonal ARIMA models can take a long time to estimate if they have any seasonal AR or seasonal MA components, as those greatly increase the complexity of the likelihood function. As a result, it really isn't feasible to do a search across possible outliers by likelihood ratio tests. Instead, the technique used is something of a stepwise technique:

1. Estimate the basic ARIMA model.
2. Scan using LM tests for the outlier type(s) of interest.
3. If one is found to be significant enough, add the effect to the ARIMA model and re-estimate. Repeat Step 2.
4. If there are no new additions, look at the t -statistics on the shifts in the final ARIMA model. If the least significant one is below a cutoff, prune it, and re-estimate the reduced model. Repeat until all remaining shift variables are significant.

This process is built into the **BOXJENK** instruction. To employ it, add the option **OUTLIER** to the **BOXJENK**. Example 3.2 applies this to the well-known "airline" data with:

```
boxjenk(diffs=1,sdiffs=1,ma=1,sma=1,outliers=ao) airline
```

The main choices for **OUTLIERS** are **AO** (only the single data points), **LS** (for level shift) and **STANDARD**, which is the combination of **AO** and **LS**. With **STANDARD**, both the **AO** and **LS** are scanned at each stage, and the largest between them is kept if sufficiently significant. The scan output from this is:

```
Forward Addition pass 1
Largest t-statistic is AO(1960:03)=   -4.748 >   3.870 in abs value

Forward Addition pass 2
Largest t-statistic is AO(1951:05)=    2.812 <   3.870 in abs value

Backwards Deletion Pass 1
No outliers to drop. Smallest t-statistic   -5.148 >   3.870 in abs value
```

The tests are reported as t -statistics, so the LM statistics will be the squares. The 3.870 is the standard threshold value for this size data set—it may seem rather high, but the X11 algorithm has its own "robustness" calculations, so only very significant effects need special treatment. As we can see, this adds one outlier dummy at 1960:3, fails to find another, then keeps the one that was added in the backwards pass. The output from the final **BOXJENK** (the output from all the intermediate ones is suppressed) is in Table 3.1.

⁵The rate of decline has to be fixed in order for it to be analyzed easily.

Table 3.1: ARIMA Output with Outliers

Box-Jenkins - Estimation by ML Gauss-Newton				
Convergence in 11 Iterations. Final criterion was 0.0000049 <= 0.0000100				
Dependent Variable AIRLINE				
Monthly Data From 1950:02 To 1960:12				
Usable Observations		131		
Degrees of Freedom		128		
Centered R ²		0.9914		
R-Bar ²		0.9912		
Uncentered R ²		0.9989		
Mean of Dependent Variable		295.6336		
Std Error of Dependent Variable		114.8447		
Standard Error of Estimate		10.7636		
Sum of Squared Residuals		14829.4453		
Log Likelihood		-495.7203		
Durbin-Watson Statistic		2.0011		
Q(32-2)		42.9961		
Significance Level of Q		0.0586		
<hr/>				
	Variable	Coeff	Std Error	T-Stat Signif
1.	AO(1960:03)	-43.2540	8.5271	-5.0726 0.0000
2.	MA{1}	-0.2471	0.0866	-2.8525 0.0051
3.	SMA{12}	-0.0884	0.0949	-0.9314 0.3534

Note that both the forward and backwards passes use a sharp cutoff. As a result, it's possible for small changes to the data (one extra data point, minor revision to a reported value) to change whether something shows as a significant outlier or not. As a result, the Census Bureau generally does not do a new outlier analysis every month, but instead, hard codes a set that were the ones detected in a previous benchmarking run.

3.2.3 GARCH models

The “mean” model in a GARCH is often neglected in favor of the “variance” model, but the very first assumption underlying a GARCH is that the residuals are mean zero and serially uncorrelated. You can check serial correlation by applying standard tests like Ljung-Box or West-Cho to the standardized residuals. But that won't help if the basic mean model is off.

In deriving an LM test for outliers and level shifts, the variance can't be treated as a constant in taking the derivative with respect to mean parameters, since it's a recursively-defined function of the residual. Simple LM tests based upon (3.1) won't work because the likelihood at t is a function of the data for all previous values. The recursion required to get the derivative will be different for each type of GARCH variance model. For the simple univariate GARCH(1,1), we have

$$h_t = c + bh_{t-1} + a\varepsilon_{t-1}^2$$

so for a parameter θ which is just in the mean model:

$$\frac{\partial h_t}{\partial \theta} = b \frac{\partial h_{t-1}}{\partial \theta} + 2a\varepsilon_{t-1} \frac{\partial \varepsilon_{t-1}}{\partial \theta}$$

For a one-shot outlier,

$$\frac{\partial \varepsilon_t}{\partial \theta} = \begin{cases} 1 & \text{if } t = T_0 \\ 0 & \text{o.w.} \end{cases}$$

and for a level shift,

$$\frac{\partial \varepsilon_t}{\partial \theta} = \begin{cases} 1 & \text{if } t > T_0 \\ 0 & \text{o.w.} \end{cases}$$

The procedure `@GARCHOutlier` in Example 3.3 does the same types of outlier detection as we described for ARIMA models. The calculation requires that you save the partial derivatives when the model is estimated:⁶

```
garch(p=1,q=1,reg,hseries=h,resids=eps,derives=dd) start end y
# constant previous
```

Once the gradient of the log likelihood with respect to the shift dummy is computed (into the series `GRAD`), the LM statistic is computed using:

```
mcov(opgstat=lmstat(test))
# dd grad
```

This uses a sample estimate for the covariance matrix of the first order conditions, basically, the BHHH estimate of the covariance matrix and computes the LM statistic using that.

The following does the sequence of LM tests for both additive outliers and level shifts (`OUTLIER=STANDARD` option controls that). The `VECT[SERIES]` `OUTLIERS` (initially empty) is used for previously identified dummies.

```
dec vect[series] outliers(0)
@GARCHOutlier(outlier=standard,graph) dlogdm / outliers
```

This generates Figures 3.1 for the additive outliers and 3.2 for level shifts. The largest of test across both is reported as

Maximum LM for Level Shift 17.703 at 1303

The level shift test gives us the same result (in effect) that we got with the fluctuation test in Example 2.2. You might ask why the 5 standard deviation outlier described on 23 doesn't show more prominently in Figure 3.1. There are two main reasons for that. First, unlike both least squares and ARIMA models, the covariance matrix of the gradient can't be split by conditioning into a full-sample variance estimate times a calculation based upon the X_{T_0}

⁶The use of `PREVIOUS` allows you to feed in previously located shifts.

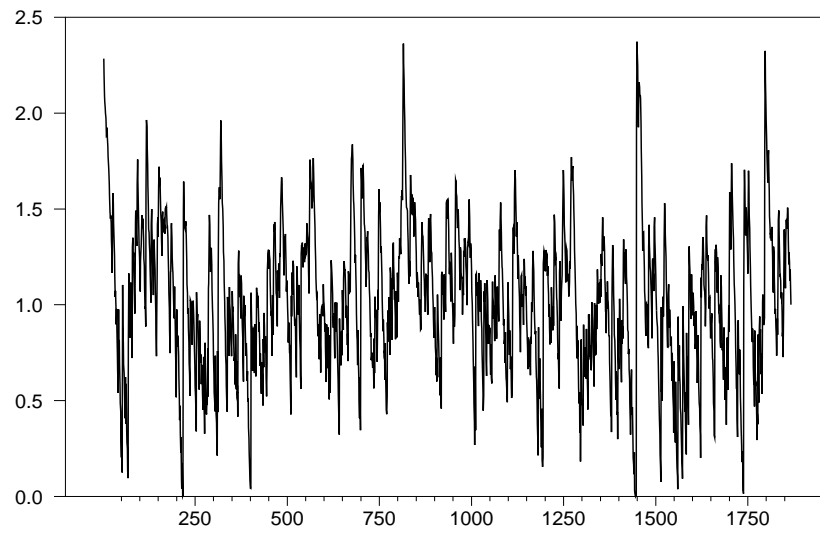


Figure 3.1: GARCH Additive Outlier LM Tests

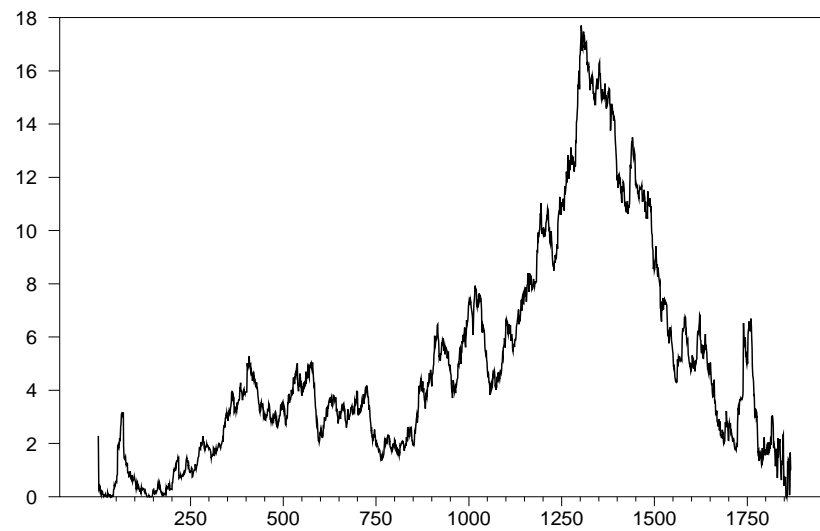


Figure 3.2: GARCH Level Shift Outlier LM Tests

only. Since the gradient on the outlier dummy is largely zero except at time T_0 (it will be zero before that, and non-zero but generally small after it), the covariance matrix estimate for the gradient will be dominated by the square of the one value, so an LM statistic formed by squaring the gradient and dividing by that would be expected to be close to 1. Now the *actual* LM statistic is formed by including the gradient elements of all the other parameters, which might cause an effect similar to the “studentization” in Section 3.2.1. However, that only happens if there is just one major single-observation outlier (so the other parameters will be adjusting to try to reduce the effect of it). In this case, while the 1448 is the biggest standardized residual, there are quite a few others that are 4 standard deviations or larger, so it isn’t as prominent as it would be in a more typical regression model.

The example program then does a second pass at the outlier detection adding in a mean shift for the largest value detected in the first try.

```
dim outliers(1)
set outliers(1) = (t>=1303)
@GARCHOutlier(outlier=standard,graph) dlogdm / outliers
```

This detects no further outliers in either form.

3.3 Fixed Regressor Bootstrap

The *fixed regressor bootstrap* was proposed in Hansen (2000) to provide a relatively straightforward method to approximate the p values for tests in a wide variety of settings, and, in particular, for stability tests. There are two key assumptions:

1. You have a search over a range of “nuisance parameters” which are unidentified under the null.
2. The test statistic would have an asymptotic chi-squared distribution if those nuisance parameters were known.

The breakpoint tests in Section 3.1 are an example of this. Under the null that there is no break, there is no break *point*, but if we choose to look at a break at a particular entry, the test for a difference between the regressions in the two regimes will be asymptotically chi-squared. Similarly, the threshold autoregression in Chapter 4 requires a search over both a delay (lag for determining a break) and threshold value. Hansen demonstrates that the p -value of a test done by optimizing a functional of the basic test statistics (such as the sup or an average) over the nuisance parameters can be approximated by randomizing only the residual, treating the regressors (and possible threshold variables) as fixed (hence the name of the test). Note that this is for evaluating test statistics only—not for the more general task of evaluating the full distribution of the estimates.

If we look at a test which comes down to (3.5), the idea is that if you multiply g_t by ξ_t where $\xi_t \sim N(0, 1)$ *i.i.d.*, then Eg_t is zero for all t regardless, and the variance of g_t is unchanged—Hansen shows that if you do the testing procedure on this altered “data”, the functionals will have the same distribution as they do in the original model under the null.⁷

For example, in Section 3.1.2, g_t is

$$\mathbf{X}'\mathbf{Z}[\mathbf{W}]^{-1}\mathbf{Z}'_t u_t$$

so $g_t \xi_t$ is

$$\mathbf{X}'\mathbf{Z}[\mathbf{W}]^{-1}\mathbf{Z}'_t (u_t \xi_t)$$

The actual test statistic is computed by doing the calculations with the original u_t , while the bootstrapped calculations are done using $u_t \xi_t$ —all the other parts of the calculation remain the same. Because of that, a convenient way to do the calculations is to loop over the breakpoint calculations for $N + 1$ trips, where N is the number of bootstrap replications. On one of those passes, use the original residuals, and on all the others, use the altered residuals. Keep a series of generated tests on the bootstrapped draws and compare them against the test statistics from the actual residuals. Example 3.4 demonstrates this. It does 500 bootstrap draws (which should be adequate for almost any purpose), and (as in the earlier example) computes three functionals of the LM statistics: the sup (Andrews-Quandt), arithmetic average and Andrews-Ploberger exponential average. This does the additional setup required outside the loop. To this point everything is identical to Example 3.1.

```
compute nreps=500
set aqtests 1 nreps = 0.0
set avtests 1 nreps = 0.0
set aptests 1 nreps = 0.0
```

A DO loop is put around the original loop over the break points. We’re running this from 1 to NREPS+1, as we’ll do the actual test on the real data on the final trip through this. The top of the loop looks like:

```
do rep=1,nreps+1
  set lms   cstart cend = 0
  *
  if rep==nreps+1
    set u = resids
  else
    set u = %ran(1.0)*resids
```

⁷Note that this approximation result is based upon local-to-null alternatives, which means that as T gets large, the alternative moves towards the null at a controlled rate. If the null is (very) false, the bootstrapped test values will be quite a bit more extreme than they would be under the null. However, the test should still strongly reject in that case—you will just understate the true significance level. The point of the fixed regressor bootstrap is that it can be used in situations where simulations under the null itself are either difficult or impossible.

The calculation of the LM statistic for a particular split point is done with

```
cmom(zxmatrix=m2pi,instruments) splits+1 send
# u
compute lms(splits)=1.0/(pi*(1-pi))*%qform(lmv,m2pi)
```

where U is either the original residuals from the full-sample GMM (when `REP` is `NREPS+1`) or that multiplied by random normals on all the bootstrap draws.

The end of the loop calculates the three functionals on the LM statistics, and saves the bootstrapped values. Since the actual test statistics are computed on the final pass, the final values for `AQTEST`, `AVTEST` and `APTEST` will be holding the ones we want when we're done.

```
sstats(max) cstart cend lms>>aqtest
sstats(mean) cstart cend lms>>avtest exp(.5*(lms-aqtest))>>aptest
compute aptest=log(aptest)+.5*aqtest
if rep<=nreps {
  compute aqtests(rep)=aqtest
  compute avtests(rep)=avtest
  compute aptests(rep)=aptest
}
end do rep
```

The bootstrapped p -values are computed as the fraction of the bootstrapped values which exceed the actual sample test statistics:

```
sstats(mean) 1 nreps (aqtests>=aqtest)>>pvalaq $
(avtests>=avtest)>>pvalav (aptests>=aptest)>>pvalap

report(action=define,title="Structural Change Statistics, GMM Model")
report(atrow=1,atcol=1) "Test Form" "Statistic" "P-Value"
report(atrow=2,atcol=1) "Sup"          aqtest      pvalaq
report(atrow=3,atcol=1) "Arith Avg"    avtest      pvalav
report(atrow=4,atcol=1) "Exp Avg"      aptest      pvalap
report(action=format,picture="*.###")
report(action=show)
```

The test statistics themselves are clearly identical to what they were in Example 3.1, but now we have confirmation that they're insignificant at conventional levels without having to use a lookup table.⁸

Test Form	Statistic	P-Value
Sup	10.381	0.274
Arith Avg	4.416	0.194
Exp Avg	3.192	0.236

A few things to note:

⁸The p -values are subject to simulation error, so they won't match exactly.

- In some cases, the bootstrap can be done by directly replacing the residuals with $N(0, 1)$ i.i.d. draws (if the model assumes homoscedasticity and the LM test statistic computes its own estimate of the residual variance.)
- If the model has more than one series of residuals, all of them get multiplied by the same ξ_t in computing the bootstrapped g .
- If the LM test statistic requires a calculation which is based upon serial correlation in g (for instance, you're doing Newey-West covariance matrices), you can't use the fixed regressor bootstrap, since multiplying by the independent ξ will break the pattern of serial correlation.
- This is not designed to test for breaks in *variance*, or tests which allow for breaks in the variance—the variance of residuals (or covariance if it's a multivariate model) is assumed to be unconditionally constant.

Example 3.1 Break Analysis for GMM

This is based on an example from Greene's *Econometric Analysis*, 5th Edition, which was dropped from later editions. It's Example 7.7 on pp 142-143. The example in the book does LM, Wald and "likelihood ratio" tests for breaks—the last two require doing estimates of the model in each subsample, which the LM, which is the only one included here, does not. Unlike the case with least squares, the three don't give identical results.

This is revisited in Example 3.4, which does a fixed regressor bootstrap to compute approximate p -values.

```
open data tablef5-1[1].txt
calendar(q) 1950
data(format=prn,org=columns) 1950:1 2000:4 year qtr realgdp realcons $
    realinvs realgovt realdpi cpi_u m1 tbilrate unemp pop infl realint
*
set logmp = log(m1/cpi_u)
set logy = log(realgdp)
*
compute sstart=1951:3
compute send=2000:4
*
* Do the regression over the full period
*
instruments constant tbilrate{0 1} logy{1 2}
linreg(inst,optimal,lags=4,lwindow=bartlett) logmp $
    sstart send resid
# constant logy tbilrate
*
* Get the full sample weight matrix, and the number of observations
*
compute wfull=%wmatrix,baset=float(%nobs)
*
* Compute the full sample X'Z matrix (this is -derivative of moment
* conditions wrt the coefficients). This defines X'Z (X =
* regressors, Z = instruments) because the regressor list is used
* as the first list, and the instruments as the second.
*
cmom(zxmatrix=xz,lastreg,nodepvar,instruments) sstart send
*
* Compute the W * (Z'X) x inv(J) x X'Z W needed for the LM tests.
*
compute lmv=%mqform(%xx,xz*wfull)
*
* Set the change point range to roughly (.15,.85)
*
compute cstart=1957:3,cend=1993:3
*
set lms    cstart cend = 0
do splits=cstart,cend
*
```



```

* Figure out what the pi value is for this change point
*
compute pi=(splits-sstart)/baset
*
* Get the moment conditions for the second subsample
*
cmom(zxmatrix=m2pi,instruments) splits+1 send
# resids
compute lms(splits)=1.0/(pi*(1-pi))*qform(lmv,m2pi)
end do splits
*
* Graph the test statistics
*
graph(key=upleft,footer="Structural Change Test Statistics") 1
# lms
*
* Figure out the grand test statistics
*
sstats(max) cstart cend lms>>aqtest
sstats(mean) cstart cend lms>>avtest exp(.5*(lms-aqtest))>>aptest
compute aptest=log(aptest)+.5*aqtest
report(action=define,title="Structural Change Statistics")
report(atrow=1,atcol=2,align=center) "Sup" "Arith Avg" "Exp Avg"
report(atrow=2,atcol=1) "LM" aqtest avtest aptest
report(action=format,picture="*.##")
report(action=show)

```

Example 3.2 ARIMA Model with Outlier Handling

This applies the standard outlier tests for an ARIMA model. From Section 3.2.2.

```

cal(m) 1949
data(format=free,unit=input) 1949:1 1960:12 airline
  112 118 132 129 121 135 148 148 136 119 104 118
  115 126 141 135 125 149 170 170 158 133 114 140
  145 150 178 163 172 178 199 199 184 162 146 166
  171 180 193 181 183 218 230 242 209 191 172 194
  196 196 236 235 229 243 264 272 237 211 180 201
  204 188 235 227 234 264 302 293 259 229 203 229
  242 233 267 269 270 315 364 347 312 274 237 278
  284 277 317 313 318 374 413 405 355 306 271 306
  315 301 356 348 355 422 465 467 404 347 305 336
  340 318 362 348 363 435 491 505 404 359 310 337
  360 342 406 396 420 472 548 559 463 407 362 405
  417 391 419 461 472 535 622 606 508 461 390 432
*
boxjenk(diffs=1,sdifs=1,ma=1,sma=1,outliers=ao) airline

```

Example 3.3 GARCH Model with Outlier Handling

This demonstrates outlier detection for a GARCH model. This is described in detail in Section 3.2.3.

```
* @GARCHOutlier y start end previous
* does an automatic outlier test for additive outliers and level shifts
* in a GARCH(1,1) model. previous is a VECT[SERIES] of shift series that
* have already been identified.
*
* Options:
* OUTLIER=NONE/AO/LS/[STANDARD]
*   The types of outliers for which to scan. STANDARD means both
* GRAPH/[NOGRAPH]
*   GRAPH requests graphs of the LM statistics.
*
procedure GARCHOutlier y start end previous
type vect[series] *previous
*
option choice outlier 4 none ao ls standard
option switch graph 0
*
local vect[series] dd
local series h eps dh deps grad lmstat
local integer test gstart gend
local real beststat
local integer bestbreak besttype
*
garch(p=1,q=1,reg,hseries=h,resids=eps,derives=dd) start end y
# constant previous
compute gstart=%regstart(),gend=%regend()
*
compute beststat=0.0
if outlier==2.or.outlier==4 {
  *
  * Additive outlier
  *
  set lmstat gstart gend = 0.0
  do test=gstart,gend
    set deps gstart gend = (t==test)
    set(first=0.0) dh gstart gend = $
      %beta(%nreg)*dh{1}+%beta(%nreg-1)*eps{1}*deps{1}
    set grad gstart gend = -.5*dh/h+.5*(eps/h)^2*dh-(eps/h)*deps
    mcov(opgstat=lmstat(test))
    # dd grad
  end do test
  ext(noprint) lmstat gstart gend
  if graph {
    graph(header="Additive Outliers")
    # lmstat gstart gend
  }
  compute bestbreak=%maxent
  compute beststat =%maximum
}
```

```

    compute besttype =2
}
if outlier==3.or.outlier==4 {
*
* Level shift. We leave out the first and last entries, since those
* are equivalent to additive outliers.
*
set lmstat gstart+1 gend-1 = 0.0
do test=gstart+1,gend-1
    set deps gstart gend = (t>=test)
    set(first=0.0) dh gstart gend = $
        %beta(%nreg)*dh{1}+%beta(%nreg-1)*eps{1}*deps{1}
    set grad gstart gend = -.5*dh/h+.5*(eps/h)^2*dh-(eps/h)*deps
    mcov(opgstat=lmstat(test))
    # dd grad
end do test
ext(noprint) lmstat gstart+1 gend-1
if graph {
    graph(header="Level Shifts")
    # lmstat gstart gend
}
if %maximum>beststat {
    compute beststat=%maximum
    compute bestbreak=%maxent
    compute besttype =3
}
}
*
if outlier<>1 {
    if besttype==2
        disp "Maximum LM for Additive Outlier" *.### beststat $
            "at" %datelabel(bestbreak)
    else
        disp "Maximum LM for Level Shift" *.### beststat $
            "at" %datelabel(bestbreak)
}
end
*****
open data garch.asc
all 1867
data(format=free,org=columns) / bp cd dm jy sf
set dlogdm = 100*log(dm/dm{1})
*
dec vect[series] outliers(0)
@GARCHOutlier(outlier=standard,graph) dlogdm / outliers
dim outliers(1)
set outliers(1) = (t>=1303)
@GARCHOutlier(outlier=standard,graph) dlogdm / outliers

```

Example 3.4 Fixed Regressor Bootstrap

This demonstrates the fixed regressor bootstrap (Section 3.3) with the GMM stability test used in Example 3.1.

```

open data tablef5-1[1].txt
calendar(q) 1950
data(format=prn,org=columns) 1950:1 2000:4 year qtr realgdp realcons $
    realinvs realgovt realdpi cpi_u m1 tbilrate unemp pop infl realint
*
set logmp = log(m1/cpi_u)
set logy  = log(realgdp)
*
compute sstart=1951:3
compute send=2000:4
*
* Do the regression over the full period
*
instruments constant tbilrate{0 1} logy{1 2}
linreg(inst,optimal,lags=4,lwindow=bartlett) logmp $
    sstart send resids
# constant logy tbilrate
*
* Get the full sample weight matrix, and the number of observations
*
compute wfull=%wmatrix,baset=float(%nobs)
*
* Compute the full sample X'Z matrix (this is -derivative of moment
* conditions wrt the coefficients). This defines X'Z (X =
* regressors, Z = instruments) because the regressor list is used
* as the first list, and the instruments as the second.
*
cmom(zxmatrix=xz,lastreg,nodepvar,instruments) sstart send
*
* Compute the  $W * (Z'X) * \text{inv}(J) * X'Z W$  needed for the LM tests.
*
compute lmv=%mqform(%xx,xz*wfull)
*
* Set the change point range to roughly (.15,.85)
*
compute cstart=1957:3,cend=1993:3
*
compute nreps=500
set aqtests 1 nreps = 0.0
set avtests 1 nreps = 0.0
set aptests 1 nreps = 0.0
*
do rep=1,nreps+1
    set lms    cstart cend = 0
    *
    if rep==nreps+1
        set u = resids
    else

```

```

    set u = %ran(1.0)*resids
do splits=cstart,cend
*
* Figure out what the pi value is for this change point
*
compute pi=(splits-sstart)/baset
*
* Get the moment conditions for the second subsample
*
cmom(zxmatrix=m2pi,instruments) splits+1 send
# u
compute lms(splits)=1.0/(pi*(1-pi))*%qform(lmv,m2pi)
end do splits
sstats(max) cstart cend lms>>aqtest
sstats(mean) cstart cend lms>>avtest exp(.5*(lms-aqtest))>>aptest
compute aptest=log(aptest)+.5*aqtest
if rep<=nreps {
    compute aqtests(rep)=aqtest
    compute avtests(rep)=avtest
    compute aptests(rep)=aptest
}
end do rep
*
sstats(mean) 1 nreps (aqtests>=aqtest)>>pvalaq $
(avtests>=avtest)>>pvalav (aptests>=aptest)>>pvalap
*
report(action=define,title="Structural Change Statistics, GMM Model")
report(atrow=1,atcol=1) "Test Form" "Statistic" "P-Value"
report(atrow=2,atcol=1) "Sup"          aqtest      pvalaq
report(atrow=3,atcol=1) "Arith Avg"    avtest      pvalav
report(atrow=4,atcol=1) "Exp Avg"      aptest      pvalap
report(action=format,picture="*.###")
report(action=show)

```

TAR Models

Figure 4.1 shows data on the U.S. Civilian Unemployment Rate. Note that the behavior of the series is asymmetrical—the increases are more rapid than the decreases. This is a problem for “linear” time series models like autoregressions or ARIMA models, which, by their nature, have up and down phases which have to have similar properties.

In Example 4.1, we estimate a linear AR(4) model on the first difference of this series. There’s no obvious problem with the estimates, but if we simulate a realization of the estimated model for the unemployment rate, we get Figure 4.2, which, as you can see, shows very similar upwards and downwards movements, and thus hasn’t really captured the dynamics of the actual data.

The Threshold Autoregression (TAR) model is an autoregression allowing for two or more branches governed by the values for a threshold variable. This allows for asymmetric behavior that can’t be explained by a single ARMA model. For a two branch model, one way to write this is:

$$y_t = \begin{cases} \phi_{11}y_{t-1} + \dots + \phi_{1p}y_{t-p} + u_t & \text{if } z_{t-d} \leq c \\ \phi_{21}y_{t-1} + \dots + \phi_{2q}y_{t-q} + u_t & \text{if } z_{t-d} > c \end{cases} \quad (4.1)$$

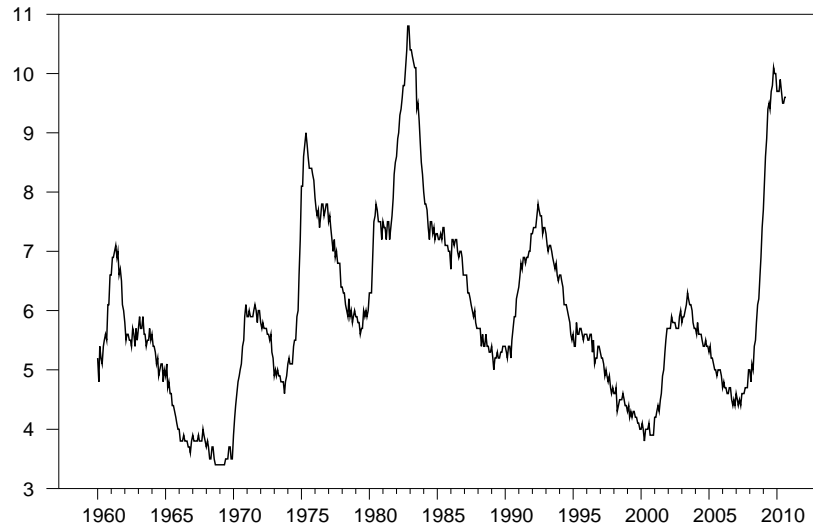


Figure 4.1: U.S. Civilian Unemployment Rate

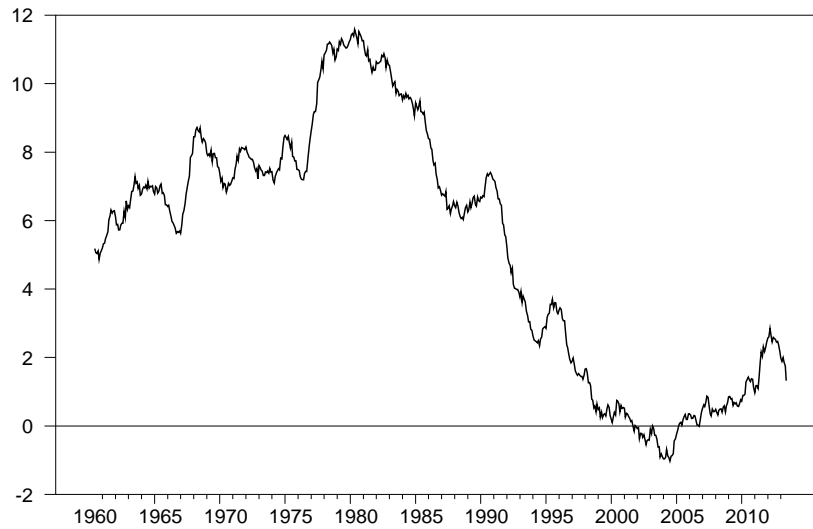


Figure 4.2: Simulated Unemployment Rate from Linear AR

A SETAR model (Self-Exciting TAR) is a special case where the threshold variable is y itself.

In this chapter, we'll work with two data series. Example 4.1 uses the U.S. unemployment rate and employs a standard TAR model. In Example 4.2, we'll look at the spread between U.S. short and long-run interest rates (Figure 4.3), where we will be using the data set from Enders & Granger (1998). This will use what the authors call the "linear attractor" model:

$$\Delta y_t = \begin{cases} \beta \Delta y_{t-1} + \rho_2 (y_{t-1} - a_0) + u_t & \text{if } y_{t-1} \leq a_0 \\ \beta \Delta y_{t-1} + \rho_1 (y_{t-1} - a_0) + u_t & \text{if } y_{t-1} > a_0 \end{cases}$$

Unlike (4.1), where the two branches are completely separate, this has β in common, which requires estimating the model with dummies over the full sample, and will thus require some special purpose programming.

Throughout the chapter, we'll switch between discussions of the two examples, since each is very similar analysis applied to different models.

Note that each of these examples might potentially have a problem with round-off error in the data. "Headline" unemployment in the U.S. is usually presented with just one decimal place, so period to period differences will usually be one of a small number of values (-.2,-.1,0,.1,.2). However, in machine-arithmetic, all .1's are not "equal" because of rounding, and that could cause a problem for "sharp cutoff" models. In this case, we avoid that by using a three decimal place version; alternatively, you can use the `%ROUND` function to force the differenced values to match. See page 65 for more on that.

4.1 Estimation

The first question is how to pick p in (4.1). If there really is a strong break at an unknown value, then standard methods for picking p based upon information

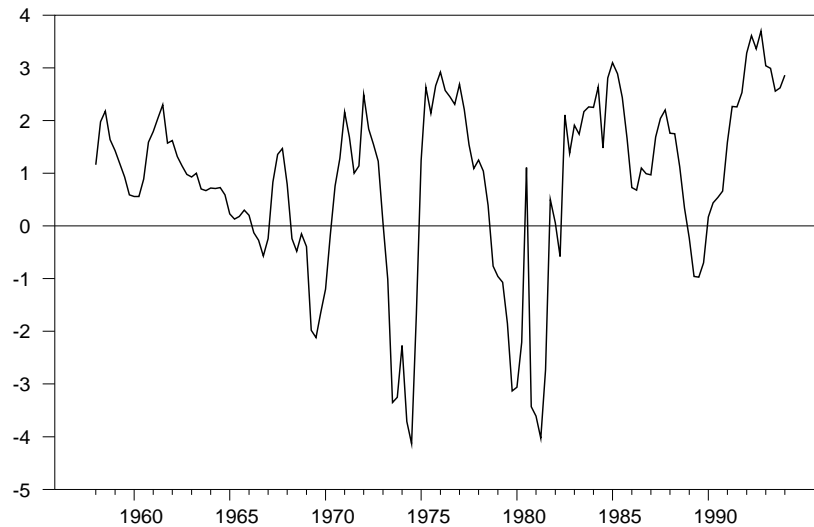


Figure 4.3: U.S. Interest Rate Spread

criteria won't work well, since the full data series doesn't follow an $AR(p)$, and you can't easily just apply those to a subset of the data, like you could if the break were based upon time and not a threshold value. The standard strategy is to overfit to start, then prune out unnecessary lags. While (4.1) is written with the same structure in each branch, in practice they can be different.

The second question is how to pick the threshold z and delay d . For a SETAR model, z is y . Other variables might be suggested by theory, as, for instance, in the Enders-Granger paper, which includes an M-TAR (Momentum TAR) model where z is Δy . The delay is obviously a discrete parameter, so estimating it requires a search over the possible values. c is less obviously discrete, but, in fact, the only values of c at which (4.1) changes are observed values for z_{t-d} , so it's impossible to use variational methods to estimate c . Instead, that will *also* require a search. If both d and c are unknown, a nested search over d and, given a test value of d , over the values of z_{t-d} is necessary.

4.2 Testing

The test which suggests itself is a likelihood ratio test or something similar which compares the one-branch model with the best two-branch model. The problem with that approach is that the test statistic will not have a standard asymptotic distribution because of both the lack of differentiability with respect to c , plus the lack of identification of c if there is no difference between the branches. The fixed regressor bootstrap (Section 3.3) can be used to approximate the p -value for the standard test. First, however, we'll look at another procedure isn't quite as powerful, but is much quicker.

4.2.1 Arranged Autoregression Test

The Arranged Autoregression Test (Tsay (1989)) first runs recursive estimates of the autoregression with the data points added, not in the conventional time series order, but in the order of the test threshold variable. Under the null of no break, the residuals should be fairly similar to the least squares residuals, so there should be no correlation between the recursive residuals and the regressors, and Tsay shows that under the null, an exclusion test for the full coefficient vector in a regression of the recursive residuals on the regressor set is asymptotically a chi-square (or approximated by a small-sample F).¹ If there is a break, and you have the correct threshold variable, then the recursive estimates will be expected to be quite different for the entries with low values of the threshold than they will be for the high values, and we would likely see some correlation in regressing the recursive residuals on the regressors, which would show as a significant exclusion test. In effect, this does the testing procedure in reverse order—rather than do the standard regression first and then see if there is a correlation between the residuals and regressors across subsamples, this does the “subsamples” first, and sees if there is a correlation with the full sample.

This is quite simple to do with RATS because the **RLS** instruction allows you to provide an **ORDER** series which does the estimates in the given order, but keeps the original alignment of the entries. This is already implemented in the **@TSAYTEST** procedure, which, if you look at it, is rather short. **@TSAYTEST** can be applied with any threshold series, not just a lag of the dependent variable—use the **THRESHOLD** option to provide the threshold series:

```
@tsaytest(thresh=dur,d=1) dur
# constant dur{1 to 4}
```

which produces:

TSAY Arranged Autoregression Test F(5,627) 3.071 Signif 0.010
--

4.2.2 Direct Threshold Tests

The direct test for the threshold can be done with either of two procedures. **@THRESHTEST** is the more general procedure, which can be used for any linear regression with any threshold series. Both of these use the fixed regressor bootstrap to approximate the p -values.

```
@threshtest(thresh=dur,d=1,nreps=500) dur
# constant dur{1 to 4}
```

¹Note that this isn't the same as the conventional regression F statistic, which doesn't test the intercept. It's a test on the full vector, including the constant.

Test of Null of No Threshold Against Alternative of Threshold Under Maintained Assumption of Homoskedastic Errors	
Dependent Variable	DUR
Threshold Variable	DUR{1}
Observations	637
From	1960:06
To	2013:06
Maximum F-Test	5.2149
Achieved at	0.0250
Bootstrap Replications	500
Bootstrap P-Value	0.0020

The other is the `@TAR` procedure, which is specifically designed for working with (self-exciting) TAR models. It tests *all* the lags as potential thresholds and reports the one which gives the most significant break test.

```
@tar(p=4,nreps=500) dur
```

For the unemployment rate, this gives a rather unambiguous result supporting a threshold effect:

Threshold Autoregression		
Threshold is DUR{2}=0.0220		
Tests for Threshold Effect use 500 draws		
SupLM	25.746219 P-value	0.002000
ExpLM	9.382281 P-value	0.000000
AveLM	11.404145 P-value	0.004000

The three variations of the test statistic are the maximum LM, the Andrews-Ploberger exponential average (page 35) and the arithmetic average. The `@TAR` procedure chooses lag 2 as the threshold series with the most significant test, however, for further work, we'll use lag 1.

These procedures won't work with the linear attractor model of the Enders-Granger paper since it has a common regressor. In addition, the presence of a_0 in the regression function itself, not just in the regime-selection criterion, means that the function changes, not just at the observed values of y_{t-1} , but also in between—it will be discontinuous at the observed values (since the subsamples change) and continuous but not constant in between. Instead of searching over the observed values of the threshold variable, knowing that the optimum has to be at one of them, we'll use a fine grid of values, hoping that we'll be close to the optimum. (We can't use a variational method like non-linear least squares because of the discontinuities at the observed threshold values).

This generates a grid with 501 points running from the 15%-ile to the 85%-ile of the threshold series. The search is initialized at the sum of squared residuals for a restricted model (symmetrical adjustments).

```

set thresh = spread{1}
*
compute trim=.15
compute startl=%regstart(),endl=%regend()
set copy startl endl = thresh
compute limits=%fractiles(copy,||trim,1-trim||)
compute grid=%seqrage(limits(1),limits(2),501)
compute ssqmin=%rss

```

This next part runs over the test values in the grid, generating the “plus” and “minus” versions of the difference between the threshold series and the test. Given that, the estimates can be done with a **LINREG**. Inside the loop, we find the smallest sum of squares and save the threshold value which gives that.

```

dofor [real] test = grid
  set splus = %max(thresh-test,0.0)
  set sminus = %min(thresh-test,0.0)
  linreg(noprint) ds
  # splus sminus ds{1}
  if %rss<ssqmin
    compute ssqmin=%rss,breakvalue=test
end do time
disp "For Linear Attractor model, best SSQ is " ssqmin $
    " at " breakvalue

```

The results are:

For Linear Attractor model, best SSQ is	101.81867	at	-0.27393
---	-----------	----	----------

which is a bit disappointing, since the restricted model has a sum of squares of just 103.74 so the improvement with the TAR model is fairly minor. The F or LR tests for asymmetry have the typically non-standard distributions, but here they end up being insignificant at *conventional* levels. However, for illustration, we’ll stick with this model.

4.3 Forecasting

A TAR model is a self-contained dynamic model for a series. However, to this point, there has been almost no difference between the analysis with a truly exogenous threshold and a threshold formed by lags of the dependent variable—whether the threshold is endogenous or exogenous, for estimation purposes it’s observable. Once we start to forecast, that’s no longer the case—we need the link between the threshold and the dependent variable to be included in the model.

While the branches may very well be (and probably are) linear, the connection isn’t, so we need to use a non-linear system of equations. For the unemployment series, the following estimates the two branches, given the

identified break (computed using the `@THRESHTEST` procedure, which defines `%%BREAKVALUE` as the value giving the most extreme break test statistic—`BREAK1` is a copy of that).

```
linreg(smpl=dur{1}<=break1,frml=branch1) dur
# constant dur{1 to 4}
compute rss1=%rss,ndf1=%ndf
linreg(smpl=dur{1}>break1,frml=branch2) dur
# constant dur{1 to 4}
compute rss2=%rss,ndf2=%ndf
compute seesq=(rss1+rss2)/(ndf1+ndf2)
```

The forecasting equation is created by gluing these two branches together with:

```
frml(variance=seesq) tarfrml dur = $
    %if(dur{1}<=break1,branch1,branch2)
```

The following is for accumulating back the unemployment rate from the change, and combines the non-linear equation plus the identity into a model:

```
frml(identity) urid unrate = unrate{1}+dur
group tarmodel tarfrml urid
```

Because of the non-linearity of the model, the minimum mean square forecast can't be computed using the point values done by `FORECAST`, but need the average across simulations.

```
compute fstart=2009:1,fend=2010:06
set urfore fstart fend = 0.0
compute ndraws=5000
do draw=1,ndraws
    simulate(model=tarmodel,from=fstart,to=fend,$
        results=sims,noprint)
    set urfore fstart fend = urfore+sims(2)
end do draw
set urfore fstart fend = urfore/ndraws
```

This produces Figure 4.4, which also includes actual data through the end of 2008.

This next estimates the rate spread model at the chosen break value, and converts it into a non-linear `FRML` using an explicit reference to `spread1` as the threshold. (B rather than `%BETA` is used for the coefficients so the `FRML` won't be damaged if you do another estimation later). As with the unemployment model, we have a separate identity in the model which accumulates the changes in the spread to create the spread itself.

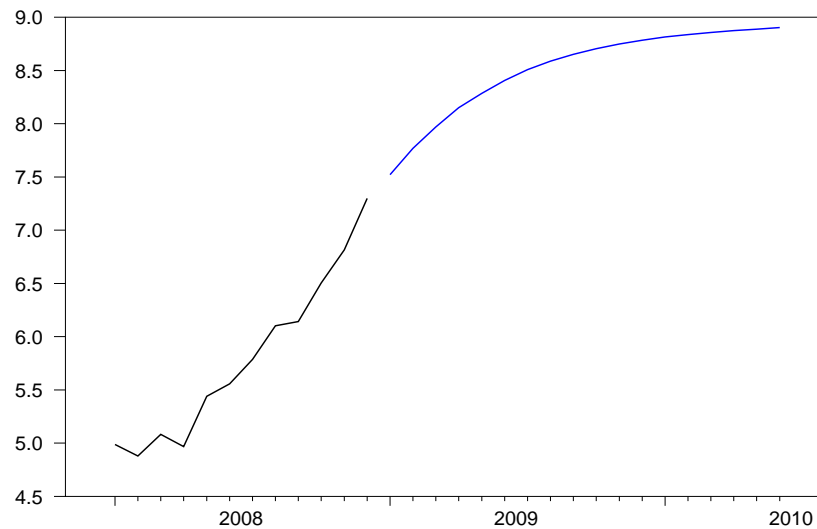


Figure 4.4: Unemployment Forecast from TAR Model

```

set splus = %max(thresh-breakvalue,0.0)
set sminus = %min(thresh-breakvalue,0.0)
linreg(print) ds
# splus sminus ds{1}
*
compute b=%beta
frml adjust ds = b(3)*ds{1}+$
                b(1)*%max(spread{1}-breakvalue,0.0)+$
                b(2)*%min(spread{1}-breakvalue,0.0)
frml(identity) spreadid spread = spread{1}+ds
*
group latarmodel adjust spreadid

```

This does the forecasts by the same process, here starting at the end of the sample, where the spread is relatively high (Figure 4.5). Notice that the forecasts are converging to roughly .85, *not* the estimated “attractor” value of -0.27393. Both branches of the model push the process towards the attractor, but the “minus” branch has a substantially more negative coefficient (-.286 vs -.065). If the residual standard deviation were very small, the mean would indeed converge to the attractor. Here, however, it’s nearly .9. Shocks which push the process below the attractor are much more quickly reversed than shocks which push the process above it, so the process mean is above the attractor (fairly substantially so). How *far* above it depends upon the variance, which is another major difference with linear models, where the variance doesn’t affect the mean at all.

```

compute ndraws=5000
compute nsteps=60
compute istart=1994:2,iend=istart+nsteps-1
set highspread istart iend = 0.0
do draw=1,ndraws
  simulate(model=latarmodel,from=istart,to=iend,$
    results=sims,cv=%seesq)
  set highspread istart iend = highspread+sims(2)
end do draw
set highspread istart iend = highspread/ndraws
graph(footer="Forecasts from High Spread Initial Conditions") 2
# spread istart-12 istart-1
# highspread

```

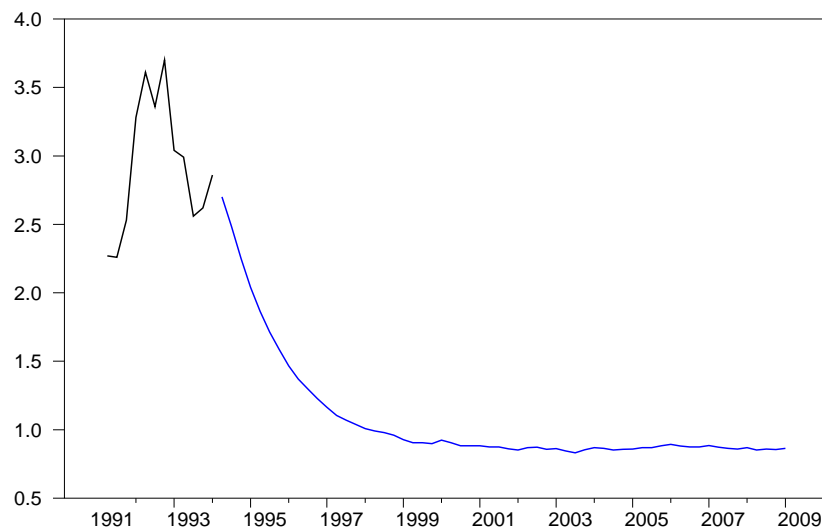


Figure 4.5: Forecasts of Spread from Linear Attractor TAR

4.4 Non-linear Impulse Responses

All threshold models are non-linear. As a result, there is no single “impulse response function”. For a linear model (like a VAR), the effect of superimposing a shock is the same regardless of the past values for the data—the response to a linear combination of initial shocks is the same linear combination of responses from the component shocks. Now, in some ways, this is unrealistic—the true response of the economy to a 100 point shock in the interest rate is unlikely to be anything like 100 times the response to a 1 point shock, that is, the linearity of a standard VAR is only approximate and doesn’t extend to atypical behavior. But for a *non-linear* model, the effects of a shock can be quite different even with historically typical shock sizes in historically typical situations.

The linear impulse response function is computed by zeroing out the initial conditions and simulating the model with a fixed set of first period shocks,

whether unit shocks or some other pattern. Neither part of this is going to be a proper practice with a TAR or similar model. First, the response, in general, will be very sensitive to the initial conditions—if we’re far from the threshold in either direction, any shock of reasonable size will be very unlikely to cause us to shift from one branch to the other, so the differential effect will be whatever the dynamics are of the branch on which we started. On the other hand, if we’re near the threshold, a positive shock will likely put us into one branch, while a negative shock of equal size would put us into the other, and (depending upon the dynamics), we might see the branch switch after a few periods of response.

The *non-linear impulse response* is computed by simulating the model with and without a fixed initial shock, and averaging the difference between those across a large number of simulations. This is often called the *generalized impulse response* as the definition is from Koop et al. (1996), however, we’ll use the term non-linear rather than generalized as the latter term is related more specifically to the implications of the technique to VAR’s rather than non-linear models.

Non-linear IRF’s depend upon the initial conditions. Quite a few different methods have been proposed to deal with that, and you should never simply say that you did non-linear IRF’s without being clear about this. With few exceptions, the initial conditions are based upon histories observed in the actual data. With these you can:

1. Pick specific entries which have properties of interest.
2. Average across all possible histories, which will give an “average” response. This can either be done as a simple average, or by bootstrap draws from all possible histories.
3. Similar to #2, but restricting the histories to those which have certain properties. This will give you an average response for that subset of entries.

#2 is the closest to the IRF in a linear model, since you’re not singling out any special conditions. However, it does lose some of the special nature of the non-linear model.

There’s also a question of whether you should use Gaussian draws or bootstrapping to do the simulations—the difference between the two should not be all that great, and Gaussian draws are generally substantially simpler. And, because the shape (and not just the size) of the IRF depend upon the size and sign of the shocks, there are different ways to present that information—you can show responses to + and - shocks, or + shocks of different scales, or even + and - shocks of different scales. With the shocks of different scales, one possibility is to divide those through by the size of the shock, to allow a more direct comparison of the *shapes*. Again, it’s important to document exactly what you

do, and it's important to read information very carefully in papers to see what was done (assuming the paper is careful to document it).

Computing non-linear impulse responses is very similar to the forecast procedure, but the random draws must be controlled better, so, instead of **SIMULATE**, you use **FORECAST** with `PATHS`.

For the spread model, this computes the non-linear IRF to a positive unit shock (the `delta` variable) based on 1974:3, where the spread is large and negative. When doing non-linear IRF's, it's important to use shock sizes that are reasonable for the model—here, the unit shock isn't just a convenient number, but also a typical size.

```
compute ndraws=5000
compute nsteps=60
compute stddev=sqrt(%seesq)
compute delta=1.0
compute istart=1974:4,iend=istart+nsteps-1
set nlirf istart iend = 0.0
*
do draw=1,ndraws
  set shocks = %ran(stddev)
  forecast (paths,model=latarmodel,from=istart,to=iend,$
    results=basesims)
  # shocks
  compute shocks(istart)=delta
  forecast (paths,model=latarmodel,from=istart,to=iend,$
    results=sims)
  # shocks
  set nlirf istart iend = nlirf+(sims(2)-basesims(2))
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for Spread at 1974:3")
# nlirf istart iend
```

The first **FORECAST** will be identical to what you would get with **SIMULATE**. The second replaces the first period shock with a shock of a specific size. The NL-IRF's at two different sets of initial conditions are shown in Figures 4.6 and 4.7 (1974:3 has a historically low value of the spread and 1994:1 a historical high value). Remember that these are *not* forecasts—they are the differences between forecasts with and without the added initial +1 shock to the spread. With a stationary process, these will have to converge to zero.

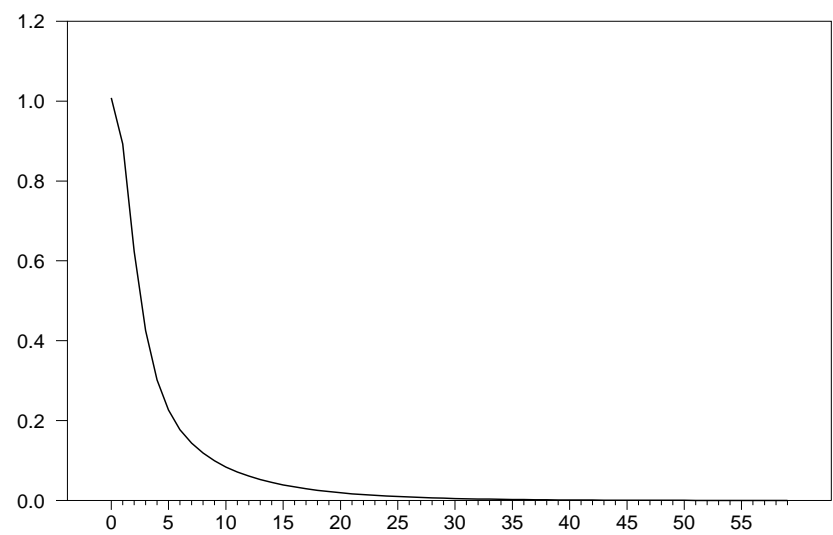


Figure 4.6: NL-IRF for Spread at 1974:3

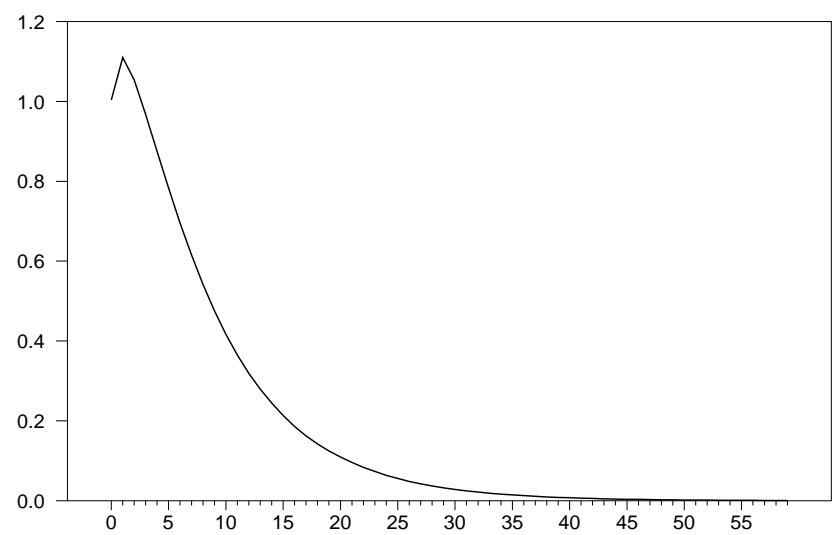


Figure 4.7: NL-IRF for Spread at 1994:1

4.5 Tips and Tricks

Rounding the Threshold Series

You may need to be careful creating a threshold series such as the difference in the unemployment rate or interest rate differentials. The (headline) U.S. unemployment rate is reported at just one decimal digit, and it's a relatively slow-moving series. As a result, almost all the values for the difference are one of -.2, -.1, 0, .1 or .2. However, in computer arithmetic, all .1's are not the same—due to machine rounding there can be several values which disagree in the 15th digit when you subtract. (Fortunately, zero differences will be correct). Because the test in (4.1) has a sharp cutoff, it's *possible* for the optimal threshold to come out in the middle of (for instance) the .1's, since some will be (in computer arithmetic) larger than others. While this is unlikely (since the rounding errors are effectively random with respect to time), you can protect against it by using the `%ROUND` function to force all the similar values to map to exactly the same result. We're using a different reporting of the unemployment rate which uses three (rather than one) decimal digits, so it doesn't have this issue. The interest rate spread is reported at two digits, and takes many more values, so while we include the rounding in our program, it is quite unlikely that rounding error will cause a problem. This forces all two digit differences to have exactly the same value:

```
set spread = %round(r_10-r_short,2)
```

Example 4.1 TAR Model for Unemployment

This is an example of a SETAR model applied to the U.S. unemployment rate series. It is based upon an example from Enders (2015).

```
seed 550103
*
calendar(m) 1960:1
open data unrate.xls
data(format=xls,org=columns) 1960:01 2013:6
*
graph(footer="The U.S. Unemployment Rate",vlabel="Percent")
# unrate
*
@bjident(diffs=1) unrate
*
* Note well that this data set has unemployment calculated to three
* decimal places, not the "headline" US unemployment numbers which
* just use one. At one decimal place, there are too few distinct
* values to use the difference as a threshold variable (almost all
* differences are -.2, -.1, 0, .1 or .2).
*
set dur = unrate-unrate{1}
*
@arautolags(crit=hq) dur
@arautolags(crit=bic) dur
*
* Estimate the linear AR model suggested by @ARAutoLags
*
linreg dur
# constant dur{1 2 3 4}
*
* Simulate and graph an example with the linear AR model
*
uforecast(simulate) sdur %regstart() %regend()
*
* Transform back to levels from differences
*
set sur %regstart() %regend() = $
    %if(t==%regstart(),unrate{1},sur{1})+sdur
graph(footer="Simulated UR with Linear AR")
# sur
*
* Do Tsay arranged regression test
*
@tsaytest(thresh=dur,d=1) dur
# constant dur{1 to 4}
@threshtest(thresh=dur,d=1,nreps=500) dur
# constant dur{1 to 4}
compute break1=%breakvalue
*
* Do TAR test
*
```

```

@tar(p=4,nreps=500) dur
*
* Estimate the two branches and define equations
*
linreg(smpl=dur{1}<=break1,frml=branch1) dur
# constant dur{1 to 4}
compute rss1=%rss,ndf1=%ndf
linreg(smpl=dur{1}>break1,frml=branch2) dur
# constant dur{1 to 4}
compute rss2=%rss,ndf2=%ndf
compute seesq=(rss1+rss2)/(ndf1+ndf2)
*
* Define the forecasting model
*
frml(variance=seesq) tarfrml dur = $
    %if(dur{1}<=break1,branch1,branch2)
frml(identity) urid unrate = unrate{1}+dur
group tarmodel tarfrml urid
*
* Compute average across simulations
*
compute fstart=2009:1,fend=2010:06
set urfore fstart fend = 0.0
compute ndraws=5000
do draw=1,ndraws
    simulate(model=tarmodel,from=fstart,to=fend,$
        results=sims,noprint)
    set urfore fstart fend = urfore+sims(2)
end do draw
set urfore fstart fend = urfore/ndraws
graph(footer="Out-of-sample forecasts") 2
# unrate fstart-12 fstart-1
# urfore
*
* Average "impulse response"
* We need to simulate over the forecast range, but also need to
* control the initial shock. The results of this will depend upon
* the starting point (ISTART).
*
* A 0.25 shock is not atypical at the start of a recession.
*
compute delta=0.25
compute stddev=sqrt(seesq)
compute ndraws=5000
compute nsteps=36
*
compute istart=2009:1,iend=istart+nsteps-1
set nlirf istart iend = 0.0
*
do draw=1,ndraws
    *
    * Do a simulation with fully randomized shocks. These are
    * controlled so we can reuse them.
    *

```

```

set shocks istart iend = %ran(stddev)
forecast(paths,model=tarmodel,from=istart,to=iend,results=basesims)
# shocks
*
* Do a second simulation using the previous shocks beyond the
* initial period and the controlled shock in the first period of
* forecast.
*
compute shocks(istart)=delta
forecast(paths,model=tarmodel,from=istart,to=iend,results=sims)
# shocks
*
* Add the difference between the two simulations to the NLIRF
* series. The unemployment rate itself is the dependent variable
* of the second equation.
*
set nlirf istart iend = nlirf+(sims(2)-basesims(2))/delta
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for unemployment at "+%datelabel(istart))
# nlirf
*
compute stddev=sqrt(%seesq)
clear nlirf
*
* Same thing, different initial time period
*
compute istart=1984:1,iend=istart+nsteps-1
set nlirf istart iend = 0.0
do draw=1,ndraws
    set shocks istart iend = %ran(stddev)
    forecast(paths,model=tarmodel,from=istart,to=iend,results=basesims)
    # shocks
    compute shocks(istart)=delta
    forecast(paths,model=tarmodel,from=istart,to=iend,results=sims)
    # shocks
    set nlirf istart iend = nlirf+(sims(2)-basesims(2))/delta
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for unemployment at "+%datelabel(istart))
# nlirf istart iend

```

Example 4.2 TAR Model for Interest Rate Spread

This is an example of a threshold autoregressive model with a restriction to require a common coefficient between the regimes. This makes analysis more difficult because the two regimes have to be estimated together. This is based upon Enders & Granger (1998).

open data granger.xls

```

calendar(q) 1958:1
data(format=xls,org=columns) 1958:01 1994:01 date r_short r_10
*
* The %ROUND function is used to make sure all spreads which should
* be the same actually are. If you don't use this, various
* differences which should be .01 will come out slightly different
* due to round off error.
*
set spread = %round(r_10-r_short,2)
graph(footer="Interest rate spread")
# spread
*
* ADF Test. @ADFAutoSelect picks either 0 or 1 augmenting lags for
* most of the criteria. The test statistic shown on @ADFAutoSelect
* is slightly different from that on @DFUnit with the same number
* of lags because the first procedure uses a common range for all
* lags tested.
*
@adfautoselect(print,det=constant) spread
@dfunit(lags=1,det=constant) spread
*
* RESET tests of the ADF regression.
*
set ds = spread-spread{1}
*
* This is equivalent to the restricted form of the model (with
* "symmetrical adjustment").
*
linreg ds
# constant spread{1} ds{1}
*
* Do RESET tests
*
@regreset(h=3)
@regreset(h=4)
*
set thresh = spread{1}
*
compute trim=.15
compute startl=%regstart(),endl=%regend()
set copy startl endl = thresh
compute limits=%fractiles(copy,||trim,1-trim||)
compute grid=%seqrangle(limits(1),limits(2),501)
*
compute ssqmin=%rss
dofor [real] test = grid
    set splus = %max(thresh-test,0.0)
    set sminus = %min(thresh-test,0.0)
    linreg(noprint) ds
    # splus sminus ds{1}
    if %rss<ssqmin
        compute ssqmin=%rss,breakvalue=test
end do time
disp "For Linear Attractor model, best SSQ is " ssqmin $

```

```

" at " breakvalue
*
* Further analysis of the model.
* Generate a self-contained model of the process, estimated at the
* least squares fit.
*
set splus = %max(thresh-breakvalue,0.0)
set sminus = %min(thresh-breakvalue,0.0)
linreg(print) ds
# splus sminus ds{1}
*
* To do any forecasting, we need to write this in terms of
* spread{1}, which will be generated by the model.
*
compute b=%beta
frml adjust ds = b(3)*ds{1}+$
                b(1)*%max(spread{1}-breakvalue,0.0)+$
                b(2)*%min(spread{1}-breakvalue,0.0)
frml(identity) spreadid spread = spread{1}+ds
*
group latarmodel adjust spreadid
*
* Starting from end of sample, where spread is relatively high
*
compute ndraws=5000
compute nsteps=60
compute istart=1994:2,iend=istart+nsteps-1
set highspread istart iend = 0.0
do draw=1,ndraws
    simulate(model=latarmodel,from=istart,to=iend,$
            results=sims,cv=%seesq)
    set highspread istart iend = highspread+sims(2)
end do draw
set highspread istart iend = highspread/ndraws
graph(footer="Forecasts from High Spread Initial Conditions") 2
# spread istart-12 istart-1
# highspread
*
* Starting from 1974:3, where the spread is large negative.
*
compute istart=1974:4,iend=istart+nsteps-1
set lowspread istart iend = 0.0
compute ndraws=5000
do draw=1,ndraws
    simulate(model=latarmodel,from=istart,to=iend,$
            results=sims,cv=%seesq)
    set lowspread istart iend = lowspread+sims(2)
end do draw
set lowspread istart iend = lowspread/ndraws
graph(footer="Forecasts from Low Spread Initial Conditions") 2
# spread istart-12 istart-1
# lowspread
*
* Non-linear impulse responses

```

```

* We need to simulate over the forecast range, but also need to
* control the initial shock. The results of this will depend upon
* the starting point.
*
compute ndraws=5000
compute nsteps=60
compute stddev=sqrt(%seesq)
compute delta=1.0
compute istart=1974:4,iend=istart+nsteps-1
set nlirf istart iend = 0.0
*
do draw=1,ndraws
  set shocks = %ran(stddev)
  forecast(paths,model=latarmodel,from=istart,to=iend,$
    results=basesims)
  # shocks
  compute shocks(istart)=delta
  forecast(paths,model=latarmodel,from=istart,to=iend,$
    results=sims)
  # shocks
  set nlirf istart iend = nlirf+(sims(2)-basesims(2))
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for Spread at 1974:3")
# nlirf istart iend
*
compute istart=1994:2,iend=istart+nsteps-1
clear nlirf
set nlirf istart iend = 0.0
do draw=1,ndraws
  set shocks istart iend = %ran(stddev)
  forecast(paths,model=latarmodel,from=istart,to=iend,$
    results=basesims)
  # shocks
  compute shocks(istart)=delta
  forecast(paths,model=latarmodel,from=istart,to=iend,$
    results=sims)
  # shocks
  set nlirf istart iend = nlirf+(sims(2)-basesims(2))
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for Spread at 1994:1")
# nlirf

```

Threshold VAR/Cointegration

The extension of threshold models to multivariate settings is largely a matter of replacing univariate likelihoods with multivariate ones. Threshold error correction models are probably the most useful of the three basic techniques examined here, since the equilibrium condition is an obvious candidate for a threshold variable.

5.1 Threshold Error Correction

The first paper to address the question of *threshold cointegration* is Balke & Fomby (1997), though it's probably more accurate to describe this as threshold error correction. Their paper¹ analyzed the spread between the Federal Funds rate and the discount rate; thus, the cointegrating vector is considered to be known in advance as $(1, -1)$. The case where the cointegrating vector isn't known and must be estimated is much more complicated, and will be addressed in Section 5.3.

The point raised by Balke and Fomby is that the Fed would be unlikely to allow arbitrary spreads between those two rates. The Fed controls the discount rate, but exercises only indirect control over the funds rate. They would likely make a policy reaction to reduce the gap if the spread got either too large or too small, but would be less likely to intervene in a band closer to a normal spread. The expected behavior would be that the error correction term would be small (effectively zero) in the center band, but non-zero in the high and low bands.

Most of the analysis is a TAR model on the spread itself. The authors do a Tsay threshold test (Section 4.2.1) with several different delays, and in both the direct and reversed direction. Because the `@TSAYTEST` procedure allows arbitrary threshold series (not just lags of the dependent variable), the reversed test is easily done with:

```
set thresh = -spread{d}
@tsaytest(thresh=thresh) spread
# constant spread{1 2}
```

All the test statistics are significant, but the two with $d = 1$ are by far the strongest, so the remaining analysis uses the lagged spread as the threshold variable.

¹The empirical application is only in the working paper version, not the journal article.

Under the presumed behavior, the spread should show stationary behavior in the two tails and unit root behavior in the middle portion. To analyze this, the authors do an arranged Dickey-Fuller test—a standard D-F regression, but with the data points added in threshold order. Computing this requires the use of the **RLS** instruction, saving both the history of the coefficients and of the standard errors of coefficients to compute the sequence of D-F *t*-tests:

```
set dsread = spread-spread{1}
set thresh = spread{1}
rls(order=thresh, cohstory=coh, sehstory=seh) dsread
# spread{1} constant dsread{1}
*
set tstats = coh(1)(t)/seh(1)(t)
```

This is done with the recursions in each direction. The clearer of the two is with the threshold in increasing order (Figure 5.1). Once you have more than a handful of data points in the left tail, the D-F tests very clearly reject unit root behavior, a conclusion which reverses rather sharply beginning around a threshold of 1, though where exactly this starts to turn isn't easy to read off the graph since the data in use at that point are overwhelmingly in the left tail—the observed values of the spread are mainly in the range of -.5 to .5.

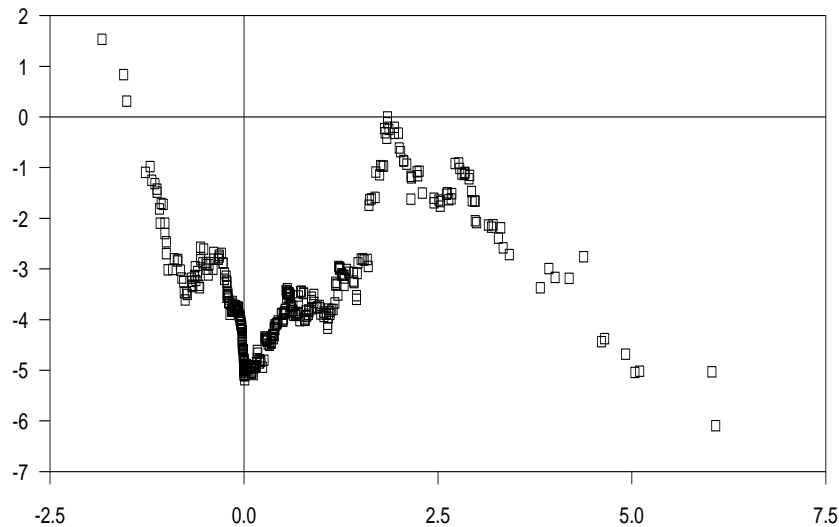


Figure 5.1: Recursive D-F T-Statistics

Based upon this graph, the authors did a (very) coarse grid search for the two threshold values in a two-lag TAR on the spread using all combinations of a lower threshold in $\{-.2, -.1, 0, .1, .2\}$ and an upper threshold in $\{1.6, 1.7, 1.8, 1.9, 2.0\}$, getting the minimum sum of squares at -.2 and 1.6. A more complete grid search can be done fairly easily and quickly at modern computer speeds. As is typical of (hard) threshold models, the sum of squares function is flat for thresholds between observed values, so the search needs

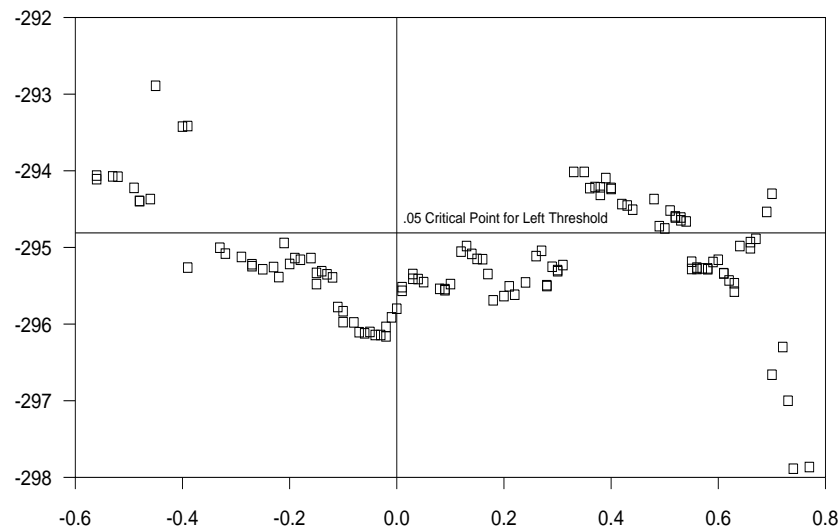


Figure 5.2: Log Likelihood as function of left threshold given right threshold

only to look at those observed values as potential break points. A more comprehensive search finds the optimal breaks as $-.45$ and 1.45 . If we look at the cross section of the log likelihood surface with the upper threshold fixed at the optimizing 1.45 (Figure 5.2), you can see that the function is not just discontinuous, but more generally quite ill-behaved. This is fairly typical since each new threshold value generally shifts only one data point between two of the partitions—if that data point happens to be very influential in one or both of the subsamples, it could cause a temporary blip up or down to the overall sum of squares.

One thing to note is that the overall range in log likelihoods is actually quite small—the top to bottom range is only about 5 and almost $1/3$ of the values (across a broad range) have log likelihoods high enough that they would be acceptable at the $.05$ level in a likelihood ratio test for a specific value for the left threshold given the right threshold.² Thus, while the data strongly support a two threshold model, they don't so strongly identify where they are. The analysis using the techniques from the paper are shown in Example 5.1.

An alternative to doing the TAR model on the spread is to choose breaks using the multivariate likelihood for the actual VECM model. This is also done easily using the instruction **SWEET**. The following does the base multivariate regression of the two differenced variables on 1, one lag of the differenced variables and the lagged spread:

²And since the right threshold of 1.45 isn't even necessarily the constrained maximizer for any given value of the left threshold, even more left threshold values are likely to be acceptable if we did the required number-crunching to do a formal likelihood ratio test for each.

```

sweep
# dff ddr
# constant dff{1} ddr{1} spread{1}
compute loglbest=%logl

```

%LOGL is the log likelihood of the systems regression. With groupings, it does separate systems estimates, aggregating the outer product of residuals to get an overall likelihood:

```

sweep(group=(thresh>=tvalues(lindex))+(thresh>tvalues(uindex)))
# dff ddr
# constant dff{1} ddr{1} spread{1}

```

The estimation using this gives the (much) tighter range of .63 to 1.22 as the center subsample.

In order to do any type of forecasting calculation with this type of model, we need to define identities to connect the five series: the two rates, the spread and the changes. And we need to define formulas for the two rates of change which switch with the simulated values of the threshold variable.

With the lower and upper threshold values in the variables of the same name, the following formula will evaluate to 1, 2 or 3 depending upon where the value of SPREAD{1} falls:

```

dec frml[int] switch
frml switch = 1+fix((spread{1}>=lower)+(spread{1}>upper))

```

Note that you must use SPREAD{1} in this, not a fixed series (THRESH) generated from SPREAD that we can use during estimation—when we do simulations SPREAD is no longer just data, but is part of the simulation.

The most flexible way to handle the switching functions for the changes is to define a RECT[FRML] with rows for equations and columns for partitions. Using separate formulas for each case (rather than using the same equation changing the coefficients) makes it easier to use different forms. In this case, we *are* using the same functional form for all three branches, but that might not be the best strategy.

```

system(model=basevecm)
variables dff ddr
lags 1
det constant spread{1}
end(system)
*
dec rect[frml] tvecfrml(2,3)
do i=1,3
    estimate(smpl=(switch(t)==i))
    frml(equation=%modeleqn(basevecm,1)) tvecfrml(1,i)
    frml(equation=%modeleqn(basevecm,2)) tvecfrml(2,i)
end do i

```

The identities are straightforward:

```

frml(identity) ffid fedfunds = fedfunds{1}+dff
frml(identity) drid mdiscrt = mdiscrt{1}+ddr
frml(identity) spid spread = fedfunds-mdiscrt

```

With the work already done above, the switching formulas are now quite simple:

```

frml dffeq dff = tvecfrml(1,switch(t))
frml ddreq ddr = tvecfrml(2,switch(t))

```

The working model is then constructed with:

```

group tvecm dffeq ddreq ffid drid spid

```

An eventual forecast function can be generated using the **FORECAST** instruction. This is intended to show one possible path of the system, not to give a minimum mean-square error forecast.

```

forecast(model=tvecm,from=1981:1,steps=40,results=eff)
graph(footer=$
    "Eventual Forecast Function for SPREAD, starting at 1981:1")
# eff(5)
graph(footer=$
    "Eventual Forecast Function for Change in DR, starting at 1981:1")
# eff(2)

```

The full code for the joint analysis of the model is in example 5.2.

5.2 Threshold VAR

Unlike the error correction model, the full threshold VAR is a systems estimator with more than one dependent variable. From the start, we should note

that there is nothing about either the testing or estimation that requires that the model actually be a VAR (in the sense that the explanatory variables are only lags of the dependent variables plus deterministics), and there is also no requirement that the threshold variable be a function of one of the dependent variables—it could be exogenous to the model. Where the structure of the equations and the threshold variables would matter is if you want to do impulse responses or forecasts, that is, where you need to simulate *out-of-sample* behavior. Estimation and testing depend only upon observed sample data. The model (with a single threshold and thus two regimes) is most conveniently written:

$$\mathbf{Y}_t = \begin{cases} (I \otimes \mathbf{X}_t)\beta_1 + \mathbf{u}_t, E\mathbf{u}_t\mathbf{u}'_t = \Sigma_1 & \text{if } Z_{t-d} \leq \gamma \\ (I \otimes \mathbf{X}_t)\beta_2 + \mathbf{u}_t, E\mathbf{u}_t\mathbf{u}'_t = \Sigma_2 & \text{if } Z_{t-d} > \gamma \end{cases} \quad (5.1)$$

As written, this has identical explanatory variables in each equation and in each regime. Can that be relaxed? Yes. However, if the explanatory variables differ across equations within a regime, you (at least theoretically) would have to estimate using SUR rather than least squares equation by equation (or more conveniently the **SWEEP** instruction). As written, this allows for different covariance matrices between the regimes; however, that's not required and you can restrict them to a single common covariance matrix. It's also possible to have coefficients fixed and only *the covariance matrix* switching. Note that, given γ , the estimates of the β 's don't depend upon the Σ 's if the explanatory variables are the same across equations, as the usual results for multivariate regressions apply—the assumptions regarding Σ only matter for computing the likelihoods.

As written, this has all coefficients differing between regimes. As with the linear attractor model in Example 4.1, if there are coefficients which are common between the two regimes, you can only estimate using a joint estimator with dummied-out explanatory variables. That's not especially difficult if the Σ 's are fixed between the regimes, but is much more complicated if they are not.

As written, all equations are subject to the same threshold. Loosening that (other than, for instance, having an equation which doesn't switch at all, which is addressed in the previous paragraph) is unlikely to produce a reasonable model—it's highly unlikely that you would have a set of equations on related data, each of which would be a two-branch threshold model, but each with a different threshold. It would be quite difficult to do a joint search across differing threshold values for multiple equations using a systems estimator, so you would probably need to just do equation-at-a-time estimates. The examples here will use the model as written in (5.1). Both will be actual VAR's.

5.2.1 Tsay(1998) interest rates example

The first (Example 5.3) is from Tsay (1998). This does estimating and testing only. This is his U.S. interest rates example. The two dependent variables are

yields on three-month and three-year U.S. treasury securities. The threshold variable is a lag of a three-month moving average on the spread between them. This clearly has some similarities to Example 5.1, except that neither of the yields is a policy variable, so the “control” argument wouldn’t apply. The use of a moving average of the spread rather than just the lagged spread itself is to get a stronger signal by not reacting to possibly short-term movements in one of the yields—it will probably require some experimentation to determine what’s appropriate in a given situation.³ To test for the presence of a break, Tsay uses a multivariate extension of the arranged autoregression test (Section 4.2.1). The number of lags in the VAR is chosen by AIC as 7. This is done using a fixed-regime, so it is unlikely to be too *short*. The arranged regression test is done by doing recursive least squares on the two equations ordered according to the threshold variable. These are done conditioning on a certain number of initial observations (he tries 50 and 100) rather than just the minimal number required to do the regression (which would here be 15, for 7 lags on each of 2 variables plus the constant) and trying different delays on the spread from 1 to 7. Given the number of conditioning observations (M_0) and the threshold delay (D), this does the test calculation:

```
set thresh = ssread{d}
*
rls(noprint,order=thresh,condition=m0) g3year / rr3year
# constant g3year{1 to p} g3month{1 to p}
rls(noprint,order=thresh,condition=m0) g3month / rr3month
# constant g3year{1 to p} g3month{1 to p}
*
order(ranks=rr) thresh %regstart() %regend()
*
sweep(smpl=rr>m0,series=tresids)
# rr3year rr3month
# constant g3year{1 to p} g3month{1 to p}
*
ratio(mcorr=%nreg,degrees=k*%nreg,noprint)
# rr3year rr3month
# tresids
disp "D=" d "m0=" m0 @16 "C(d)=" *.## %cdstat $
      @28 "P-value" #.##### %signif
```

The recursive residuals are generated for each equation and then those are regressed on the explanatory variables. As in the univariate case, if there is no threshold effect, there should be no correlation between the recursive residuals and the explanatory variables if the reordering has no effect (that is, there is no difference between the regressions based upon the threshold), and the test

³Tsay doesn’t describe how he chooses it, though an extension of the search procedure used for choosing the threshold delay could be utilized.

statistic has an asymptotic χ^2 distribution under the null. The largest test statistic occurs with a delay of 1 and a conditioning sample of 50:

D= 1 m0= 50 C(d)= 65.53 P-value 0.00019

and it's clearly quite significant. Unfortunately, the further analysis of this seems quite poorly done. To estimate the one break model, he uses a grid over values of the threshold running from -.30 to .05. In this case, using a grid based upon empirical values would have required almost exactly the same amount of calculation. Given the values that the average spread series takes, .05 is far too large to be considered—it's larger than the 95%-ile of the series, and with a seven lag VAR, there are almost no degrees of freedom in an estimate partitioned there. A more reasonable upper limit would have been -.05,⁴ but even with that, the best break is at -.05.⁴ The following does the one break model as described in the article:

```
set thresh = sspread{1}
@gridseries(from=-.30,to=.05,n=300,pts=ngrid) rgrid
set aic 1 ngrid = 0.0
*
compute bestaic=%na
*
do i=1,ngrid
  compute rtest=rgrid(i)
  sweep(group=thresh<rtest,var=hetero)
  # g3year g3month
  # constant g3year{1 to p} g3month{1 to p}
  compute aic(i)=-2.0*%logl+2.0*%nregsystem
  if i==1.or.aic(i)<bestaic
    compute bestaic=aic(i),bestbreak=rtest
end do i

*
scatter(footer=$
  "AIC Values for Interest Rates Regression vs Break Point")
# rgrid aic
disp "Best Break is" bestbreak "with AIC" bestaic
```

Note that this uses the VAR=HETERO option on **SWEEP**. That allows the covariance matrix to be different between partitions and computes the likelihood function on that basis. This can also be done with **SWEEP** with just a single target variable. Note, however, that the “arranged regression” testing procedure assumes a fixed covariance matrix.

⁴We discovered (accidentally) that the results in the paper can be reproduced almost exactly by using an incorrect calculation of the likelihood. The error adds a term to the log likelihood which is maximized when the number of data points is roughly the same in each partition.

The double break model can now (with faster computers) be done in a reasonable amount of time using the empirical grid. As in the paper, this uses a somewhat coarser grid, though it's a slightly different one than is described there. Note that the J loop starts at $I+20$ —that forces there to be a gap between the two threshold values so the middle branch will have data with which to run its regression.

```
@gridseries (from=-.30,to=.05,n=80,pts=ngrid) rgrid
*
compute bestaic=%na
do i=1,ngrid-19
  do j=i+20,ngrid
    sweep (group=(thresh<rgrid(i))+(thresh<rgrid(j)),var=hetero)
    # g3year g3month
    # constant g3year{1 to p} g3month{1 to p}
    compute thisaic=-2.0*logl+2.0*nregsystem
    if .not.%valid(bestaic).or.thisaic<bestaic
      compute bestaic=thisaic,bestlower=rgrid(i),bestupper=rgrid(j)
  end do j
end do i
*
disp "Best double break is at" bestlower "and" $
    bestupper "with AIC" bestaic
```

The results for the one and two breaks models are:

Best Break is	0.05000	with AIC	-2805.77116
Best double break is at	-0.09438	and	0.05000 with AIC -2809.17370

Because of the error described above in evaluating the likelihood in the paper, the published results strongly prefer the two break model over the one break, while, as you can see from this, it's only a modest improvement in AIC, and the one break model would be preferred by the more conservative BIC.

5.2.2 Balke(2000) Credit Regimes

Balke (2000) estimates a four variable TVAR with output growth, inflation, the Federal Funds rate and one of several proposed measures of credit market conditions. The threshold variable is a moving average of the credit market variable—the idea is that there is a tight and a normal credit regime, and the dynamics of the economy change depending upon which is in force. This is a much more ambitious paper than Tsay's, which is really a methodological piece, while Balke's is applied. Balke's does testing, estimation and computes non-linear IRF's. (Unfortunately, quite a bit of the technical information was squeezed out of the published paper).

The paper tries three different credit market variables—we'll focus just on the Commercial Paper Rate to T-Bill Rate spread. This uses a two-month moving

average with a one-period lag as the threshold. Different credit variables had different MA lengths⁵ and they also have different signs to determine “tight” vs “normal”. For the `CPBILL` variable, a higher value represents a tight market, as commercial enterprises have to pay a higher premium over the government rate.

Balke’s original RATS program allowed for different lag lengths on each variable in each equation (thus 4×4 possible lag lengths), and estimated the equations using SUR. Since it’s largely infeasible to check on 16 different lag lengths, we will stick with 4 lags on all, which allows use of more efficient estimators both statistically and computationally.

This example is split into two parts: Example 5.4 does estimation and testing and 5.5 does impulse responses. The details on calculation of IRF’s are in Section 5.2.3—that takes the estimated threshold value from the first example as given. Obviously, you could (and in practice probably *should*) combine them into a single running program so there would be no possibility of error in transferring results, but we will break them into parts to allow emphasis on the different tasks that each program is doing.

For estimation and testing, Balke chooses to exclude a fraction `PI` (here .15) of the more extreme threshold values over and above the number needed to run the regressions. Although there are a very large number of parameters across all equations and two regimes, for estimation, the minimum number of entries required within a regime is the $4 \times 4 + 1$ for one equation out of the 4-variable, 4-lag (plus constant) single equation. So, with the added 15%, the minimal regime size allows is about 40 observations total. The original data set has data from 1947 to 1994, but Balke chose to not include the early part of that. The following sets the limits of the working range:

```
compute sstart = 1959:1
compute send   = 1997:3
```

The following copies the CPR-TBill spread to the `CREDIT` variable, and sets the delay (1) and moving average length (2) that will be used for that. A different choice for the credit proxy might require a different moving average length, though probably not a different delay.

```
set credit = cpbill1
compute d = 1
compute malength = 2
```

This sets of the standard VAR and estimates it over the maximal range, given the limits set by `SSTART` and `SEND`:

⁵From the paper, “The lengths of the moving averages were set so that all three credit threshold variables had similar autocorrelation functions”, though it’s not clear what the goal of taking a particular moving average is.

```

system(model=varmodel)
variables dly dlp money credit
lags 1 to maxlag
det constant
end(system)
*
compute rstart=sstart+maxlag,rend=send
estimate(print,resids=u) rstart rend

```

This creates the desired moving average and delay to create the threshold variable from the credit variable and makes a sorted copy of it (in increasing order):

```

filter(type=lagging,span=malength) credit / credthr
*
set copy rstart rend = credthr{d}
order copy rstart rend

```

This determines the range of threshold variable entries which will be considered: excluding the fraction πI plus the number of regressors in an equation in the VAR:

```

compute piskip=fix(pi*%nobs)+%nreg
compute pistart=rstart+piskip,piend=rend-piskip

```

This loops over the usable values of the threshold, and uses **SWEEP** to compute the log likelihood of the model with a break at the test value of **THRESH**. The **VAR=HETERO** option on **SWEEP** allows the variances to be different between regimes.

```

set teststats pistart piend = 0.0
do pientry=pistart,piend
  compute thresh=copy(pientry)
  *
  sweep(var=hetero,group=credthr{d}<thresh) rstart rend
  # %modeldepvars(varmodel)
  # %rlffromeqn(%modeleqn(varmodel,1))
  *
  if %logl>besttest
    compute besttest=%logl,bestthresh=thresh
  compute teststats(pientry)=2.0*(%logl-loglr)
end do pientry

```

As part of the calculation, this checks to see if the log likelihood is better than the best seen so far (which starts at the log likelihood from the single-regime estimates), and saves the likelihood ratio test statistic for break vs no break for each tested breakpoint. This is graphed against the threshold value (Figure 5.3).

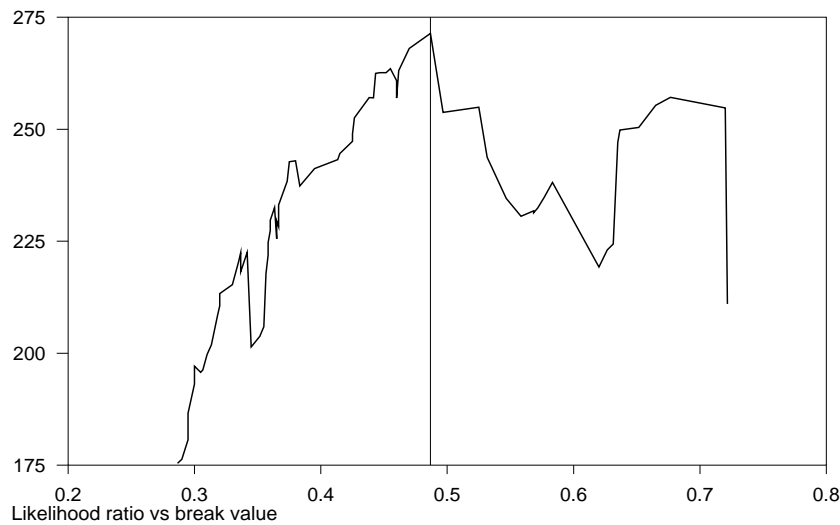


Figure 5.3: Likelihood ratio vs break value

```
disp "Best Threshold Value" bestthresh
scatter(style=lines,hgrid=bestthresh,$
        footer="Likelihood ratio vs break value")
# copy teststats
*
compute suplr=2.0*(besttest-loglr)
disp "LR Test" suplr "Degrees of freedom" %nfree-nfreer
```

The chosen threshold value, a maximum likelihood break test is

Best Threshold Value	0.48667
LR Test	271.40077 Degrees of freedom 78

There are 78 added parameters in the two-regime model, 68 regression coefficients plus 10 extra in the second covariance matrix. Of course, this has a non-standard distribution, so we can't compare this with the usual χ^2 . Balke used the fixed regressor bootstrap (Section 3.3), and we will demonstrate that as well. However:

1. The fixed regressor bootstrap doesn't allow for changes in the variance, only in the regression coefficients, so this won't be able to test for the type of model actually used.
2. Balke's fixed regressor bootstrap scaled each equation by a different independent $N(0, 1)$ at a given t rather than the correct common draw. This is an easy mistake to make and also (interestingly) was made in Hansen & Seo (2002).

As we've seen before, this involves running a loop over the test procedure. For all but the final one, it uses a series of $N(0, 1)$ random numbers and uses that

to scale all the original residuals to create the bootstrapped dependent variables (these are in the `FIDDLED` set of series). For the final one, the original residuals computed under the null are used, which would reproduce the original test statistic if we were using the same test. Instead, this uses **SWEEP** with `VARIANCE=HOMOGENEOUS`, which forces a single covariance matrix to be estimated. (This has no effect on the regime-specific coefficients; just the covariance matrix and the likelihood function).

```
dec vect[series] fiddled(%nvar)
*
compute nboot=500
set avgboot 1 nboot = 0.0
set expboot 1 nboot = 0.0
set supboot 1 nboot = 0.0
*

do reps=1,nboot+1
  set teststats pistart piend = 0.0
  *
  * All residuals are scaled by a common N(0,1) at time t.
  *

  set fiddlef = %ran(1.0)
  do i=1,%nvar
    set fiddled(i) = %if(reps<=nboot,fiddlef*u(i),u(i))
  end do i

  *
  sweep rstart rend
  # fiddled
  # %rlffromeqn(%modeleqn(varmodel,1))
  compute loglr=%logl
  compute besttest=%logl

  *
  do pientry=pistart,piend
    compute thresh=copy(pientry)
    sweep(var=homo,group=credthr{d}<thresh) rstart rend
    # fiddled
    # %rlffromeqn(%modeleqn(varmodel,1))
    *
    * If this is best so far, save it
    *
    if %logl>besttest
      compute besttest=%logl,bestthresh=thresh
      compute teststats(pientry)=2.0*(%logl-loglr)
    end do pientry
```

```

*
sstats(max)  pistart piend teststats>>aqvalue
sstats(mean) pistart piend teststats>>avgvalue $
               exp(.5*(teststats-aqvalue))>>apvalue
compute apvalue=log(apvalue)+.5*aqvalue

if reps<=nboot {
    compute expboot(reps)=apvalue
    compute avgboot(reps)=avgvalue
    compute supboot(reps)=aqvalue
}
end do reps

```

This computes the bootstrapped p -values for the three different forms of the test statistic:

```

sstats(mean) 1 nboot (expboot>apvalue)>>exppvalue $
                  (supboot>aqvalue)>>suppvalue $
                  (avgboot>avgvalue)>>avgpvalue

*
report(action=define)
report(atrow=1,atcol=1) "" "Statistic" "P-value"
report(atrow=2,atcol=1) "Sup" aqvalue suppvalue
report(atrow=3,atcol=1) "Avg" avgvalue avgpvalue
report(atrow=4,atcol=1) "Exp" apvalue exppvalue
report(atcol=2,tocol=2,action=format,width=8)
report(atcol=3,tocol=3,action=format,picture="*.###")
report(action=show)

```

The results (subject to some simulation error) are

	Statistic	P-value
Sup	164.9322	0.258
Avg	121.4247	0.122
Exp	79.2623	0.240

Now the sup value of 164.93 isn't strictly comparable to the 271.4 value seen in the first part since that allows for the switching covariance matrices and this doesn't. The conclusion would be that there is no strong evidence of a difference in the coefficients only, but given that the extra 10 covariance parameters creates a much, much higher likelihood ratio statistic, it may not be unreasonable to conclude that there is a switch in the covariance matrix.

The IRF calculations use the model with the switching covariance matrix, so the threshold value that will be input into that will be the 0.48667 estimated at the beginning.

5.2.3 Impulse Response Functions

As mentioned earlier, there is no difference between a VAR and any other multivariate regression when it comes to testing and estimation—it's when you forecast (or do impulse responses, which is a forecasting exercise), that the connections among the variables matters. You only have a self-contained process if the explanatory variables are lagged dependent variables (or purely deterministic variables, like constant, trend and seasonals) *and* the threshold variable is a function of the (lagged) dependent variables—we have that in this model since the threshold is a moving average of the credit variable. The process for generating IRF's is similar to the one for univariate models in Section 4.4, but with the added complication of being multivariate, which brings in the issue from the VAR literature of how exactly to define what a shock means when the residuals are correlated.

Again, as in Section 4.4, because of the dependence of IRF's on initial conditions, it's important to be clear about how you are choosing these. Balke's paper does the following, which may, or may not, be the best choice in other situations:

1. The IRF's are computed as straight averages across *all* initial conditions in each of the two regimes (tight and normal credit).
2. The IRF's are computed by bootstrapping the observed residuals. Because the two regimes are estimated with different covariance matrices, this requires (jointly) standardizing the observed residuals, bootstrapping those, and “reflating” them based upon the regime to create the bootstrapped residuals.
3. The shocks are orthogonalized by Cholesky factors.

The IRF's are computed in a separate program (Example 5.5). This is designed to do just the responses in *one* of the two regimes. The controls at the top determine how these are calculated.

```
comp upper      =1
comp regimeDesc="Tight Regime"
comp horizon    =15
comp nboot      =500
```

UPPER=1 is to compute the responses conditional on the upper value regime (that is, averaged over the initial conditions in that regime.) REGIMEDESC is the descriptive label of the chosen regime. Change to UPPER=0 (and reset the REGIMEDESC) to get the lower value regime. HORIZON is the number of IRF steps and NBOOT is the number of bootstrap draws (for each set of initial conditions—so this does a very large number of simulations).

The following controls the set of test shocks, which are all in standard-deviation multiples. The second gives the corresponding descriptions for output.

```
compute [vector] shocksizes=||+1.0,+2.0,-1.0,-2.0||
compute [vec[string]] shocksize=||"+1 SD", "+2 SD", "-1 SD", "-2 SD"||
```

This copies in the information determined in the estimation program. These are the choice of credit variable, the moving average length and delay for the threshold variable, and the estimated threshold value.

```
set credit      = cpbill1
comp malength   = 2
comp d          = 1
comp thresh     = 0.48667
```

In order to do any forecasting exercise, we need to “endogenize” the threshold variable so it adjusts as the underlying dependent variable changes. This first creates the threshold variable, then creates an identity FRML which defines it as a linear combination of the CREDIT variable. This is designed to adapt to different values for the MALENGTH.

```
filter(type=lagging,span=malength) credit / credthr
*
compute macoeffs=%fill(malength,1,1.0/malength)
equation(identity,coeffs=macoeffs) threqn credthr
# credit{0 to malength-1}
frml(equation=threqn,identity) thrfrml
```

This defines dummies which define the two regimes:

```
set dup = thrfrml{d}>thresh
set dlo = 1.-dup
```

and this defines the VAR that forms the base model (DEPVARs is defined earlier as a list of the dependent variables):

```
system(model=basevar)
variables depvars
lags 1 to maxlag
det constant
end(system)
```

This estimates the VAR over each of the regimes, defines regime-specific sets of FRML’s for the equations in each, regime-specific sets of residuals and regime-specific covariance matrices.


```

compute rstart=sstart+maxlag,rend=send
estimate(smpl=dup,resids=resup,title="Upper Regime") rstart rend
do i=1,nvar
    frml(equation=%modeleqn(basevar,i)) fitup(i)
end do i
compute vup=%sigma
*

estimate(smpl=dlo,resids=reslo,title="Lower Regime") rstart rend
do i=1,nvar
    frml(equation=%modeleqn(basevar,i)) fitlo(i)
end do i
compute vlo=%sigma

```

By the way, if you want a common covariance matrix instead (different coefficients, but the same covariance matrix), you can use the same process to estimate the coefficients, but the joint covariance matrix would be computing the sum of the %SIGMA times %NOBS for the two subsamples (which would give you the total cross-product of residuals) and dividing by the total number of observations.

These compute the Cholesky factors and their inverses to use in standardizing the residuals for bootstrapping (and also, indirectly, for orthogonalizing the shocks).

```

compute sup =%decomp(vup)
compute siup=inv(sup)
compute slo =%decomp(vlo)
compute silo=inv(slo)

```

Note that any method of producing an (actual) factorization of the covariance matrices can be used instead. However, something like a short- and long-run model would make very little sense, since the long-run behavior of the system isn't determined by the individual regimes, but by the combination, and the long-run constraints are based upon a single moving average representation holding throughout.

This computes the (jointly) standardized residuals. Since we only use these as a VECTOR across variables at T (never as individual series), we define it as a SERIES [VECTOR].

```

dec series[vect] stdu
gset stdu rstart rend = %if(dup,siup*%xt(resup,t),silo*%xt(reslo,t))

```

Note that, while Gaussian simulation doesn't really care what factorization is used for the covariance matrix, there will be slight difference in bootstrapping, as different factorizations can produce slightly different empirical distributions.

This saves the original data, since that will be overwritten during the bootstrap. Saving and restoring this is simpler than creating a whole parallel model for bootstrapping.

```
dec vect[series] data(nvar)
do i=1,nvar
  set data(i) = depvars(i){0}
end do i
```

This defines the switching formulas. Because the bootstrap requires taking the standardized residuals and reflating them using regime-specific covariance matrices, the reflation part has to be incorporated into the formula. Thus TVARF(*i*) is (in effect) an identity given the (still to be created) BOOTRES series.

```
do i=1,nvar
  frml tvarf(i) depvars(i) = %if(thrfrml{d}>thresh,$
    fitup(&i)+%dot(%xrow(sup,&i),bootres),$
    fitlo(&i)+%dot(%xrow(slo,&i),bootres))
end do i
```

The &I's are important in defining these—without them, FITUP(*I*) will evaluate to whatever the value of *I* is *when the FRML is actually used later*. &I forces it to be resolved now, when it's defined, so TVARF(1) will use FITUP(1) and %XROW(SUP,1) and TVARF(2) will use FITUP(2) and %XROW(SUP,2), etc. Since this is the key FRML which actually does most of the work, we'll go through it carefully. Each dependent variable has different FRML's that describe its evolution in the different regimes: FITUP for the upper regime, and FITLO for the lower one. It also has a shock which we need to add in, which has a different distribution in each regime. BOOTRES(*T*) will be a full *n*-vector, which (theoretically) has mean zero and an identity covariance matrix. The overall *n*-vector of shocks in the upper regime will be the SUP factor times BOOTRES(*T*), and it will be the SLO factor times BOOTRES(*T*) in the lower regime. For computing a simulation of dependent variable *I*, all we need is element *I* of that matrix, which we can get by taking row *I* of the factor matrix times BOOTRES(*T*).

The complete model for simulation is obtained by combining the switching equations that we just defined with the definitional identity for the threshold variable. This also includes another identity which will produce a simulated regime-dummy, which will be used for computing simulated probabilities of the regimes:

```
frml(identity) upperfrml dup = thrfrml{d}>thresh
*
group tvar tvarf thrfrml upperfrml
```

We next generate a list of the entries for the regime for which we want to compute the IRF's. This is the simplest way to do this—adapting the **PANEL** instruction which is designed to pick out subsamples based upon an individual identifier. In this case, the “identifier” is the dummy variable for the regime. The **IDENTRIES** option will have a **VECT[VECT[INTEGER]]** which here will have two “lists” of entry numbers, one for each regime, with the corresponding values created in a **VECTOR** by the **ID** option. **RDATES** is a copy of the list of entries of interest, obtained by checking to see which position the desired values is in. (The values aren't necessarily sorted—they depend upon the order in which the values are observed in the data range).

```
panel(group=dup{0},id=values,identries=entries) dup rstart rend
{
  if upper==1.and.values(1)==1.or.upper==0.and.values(1)==0
    compute rdates=entries(1)
  else
    compute rdates=entries(2)
}
```

This sets up the target series for the NL-IRF's. This is a **VECT[RECT[SERIES]]**, where the outer dimension is over the shock size and sign, and the inner is the combination of variable (in the row) and shock (in the column). Note that the paper only graphs the responses of output growth (the first series) to the other shocks, but this will compute responses to all variables.

There's also a set of series for computing the simulated probability of the upper regime. This is organized differently from the IRF's, as there is one probability series per shock and per shock size/sign, so that's a **VECT[VECT[SERIES]]** called **NLPROBS** and a single separate probability for “no shock” (called **NLPROBS0**).

```
compute nexp=%size(shocksizes)
dec vect[rect[series]] nlirfs(nexp)
dec vect[vec[series]] nlprobs(nexp)
dec series nlprobs0
do k=1,nexp
  dim nlirfs(k) (nvar,nvar)
  dim nlprobs(k) (nvar)
  clear(zeros) nlirfs(k) nlprobs(k)
end do k
clear(zeros) nlprobs0
```

The final setup is

```
compute wstart=rstart,wend=rstart+horizon-1
```

which sets up the working range for computing the IRF's. The initial condition information is moved to just before `WSTART`, and the simulations run over the `WSTART` to `WEND` range.

There's an outer simulation loop over the initial conditions and an inner one which is bootstrapping the model given the initial conditions. However, we can just do a flat average across all of those. `RDATES` is a `VECT[INTEGER]` with the dates for the initial conditions in the regime being examined. The `JREP` loop runs over the entries in that, and the `DEPVAR` data over the simulation entries is set from the copies of the original data.

```
compute ninitial=%size(rdates)
infobox(action=define,lower=1,upper=ninitial,progress) $
  "Bootstrapping Across Initial Values"
do jrep=1,ninitial
  infobox(current=jrep)
  *
  * Copy observed data into depvar slots
  *
  compute basedate=rdates(jrep)
  do i=1,nvar
    move data(i)    basedate-maxlag basedate-1 $
      depvars(i) wstart-maxlag
  end do i
```

We then do the inner loop over the bootstraps. We first use `BOOT` to draw the random entries (with replacement) from the available range into `RENTRIES` over the simulation range (`WSTART` to `WEND`). We then draw the standardized (and symmetrized by randomizing the sign) residuals into `BOOTUV` and copy that into `BOOTRES`. (We use a copy because we have to adjust the impacts to create the shocks). `BOOTRES` is what is used in the `FRML`'s to provide the standardized residuals over the simulation period—recall that it gets mapped back to the regime-specific covariance pattern by being multiplied by the regime-specific factor.

```
boot reentries wstart wend rstart rend
gset bootuv wstart wend = $
  %if(%ranflip(.5),+1,-1)*stdv(reentries(t))
*
gset bootres wstart wend = bootuv
*
forecast(model=tvar,from=wstart,to=wend,results=base)
set nlprobs0 wstart wend = nlprobs0+base(nvar+2)
```

Given the bootstrapped residuals, `FORECAST` does the simulations and saves the results in the `VECT[SERIES]` called `BASE`. The no-shock probabilities are updated using the forecasts in slot `NVAR+2` of the results (`NVAR+1` would be the

simulated threshold variable and `NVAR+2` is the dummy for the regime, as we set up the `TVAR` model).

We now do a double loop over the shock size/sign combination and the variable shocked:

```
do k=1,nexp
  do jshock=1,nvar
    *
    * Patch over the component for which we are computing
    * the response with the selected size and sign.
    *
    compute bootres(wstart)=bootuv(wstart)
    compute bootres(wstart)(jshock)=shocksizes(k)
    *
    forecast(model=tvvar,from=wstart,to=wend,results=withshock)
    do i=1,nvar
      set nlirfs(k)(i,jshock) wstart wend = $
        nlirfs(k)(i,jshock)+withshock(i)-base(i)
    end do i
    set nlprobs(k)(jshock) wstart wend = $
      nlprobs(k)(jshock)+withshock(nvar+2)
    end do jshock
  end do k
```

This first puts back the original bootstrap residuals for the `WSTART` entry only—that way everything stays the same except the desired shock. Then the component being shocked is replaced by the `SHOCKSIZES` element being examined (+1, -1, +2 or -2). Because we “reflate” the standardized residuals by premultiplying by the Cholesky factor, this means that these will be Cholesky factor shocks of that many standard deviations and signs. The **FORECAST** will now give the corresponding simulations with the desired shock—that goes into the `VECT[SERIES]` called `WITHSHOCK`. The various `NLIRF` elements are updated using the difference between `WITHSHOCK` and `BASE`, and the probabilities are updated using the simulated value for the regime dummy (again, in the `NVAR+2` slot).

That’s the end of the simulations. The statistics, which are all sums, are divided through by the total number of simulations across bootstraps and initial conditions to get averages and probabilities:

```

do k=1,nexp
  do j=1,nvar
    do i=1,nvar
      set nlirfs(k)(i,j) wstart wend = $
        nlirfs(k)(i,j)/(nboot*ninitial)
    end do i
    set nlprobs(k)(j) wstart wend = $
      nlprobs(k)(j)/(nboot*ninitial)
    end do j
  end do k
  set nlprobs0 wstart wend = nlprobs0/(nboot*ninitial)

```

and, just in case we want to do something else, we move the original data back.

As mentioned above, the paper only graphs responses of output growth to the four shocks (Figure 5.4). However, each of those will have four lines—one for each of the shock size/signs. Note that the responses don't get standardized to a common size, so the two standard deviation responses are *roughly* double the one standard deviation responses and the negative shock responses are roughly mirror images of the positive ones.

```

dec vect[series] graphs(nexp)
*
spgraph(vfields=nvar,hfields=1,footer=$
  "Response of Output Growth to Shocks, Conditional on "+$
  RegimeDesc,key=below,klabels=shocksize1,style=line,$
  ylabels=shortlabels)
do j=1,nvar
  do k=1,nexp
    move nlirfs(k)(1,j) wstart wend graphs(k) 1
  end do i
  graph(series=graphs,number=0,picture="##.##")
end do j
spgraph(done)

```

The probabilities are presented somewhat differently (Figure 5.5)—probably because the graphs would get too crowded, these show only the results for the “no shock” and the ± 2 standard deviation shocks (which are in the second and fourth slots as this is set up).

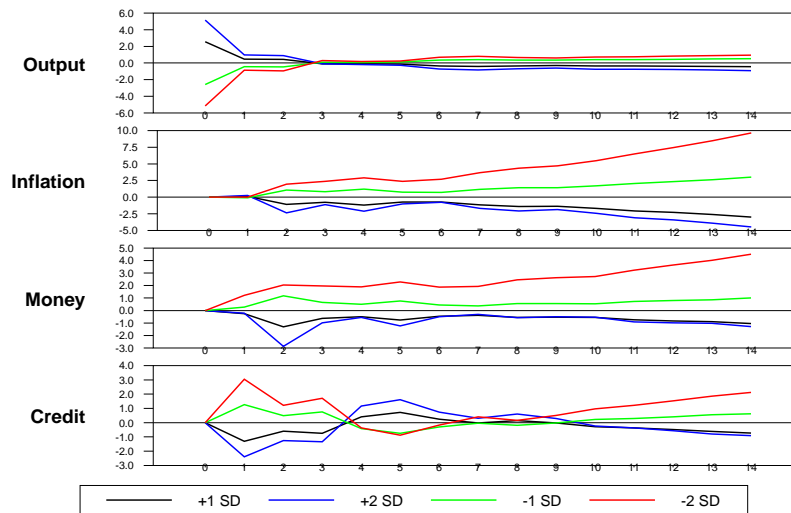


Figure 5.4: Non-linear IRF's in Tight Regime

```

spgraph(vfields=2,hfields=2,footer=$
  "Probability of Tight Regime, Conditional on Starting in "+$
  RegimeDesc,key=below,style=line,$
  klabels=|| "no shock",shocksize(2),shocksize(4) ||)
do j=1,nvar
  graph(number=0,picture="#.##",max=1.0,min=0.0,$
    header=shortlabels(j)) 3
  # nlprobs0      wstart wend
  # nlprobs(2)(j) wstart wend
  # nlprobs(4)(j) wstart wend
end do j
spgraph(done)

```

5.3 Threshold Cointegration

The first paper to tackle the question of threshold cointegration with an *unknown* cointegrating vector was Hansen & Seo (2002). The empirical example in this unfortunately suffered from a number of major errors, not the least of which was an incorrect data set.⁶

This is much more complicated than the case of the known cointegration vector—whether it's even worth the trouble is unclear since it's unlikely that it will be possible to do much in the way of interesting inference on the cointegrating vector. With a known cointegrating vector, we can compute it as a series, graph it, analyze it. We know its values, so we can do the calculations for breakpoints across a fixed set of points. If it's *unknown*, then different values of β give rise to different series with different behavior. We've already seen

⁶The data from roughly the first 50 observations were accidentally read twice and appended to the proper data set.

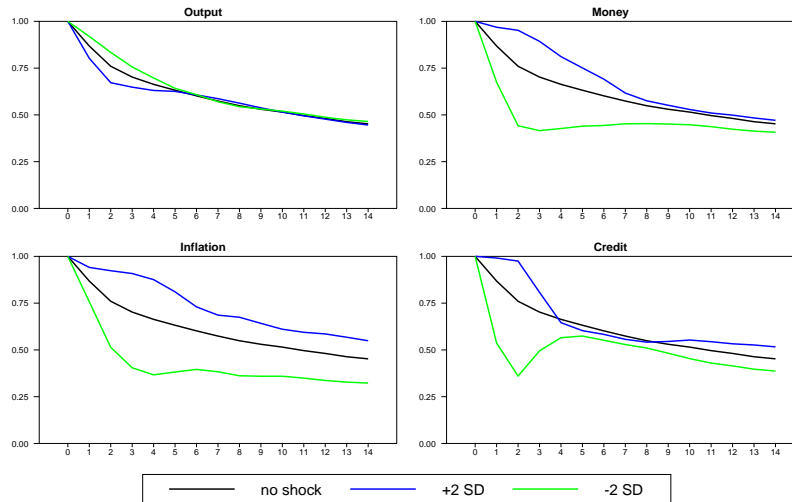


Figure 5.5: Probabilities of Tight Regime Given Tight Regime

that the log likelihood with a *known* cointegrating vector produces a highly variable function. This is compounded quite a bit when you search across both β and the threshold. Given β , the possible thresholds can be computed as before, so an exhaustive search is possible. That isn't possible for β itself, which must be analyzed on a grid. With a grid which is too coarse, it's very easy to miss the global maximum, and no matter how fine the grid, you can never really be sure that you've even found the global maximum since the likelihood function is so ill-behaved.

The procedures for implementing the Hansen-Seo techniques are all in the replication program (not included here). The estimation itself is done with the procedure `@HansenSeo`. This uses the first lag of the cointegrating relation as the threshold variable.

```
@HansenSeo( options ) start end y1 y2
```

Its options are:

- LAGS=# of lags in the VAR
- BETA =input value for cointegrating coefficient ($y_1 - \beta y_2$ stationary) [not used]
- GAMMA=threshold value for partitioning sample [not used]
- BSIZE=size of beta grid search [300]
- GSIZE=size of gamma grid search [300]
- PI=minimum fraction sample in a partition [.05]

The two key parameters are β , the coefficient in the cointegrating relation, and γ , the breakpoint on the threshold variable. You can input either or search for either.

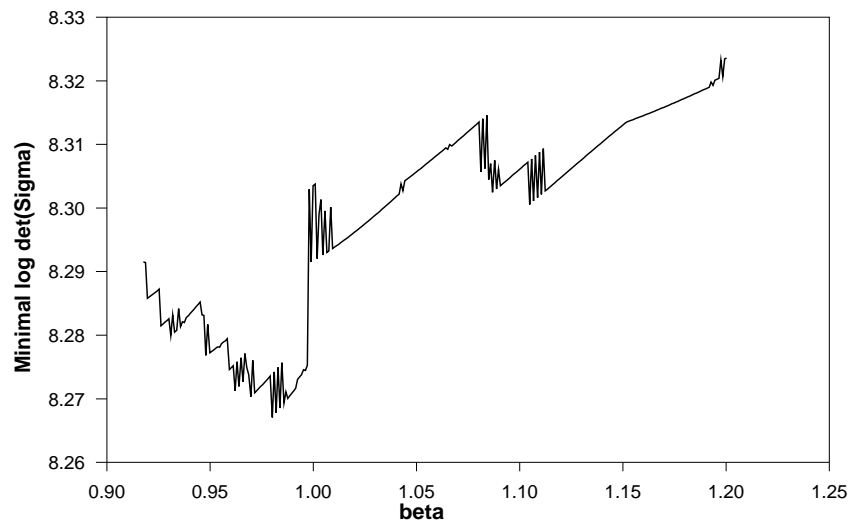


Figure 5.6: $\log |\Sigma|$ vs β in Hansen-Seo Threshold Cointegration Model

The procedure `@HSLMTest` tests for threshold cointegration *given an input* β . This does fixed regressor bootstrapping (Section 3.3) for computing an approximate p -value. For the case of the 1 year vs 10 year yields, the test comes in strongly significant:

Test of R12M and R120M	0.00400
------------------------	---------

and a grid search over β and then over γ given β produces

Estimates of a Threshold Cointegration Model	
Beta	0.980124
Gamma	-0.591760

A graph of the statistic being optimized ($\log |\Sigma|$) against different values of β on the grid is shown in Figure 5.6. Needless to say, this is not the type of graph that you would like to see—tiny changes in β produce fairly dramatic differences in the fit statistic (since they create different possible breakpoints in the search over γ). A different grid for β could end up locating a different cointegrating vector. Overall, this is a sign that this is probably not a good estimation technique, and either a two-step estimator of first estimating the cointegrating vector in a fixed model or assigning it theoretically is likely to be more successful.

Example 5.1 Threshold Error Correction Model

This largely reproduces the results in Balke & Fomby (1997). The data file is a reconstruction, so the results differ slightly. The details are in Section 5.1.

```
cal(m) 1955:1
open data irates.xls
data(format=xls,org=columns) 1955:01 1990:12 fedfunds mdiscret
*
set spread = fedfunds-mdiscret
@dfunit(lags=12) fedfunds
@dfunit(lags=12) mdiscret
@dfunit(lags=12) spread
*
* Pick lags
*
@arautolags(crit=bic) spread
*
* Tsay threshold tests with direct ordering
*
do d=1,4
  @tsaytest(thresh=spread,d=d,$
    title="Tsay Test-Direct Order, delay="+d) spread
  # constant spread{1 2}
end do d
*
* And with reversed ordering
*
set reverse = -spread
do d=1,4
  @tsaytest(thresh=reverse,d=d,$
    title="Tsay Test-Reverse Order, delay="+d) spread
  # constant spread{1 2}
end do d
*
* Arranged D-F t-statistics
*
set dspread = spread-spread{1}
set thresh = spread{1}
rls(order=thresh,cohhistory=coh,sehistory=seh) dspread
# spread{1} constant dspread{1}
*
set tstats = coh(1)(t)/seh(1)(t)
scatter(footer="Figure 1. Recursive D-F T-Statistics\\"+$
  "Arranged Autoregression from Low to High")
# thresh tstats
*
* Same thing in reversed order
*
rls(order=-thresh,cohhistory=coh,sehistory=seh) dspread
# spread{1} constant dspread{1}
*
set tstats = coh(1)(t)/seh(1)(t)
```

```

scatter(footer="Figure 2. Recursive D-F T-Statistics\\"+$
  "Arranged Autoregression from High to Low")
# thresh tstats
*
* The authors do a very coarse grid. This does a full grid over the
* possible threshold values requiring at least 15% of observed
* values to value in each group. (This isn't necessarily 15% of the
* data points because of possible duplication).
*
set thresh = spread{1}
linreg(noprint) spread
# constant spread{1 2}
compute loglbest=%logl
@UniqueValues(values=tvalues) thresh %regstart() %regend()
*
compute n=%rows(tvalues)
compute pi=.15
*
* These are for doing a graph later, and aren't necessary for the
* analysis.
*
compute x=tvalues
compute y=tvalues
compute f=%fill(n,n,%na)
*
compute spacing=fix(pi*n)
*
* These are the bottom and top of the permitted index values for the
* lower index, and the top of the permitted values for the upper
* index.
*
compute lstart=spacing,lend=n+1-2*spacing
compute uend =n+1-spacing
do lindex=lstart,lend
  do uindex=lindex+spacing,uend
    sweep(group=(thresh>=tvalues(lindex))+(thresh>tvalues(uindex)))
    # spread
    # constant spread{1 2}
    if %logl>loglbest
      compute lindexbest=lindex,uindexbest=uindex,loglbest=%logl
    compute f(lindex,uindex)=%logl
  end do uindex
end do lindex
*
disp "Best Break Values" tvalues(lindexbest) "and" tvalues(uindexbest)
*
* This graphs the log likelihood function across values of the left
* threshold, given the maximizing value for the right threshold. It
* also includes a line at the .05 critical value for a likelihood
* ratio test for a specific value of the left threshold.
*
set testf 1 n = f(t,uindexbest)
set testx 1 n = tvalues(t)
compute yvalue=loglbest-.5*%invchisqr(.05,1)

```

```

spgraph
scatter(vgrid=yvalue, footer=$
  "Log likelihood as function of left threshold given right threshold")
# testx testf
grtext(y=yvalue, x=0.0, direction=45) $
  ".05 Critical Point for Left Threshold"
spgraph(done)
*
* This will be 0, 1 or 2, depending upon the value of thresh
*
set group = (thresh>=tvalues(lindexbest))+(thresh>tvalues(uindexbest))
*
do for i = 0 1 2
  disp "**** Group " i "****"
  linreg(smpl=(group==i)) spread
  # constant spread{1 2}
  summarize(noprint) %beta(2)+%beta(3)-1.0
  disp "DF T-Stat" %cdstat
end do i
*
* Threshold error correction models
*
set dff = fedfunds-fedfunds{1}
set ddr = mdiscret -mdiscret{1}
do for i = 0 1 2
  disp "**** Group " i "****"
  linreg(smpl=(group==i)) dff
  # constant dff{1} ddr{1} spread{1}
  linreg(smpl=(group==i)) ddr
  # constant dff{1} ddr{1} spread{1}
end do i

```

Example 5.2 Threshold Error Correction Model: Forecasting

This is based upon Balke & Fomby (1997). However, unlike Example 5.1, this estimates the threshold using the bivariate likelihood, and computes eventual forecast functions and NL-IRFs.

```

cal(m) 1955:1
open data irates.xls
data(format=xls, org=columns) 1955:01 1990:12 fedfunds mdiscret
*
set spread = fedfunds-mdiscret
set thresh = spread{1}
linreg spread
# constant spread{1 2}
@UniqueValues(values=tvalues) thresh %regstart() %regend()
*
compute n=%rows(tvalues)
compute pi=.15
*
compute spacing=fix(pi*n)

```

```

*
* These are the bottom and top of the permitted index values for the
* lower index, and the top of the permitted values for the upper
* index.
*
compute lstart=spacing, lend=n+1-2*spacing
compute uend =n+1-spacing
*
set dff = fedfunds-fedfunds{1}
set ddr = mdiscrt -mdiscrt{1}
*
sweep
# dff ddr
# constant dff{1} ddr{1} spread{1}
compute loglbest=%logl
*
do lindex=lstart, lend
  do uindex=lindex+spacing, uend
    sweep(group=(thresh>=tvalues(lindex))+(thresh>tvalues(uindex)))
    # dff ddr
    # constant dff{1} ddr{1} spread{1}
    if %logl>loglbest
      compute lindexbest=lindex, uindexbest=uindex, loglbest=%logl
    end do uindex
  end do lindex
disp "Best Break Values" tvalues(lindexbest) "and" tvalues(uindexbest)
*
compute lower=tvalues(lindexbest), upper=tvalues(uindexbest)
dec frml[int] switch
frml switch = 1+fix((spread{1}>=lower)+(spread{1}>upper))
*
* Estimate the model at the best breaks to get the covariance matrix.
*
sweep(group=switch(t))
# dff ddr
# constant dff{1} ddr{1} spread{1}
compute tvecmsigma=%sigma
*
set dff = fedfunds-fedfunds{1}
set ddr = mdiscrt -mdiscrt{1}
*
system(model=basevecm)
variables dff ddr
lags 1
det constant spread{1}
end(system)
*
dec rect[frml] tvecfrml(2,3)
do i=1,3
  estimate(smpl=(switch(t)==i))
  frml(equation=%modeleqn(basevecm,1)) tvecfrml(1,i)
  frml(equation=%modeleqn(basevecm,2)) tvecfrml(2,i)
end do i
*

```

```

frml(identity) ffid fedfunds = fedfunds{1}+dff
frml(identity) drid mdiscret = mdiscret{1}+ddr
frml(identity) spid spread = fedfunds-mdiscret
*
frml dffeq dff = tvecfrml(1,switch(t))
frml ddreq ddr = tvecfrml(2,switch(t))
*
group tvecm dffeq ddreq ffid drid spid
*
* Eventual forecast function, starting with 1981:1 data (largest
* value of spread).
*
forecast(model=tvecm,from=1981:1,steps=40,results=eff)
graph(footer=$
    "Eventual Forecast Function for SPREAD, starting at 1981:1")
# eff(5)
graph(footer=$
    "Eventual Forecast Function for Change in DR, starting at 1981:1")
# eff(2)
*
* NL-IRF starting in 1969:3 for a one s.d. shock to DR correlated
* with FF using the estimated covariance matrix. (1969:3 has values
* for both rates which are close to the average for the full
* period).
*
compute ndraws=5000
compute baseentry=1969:3
compute nsteps =40
*
dec vect[series] fshocks(2) nlirf(5)
dec series[vect] bishocks
dec vect ishocks
*
smpl baseentry baseentry+(nsteps-1)
do i=1,5
    set nlirf(i) = 0.0
end do i
*
* Do a factor of the covariance matrix in the order 2,1 (which means
* FSIGMA will be upper triangular).
*
compute fsigma=%psdfactor(tvecmsigma,||2,1||)
*
do draw=1,ndraws
    *
    * Draws shocks from the estimated covariance matrix.
    *
    gset bishocks = %ranmvnormal(fsigma)
    *
    * Do forecast with drawn shocks during the whole period.
    *
    set fshocks(1) = bishocks(t)(1)
    set fshocks(2) = bishocks(t)(2)
    forecast(paths,model=tvecm,results=basesims)

```

```

# fshocks
*
* Replace the impacts with a one-standard deviation shock to the
* discount rate and a corresponding draw to the conditional
* distribution for the FFR.
*
compute %pt(fshocks,baseentry,fsigma*||%ran(1.0)|1.0||)
forecast(paths,model=tvecm,results=sims)
# fshocks
do i=1,%size(nlirf)
    set nlirf(i) = nlirf(i)+(sims(i)-basesims(i))
end do i
end do draw
*
do i=1,%size(nlirf)
    set nlirf(i) = nlirf(i)/ndraws
end do i
*
graph(number=0,footer=$
    "NL-IRF for Discount Rate to One S.D. Shock in Discount Rate")
# nlirf(4)
graph(number=0,footer=$
    "NL-IRF for FedFunds Rate to One S.D. Shock in Discount Rate")
# nlirf(3)
graph(number=0,footer=$
    "NL-IRF for Spread to One S.D. Shock in Discount Rate")
# nlirf(5)

```

Example 5.3 Threshold VAR, Multiple Breaks

This is based upon Tsay (1998). Data are similar, but not identical to those used in the paper. This is discussed in Section 5.2.1.

```

open data usrates.xls
calendar(m) 1959
data(format=xls,org=columns) 1959:1 1993:2 fcm3 ftb3
set g3year = log(fcm3/fcm3{1})
set g3month = log(ftb3/ftb3{1})
set spread = log(ftb3)-log(fcm3)
set sspread = (spread+spread{1}+spread{2})/3
compute sspread(1959:1)=spread(1959:1)
compute sspread(1959:2)=(spread(1959:1)+spread(1959:2))/2
*
spgraph(vfields=3,footer=$
    "Figure 3. Time Plots of Growth Series of U.S. Monthly Interest Rates")
graph(vlabel="3-month")
# g3month
graph(vlabel="3-year")
# g3year
graph(vlabel="Spread")
# sspread
spgraph(done)

```

```

*
@VARLagSelect(lags=12,crit=aic)
# g3year g3month
*
compute p =7
compute k =2
*
do d=1,7
  dofor m0 = 50 100
    set thresh = ssread{d}
    *
    rls(noprint,order=thresh,condition=m0) g3year / rr3year
    # constant g3year{1 to p} g3month{1 to p}
    rls(noprint,order=thresh,condition=m0) g3month / rr3month
    # constant g3year{1 to p} g3month{1 to p}
    *
    * We need to exclude the conditioning observations, so we
    * generate the series of ranks of the threshold variable over
    * the estimation range.
    *
    order(ranks=rr) thresh %regstart() %regend()
    *
    linreg(noprint,smpl=rr>m0) rr3year / wr3year
    # constant g3year{1 to p} g3month{1 to p}
    linreg(noprint,smpl=rr>m0) rr3month / wr3month
    # constant g3year{1 to p} g3month{1 to p}
    *
    ratio(mcorr=%nreg,degrees=k*%nreg,noprint)
    # rr3year rr3month
    # wr3year wr3month
    disp "D=" d "m0=" m0 @16 "C(d)=" *.## %cdstat $
      @28 "P-value" #.##### %signif
  end dofor m0
end do d
*
* Evaluate the AIC across a grid of threshold settings
*
set thresh = ssread{1}
@gridseries(from=-.30,to=.05,n=300,pts=ngrid) rgrid
set aic 1 ngrid = 0.0
*
compute bestaic=%na
*
do i=1,ngrid
  compute rtest=rgrid(i)
  sweep(group=thresh<rtest,var=homo)
  # g3year g3month
  # constant g3year{1 to p} g3month{1 to p}
  compute aic(i)=-2.0*%logl+2.0*%nregsystem
  if i==1.or.aic(i)<bestaic
    compute bestaic=aic(i),bestbreak=rtest
end do i
*
scatter(footer=$

```



```

"AIC Values for Interest Rates Regression vs Break Point")
# rgrid aic
disp "Best Break is" bestbreak "with AIC" bestaic
*
set thresh = ssread{1}
*
* This is a slightly different set of grids than used in the paper.
*
@gridseries(from=-.30,to=.05,n=80,pts=ngrid) rgrid
*
compute bestaic=%na
do i=1,ngrid-19
  do j=i+20,ngrid
    sweep(group=(thresh<rgrid(i))+(thresh<rgrid(j)),var=homo)
    # g3year g3month
    # constant g3year{1 to p} g3month{1 to p}
    compute thisaic=-2.0*%logl+2.0*%nregsystem
    if .not.%valid(bestaic).or.thisaic<bestaic
      compute bestaic=thisaic,bestlower=rgrid(i),bestupper=rgrid(j)
  end do j
end do i
*
disp "Best double break is at" bestlower "and" $
  bestupper "with AIC" bestaic

```

Example 5.4 Threshold VAR, Testing and Estimation

This is based upon Balke (2000). It does the testing and estimation for his four-variable TVAR with thresholds based upon a measure of credit market conditions. Details are in Section 5.2.2.

```

cal(q) 1947
allocate 1997:4
*
open data flobund2.dat
data(unit=data,org=obs) 1947:1 1997:4 $
  date nfbfla nfbstk nfbfln nfcfla nfcstk nfcfln $
  cmpapfln cmpapfla cmpapstk rgdpcw pgdpcw fyff m2 cp46 t6
*
* Control parameters
*
* Fraction of threshold values excluded at each end
*
compute pi=.15
*
* Number of VAR lags
*
compute maxlag=4
*
* These are the desired start and end of the sample to be used in
* the estimation. However, the transformations use the entire range.
*

```

```

compute sstart = 1959:1
compute send   = 1997:3
*
set mix        = (nfbstk+nfcstk)/(cmpapstk+nfbstk+nfcstk)
set gdp        = rgdpcw
set pgdp       = pgdpcw
set dlmix      = (mix-mix{1})
*
set cpbill11   = cp46-t6
*
set dlgrp      = 400*log(gdp/gdp{1})
set dlpgrp     = 400*log(pgdp/pgdp{1})
set ly         = log(gdp)
*
set dly        = dlgrp
set dlp        = dlpgrp
set money      = fyff
*
* This is the credit measure, and its corresponding delay and MA
* length.
*
set credit     = cpbill11
compute d = 1
compute malength = 2
*
*set credit = dsd
*set credit = dlmix
*
system(model=varmodel)
variables dly dlp money credit
lags 1 to maxlag
det constant
end(system)
*
compute rstart=sstart+maxlag,rend=send
estimate(print,resids=u) rstart rend
*
compute loglr =%logl
compute nfreer=%nfree
compute nvar  =%nvar
*
* Create the desired moving average of the credit measure for use
* as the threshold variable.
*
filter(type=lagging,span=malength) credit / credthr
*
* Get a sorted copy of the (delayed) threshold variable.
*
set copy rstart rend = credthr{d}
order copy rstart rend
*
* PI is the fraction of observations required over and above the
* number of regressors per equation (that is, the individual
* degrees of freedom).

```

```

*
compute piskip=fix(pi*%nobs)+%nreg
compute pistart=rstart+piskip,piend=rend-piskip
*
compute besttest=loglr
*
* Loop over the possible values for breaks.
*
set teststats pistart piend = 0.0
do pientry=pistart,piend
    compute thresh=copy(pientry)
    *
    * This allows the covariance matrix to be regime-dependent.
    *
    sweep(var=hetero,group=credthr{d}<thresh) rstart rend
    # %modeldepvars(varmodel)
    # %rlffromeqn(%modeleqn(varmodel,1))
    *
    * If this is best so far, save it
    *
    if %logl>besttest
        compute besttest=%logl,bestthresh=thresh
        compute teststats(pientry)=2.0*(%logl-loglr)
end do pientry
*
disp "Best Threshold Value" bestthresh
scatter(style=lines,hgrid=bestthresh,$
    footer="Likelihood ratio vs break value")
# copy teststats
*
compute suplr=2.0*(besttest-loglr)
disp "LR Test" suplr "Degrees of freedom" %nfree-nfreer
*
* Fixed regressor bootstrap. This does *not* allow for changes in
* the covariance matrix.
*
dec vect[series] fiddled(%nvar)
*
compute nboot=500
set avgboot 1 nboot = 0.0
set expboot 1 nboot = 0.0
set supboot 1 nboot = 0.0
*
* Loop over NBOOT+1 replications, where the final one uses the
* actual residuals.
*
do reps=1,nboot+1
    set teststats pistart piend = 0.0
    *
    * All residuals are scaled by a common N(0,1) at time t.
    *
    set fiddlef = %ran(1.0)
    do i=1,%nvar
        set fiddled(i) = %if(reps<=nboot,fiddlef*u(i),u(i))

```

```

end do i
*
sweep rstart rend
# fiddled
# %rlffromeqn(%modeleqn(varmodel,1))
compute loglr=%logl
compute besttest=%logl
*
do pientry=pistart,piend
  compute thresh=copy(pientry)
  sweep(var=homo,group=credthr{d}<thresh) rstart rend
  # fiddled
  # %rlffromeqn(%modeleqn(varmodel,1))
  *
  * If this is best so far, save it
  *
  if %logl>besttest
    compute besttest=%logl,bestthresh=thresh
    compute teststats(pientry)=2.0*(%logl-loglr)
  end do pientry
*
* Compute the functionals of the test statistics. This includes
* an overflow-safe calculation of the Andrews-Ploberger
* exponential average.
*
sstats(max) pistart piend teststats>>aqvalue
sstats(mean) pistart piend teststats>>avgvalue $
               exp(.5*(teststats-aqvalue))>>apvalue
compute apvalue=log(apvalue)+.5*aqvalue
if reps<=nboot {
  compute expboot(reps)=apvalue
  compute avgboot(reps)=avgvalue
  compute supboot(reps)=aqvalue
}
end do reps
*
* Compute bootstrapped p-values
*
sstats(mean) 1 nboot (expboot>apvalue)>>exppvalue $
                  (supboot>aqvalue)>>suppvalue $
                  (avgboot>avgvalue)>>avgpvalue
*
report(action=define)
report(atrow=1,atcol=1) "" "Statistic" "P-value"
report(atrow=2,atcol=1) "Sup" aqvalue suppvalue
report(atrow=3,atcol=1) "Avg" avgvalue avgpvalue
report(atrow=4,atcol=1) "Exp" apvalue exppvalue
report(atcol=2,tocol=2,action=format,width=8)
report(atcol=3,tocol=3,action=format,picture="*.*###")
report(action=show)

```

Example 5.5 Threshold VAR, Impulse Responses

This is a continuation of Example 5.4, taking the results from that (the chosen threshold value) and generating non-linear IRF's. Technical information can be found in Section 5.2.3.

```

comp upper      =1
comp regimeDesc="Tight Regime"
*
* Control parameters
*
comp horizon =15
*
* Number of bootstrap replications used in computing NLIRF's
*
comp nboot      =500
*
compute [vector] shocksizes=||+1.0,+2.0,-1.0,-2.0||
compute [vec[string]] shocksize=||"+1 SD","+2 SD","-1 SD","-2 SD"||
*
cal(q) 1947
allocate 1997:4
*
open data flobund2.dat
data(unit=data,org=obs) 1947:1 1997:4 $
    date nfbfla nfbstk nfbfln nfcfla nfcstk nfcfln $
    cmpapfln cmpapfla cmpapstk rgdpcw pgdpcw fyff m2 cp46 t6
*
* These are the desired start and end of the sample to be used in
* the estimation. However, the transformations use the entire range.
*
compute sstart = 1959:1
compute send   = 1997:3
*
set mix        = (nfbstk+nfcstk)/(cmpapstk+nfbstk+nfcstk)
set gdp        = rgdpcw
set pgdp       = pgdpcw
set dlmix      = (mix-mix{1})
*
set cpbill11   = cp46-t6
*
set dlgdp      = 400*log(gdp/gdp{1})
set dlpdp      = 400*log(pgdp/pgdp{1})
set ly         = log(gdp)
*
set dly        = dlgdp
set dlp        = dlpdp
set money      = fyff
*
compute nvar   =4
compute maxlag=4
*
dec vect[int]      depvars(nvar)

```

```

dec vect[series]    resup(nvar) reslo(nvar)
dec vect[frml]      fitup(nvar) fitlo(nvar)
dec vect[frml]      tvarf(nvar)
dec vect[series]    bootu(nvar) data(nvar)
dec series[vect]    bootuv bootres
*
dec vect[string]    shortlabels(nvar) longlabels(nvar)
dec vector macoeffs
*
* This is the credit measure, MA length, delay and threshold value
* from the estimation program.
*
set credit          = cpbill1
comp malength       = 2
comp d              = 1
comp thresh         = 0.48667
*
compute depvars      = || dly, dlp, money, credit ||
compute shortlabels = || "Output", "Inflation", "Money", "Credit" ||
compute longlabels   = || "Output Growth", "Inflation", "Fed Funds", "CP-Bill Spread" ||
compute lab          = || "Y", "P", "Fed Funds", "CP-Bill" ||
*
* This generates the actual moving average of the data
*
filter(type=lagging, span=malength) credit / credthr
*
* This generates an equation (and from it a FRML) to compute the
* moving average of the data
*
compute macoeffs=%fill(malength,1,1.0/malength)
equation(identity,coeffs=macoeffs) threqn credthr
# credit{0 to malength-1}
frml(equation=threqn,identity) thrfrml
*****
*
* Dummy variables for the two regimes
*
set dup = thrfrml{d}>thresh
set dlo = 1.-dup
*
system(model=basevar)
variables depvars
lags 1 to maxlag
det constant
end(system)
*
* Estimate under the two regimes, and define regime-specific FRML's
* for each, and save the regime-specific covariance matrix.
*
compute rstart=sstart+maxlag,rend=send
estimate(smpl=dup,resids=resup,title="Upper Regime") rstart rend
do i=1,nvar
    frml(equation=%modeleqn(basevar,i)) fitup(i)
end do i

```

```

compute vup=%sigma
*
estimate(smpl=dlo, resids=reslo, title="Lower Regime") rstart rend
do i=1, nvar
    frml(equation=%modeleqn(basevar, i)) fitlo(i)
end do i
compute vlo=%sigma
*
* Compute the Cholesky factors and inverse factors for the upper and
* lower branch covariance matrices. (The inverse is for
* standardizing the residuals and the factor for mapping
* standardized residuals back to the true levels.)
*
compute sup =%decomp(vup)
compute siup=inv(sup)
compute slo =%decomp(vlo)
compute silo=inv(slo)
*
* Compute (jointly) standardized residuals. Since we only use these
* as a VECTOR across variables at T (never as individual series),
* we define it as a SERIES[VECTOR].
*
dec series[vect] stdu
gset stdu rstart rend = %if(dup, siup*%xt(resup, t), silo*%xt(reslo, t))
*
* Save the original data since we'll overwrite it as part of the
* bootstrap.
*
dec vect[series] data(nvar)
do i=1, nvar
    set data(i) = depvars(i){0}
end do i
*
* Define the switching formulas. Because the bootstrap requires
* taking the standardized residuals and reflating them using
* regime-specific covariance matrices, the reflation part has to be
* incorporated into the formula. Thus TVARF(i) is (in effect) an
* identity given the (still to be created) bootres series.
*
do i=1, nvar
    frml tvarf(i) depvars(i) = %if(thrfrml{d}>thresh, $
        fitup(&i)+%dot(%xrow(sup, &i), bootres), $
        fitlo(&i)+%dot(%xrow(slo, &i), bootres))
end do i
*
* Create a FRML to determine which regime holds in a simulation. (It
* will be 1 for upper and 0 for lower)
*
frml(identity) upperfrml dup = thrfrml{d}>thresh
*
* Build the threshold var model using the switching equations and
* the definitional identity for the threshold variable.
*
group tvar tvarf thrfrml upperfrml

```

```

*
*****
*
* Figure out which set of entries we want. This is the simplest way
* to get this. dup{0} takes two values (0 or 1) but the order isn't
* known in advance since it will depend upon which value is seen
* first in the <<rstart>> to <<rend>> range. So we have to figure
* out which of values(1) and values(2) (and the corresponding
* entries(1) and entries(2)) is the one that we need, given the
* choice for <<upper>>.
*
panel(group=dup{0},id=values,identries=entries) dup rstart rend
{
if upper==1.and.values(1)==1.or.upper==0.and.values(1)==0
  compute rdates=entries(1)
else
  compute rdates=entries(2)
}
*
* Set up target series. There is one for each combination of test
* sign and sizes on the shock, variable shocked and target variable.
*
compute nexp=%size(shocksizes)
dec vect[rect[series]] nlirfs(nexp)
dec vect[vect[series]] nlprobs(nexp)
dec series nlprobs0
do k=1,nexp
  dim nlirfs(k) (nvar,nvar)
  dim nlprobs(k) (nvar)
  clear(zeros) nlirfs(k) nlprobs(k)
end do k
clear(zeros) nlprobs0
*
* This is the working range for calculating the NLIRF's.
*
compute wstart=rstart,wend=rstart+horizon-1
*
* Outer loop is over the initial conditions, which walk through the
* data points in the desired regime. The residuals are bootstrapped
* by taking the standardized residuals (across the entire range),
* shuffling them, and reflating them based upon the current
* threshold value in the generated series.
*
compute ninitial=%size(rdates)
infobox(action=define,lower=1,upper=ninitial,progress) $
  "Bootstrapping Across Initial Values"
do jrep=1,ninitial
  infobox(current=jrep)
  *
  * Copy observed data into depvar slots
  *
  compute basedate=rdates(jrep)
  do i=1,nvar
    move data(i)    basedate-maxlag basedate-1 $

```



```

        depvars(i) wstart-maxlag
    end do i
    *
    * Loop over bootstrap replications
    *
    do boot=1,nboot
        *
        * Generate the bootstrap shuffle
        *
        boot reentries wstart wend rstart rend
        *
        * Generate the base set of shocks by premultiplying the
        * bootstrapped standardized shocks by the factor of the
        * overall covariance matrix.
        *
        gset bootuv wstart wend = $
            %if(%ranflip(.5),+1,-1)*stdv(reentries(t))
        *
        gset bootres wstart wend = bootuv
        *
        forecast(model=tvar,from=wstart,to=wend,results=base)
        set nlprobs0 wstart wend = nlprobs0+base(nvar+2)
        do k=1,nexp
            do jshock=1,nvar
                *
                * Patch over the component for which we are computing
                * the response with the selected size and sign.
                *
                compute bootres(wstart)=bootuv(wstart)
                compute bootres(wstart)(jshock)=shocksizes(k)
                *
                forecast(model=tvar,from=wstart,to=wend,results=withshock)
                do i=1,nvar
                    set nlirfs(k)(i,jshock) wstart wend = $
                        nlirfs(k)(i,jshock)+withshock(i)-base(i)
                end do i
                set nlprobs(k)(jshock) wstart wend = $
                    nlprobs(k)(jshock)+withshock(nvar+2)
            end do jshock
        end do k
    end do boot
end do jrep
infobox(action=remove)
*
* Divide the sums through to get average responses, and regime
* probability.
*
do k=1,nexp
    do j=1,nvar
        do i=1,nvar
            set nlirfs(k)(i,j) wstart wend = $
                nlirfs(k)(i,j)/(nboot*ninitial)
        end do i
        set nlprobs(k)(j) wstart wend = $

```

```

        nlprobs(k)(j)/(nboot*ninitial)
    end do j
end do k
set nlprobs0 wstart wend = nlprobs0/(nboot*ninitial)
*
* Move the data back
*
do i=1,nvar
    set depvars(i) = data(i)
end do i
*
*****
*
* Graph response of output growth (variable 1) to all four shocks.
*
dec vect[series] graphs(nexp)
*
spgraph(vfields=nvar,hfields=1,footer=$
    "Response of Output Growth to Shocks, Conditional on "+$
    RegimeDesc,key=below,klabels=shocksizel,style=line,$
    ylabels=shortlabels)
do j=1,nvar
    do k=1,nexp
        move nlirfs(k)(1,j) wstart wend graphs(k) 1
    end do i
    graph(series=graphs,number=0,picture="##.##")
end do j
spgraph(done)
*
* Graph probabilities of upper regime. (Much of the layout of this
* is specific to this problem).
*
spgraph(vfields=2,hfields=2,footer=$
    "Probability of Tight Regime, Conditional on Starting in "+$
    RegimeDesc,key=below,style=line,$
    klabels=||"no shock",shocksizel(2),shocksizel(4)||)
do j=1,nvar
    graph(number=0,picture="##.##",max=1.0,min=0.0,$
        header=shortlabels(j)) 3
    # nlprobs0      wstart wend
    # nlprobs(2)(j) wstart wend
    # nlprobs(4)(j) wstart wend
end do j
spgraph(done)

```

STAR Models

The threshold models considered in Chapters 4 and 5 have both had sharp cutoffs between the branches. In many cases, this is unrealistic, and the lack of continuity in the objective function causes other problems—you can't use any asymptotic distribution theory for the estimates, and short-term forecasting is imprecise because it's not clear how to handle simulated values that fall between the observed data values near the cutoff.

An alternative is the STAR model (Smooth Transition AutoRegression) and more generally, the STR (Smooth Transition Regression). Instead of the sharp cutoff, this uses a smooth function of a threshold variable. One way to write this is:

$$y_t = X_t\beta_{(1)} + X_t\beta_{(2)}G(Z_{t-d}, \gamma, c) + u_t \quad (6.1)$$

The transition function G is bounded between 0 and 1, and depends upon a location parameter c and a scale parameter γ .¹ The two standard transition functions are the logistic (LSTAR) and the exponential (ESTAR). The formulas for these are:

$$G(Z_{t-d}, \gamma, c) = \begin{cases} 1 - [1 + \exp(\gamma(Z_{t-d} - c))]^{-1} & \text{for LSTAR} \\ 1 - \exp(-\gamma(Z_{t-d} - c)^2) & \text{for ESTAR} \end{cases} \quad (6.2)$$

LSTAR is more similar to the models examined earlier, with values to the left of c generally being in one branch (with coefficient vector $\beta_{(1)}$) and those to the right of c having coefficient vector more like $\beta_{(1)} + \beta_{(2)}$. ESTAR treats the tails symmetrically, with values near c having coefficients near $\beta_{(1)}$, while those farther away (in either direction) being close to $\beta_{(1)} + \beta_{(2)}$. ESTAR is often used when there are seen to be costs of adjustment in either direction. An unrestricted three branch model (such as in Balke-Fomby, Example 5.1) could be done by adding in a second LSTAR branch.

STAR models, at least theoretically, can be estimated using non-linear least squares. This, however, requires a bit of finesse—under the default initial values of zero for all parameters used for **NLLS**, both the parameters in the transition function and the autoregressive coefficients that they control have zero derivatives. As a result, if you do **NLLS** with the default `METHOD=GAUSS`, it can never move the estimates away from zero. A better way to handle this is to

¹There are other, equivalent, ways of writing this. The form we use here lends itself more easily to *testing* for STAR effects, since it's just least squares if $\beta_{(2)}$ is zero.

split the parameter set into the transition parameters and the autoregressive parameters, and first estimate the autoregressions conditional on a pegged set of values for the transition parameters. With c and γ pegged, (6.1) is linear, and so converges in one iteration. The likelihood function generally isn't particularly sensitive to the choice of γ , but it can be sensitive to the choice for c , so you might want to experiment with several guesses for c before deciding whether you're done with the estimation.

Although the likelihood is relatively insensitive to the value of γ , that's only when it's in the proper range. As you can see from (6.2), γ depends upon the scale of the transition variable Z_{t-d} . A guess value of something like 1 or 2 times σ_z^{-1} for an LSTAR and σ_z^{-2} for an ESTAR is generally adequate. In Terasvirta (1994), the LSTAR exponent is directly rescaled by (an approximate) reciprocal standard deviation. If you do that, the γ values will have some similarities from one application to the next.

For the LSTAR model, *do not* use the formula as written in (6.2)—for large positive values of Z_{t-d} , the *exp* function will overflow, causing the entire function to compute as a missing value. *exp* of a large *negative* value will underflow (which you could get in the negative tail for an LSTAR and either tail for an ESTAR), but underflowed values are treated as zero, which gives the proper limit behavior. To avoid the problem with the overflows, use the %LOGISTIC function, which does the same calculation, but computes in a different form which avoids overflow.²

LSTAR models include the sharp cutoff models as a special case where $\gamma \rightarrow \infty$. However, where a sharp cutoff is appropriate, you may see very bad behavior for the non-linear estimates from LSTAR. For instance, in an LSTAR model estimated on a one-digit version of the U.S. unemployment rate (not the three-digit one used in Example 4.1), the estimates for γ and c showed as:

Variable	Coeff	Std Error	T-Stat	Signif
11. GAMMA	220.943152	286157411.494605	7.72104e-007	0.99999939
12. C	0.018804	24356.890066	7.72021e-007	0.99999939

The standard errors look (and are) nonsensical, but this is a result of the likelihood (or sum of squares) function being flat for a range of values. It's always a good idea to graph the transition function against the threshold, particularly when the results look odd. In this case, it gives us Figure 6.1. With the exception of a very tiny non-zero weight at zero, this is the same as the sharp transition. Almost any value of c between 0 and 1 will give almost the identical transition function, and so will almost any large value of γ .

6.1 Testing

A straightforward test for the absence of a STAR effect in (6.1) won't have the standard asymptotic distribution because under the null that $\beta_{(2)} = 0$, the tran-

² $1/(1 + \exp(x)) = \exp(-x)/(\exp(-x) + 1)$, one form of which will have the safe negative exponents.

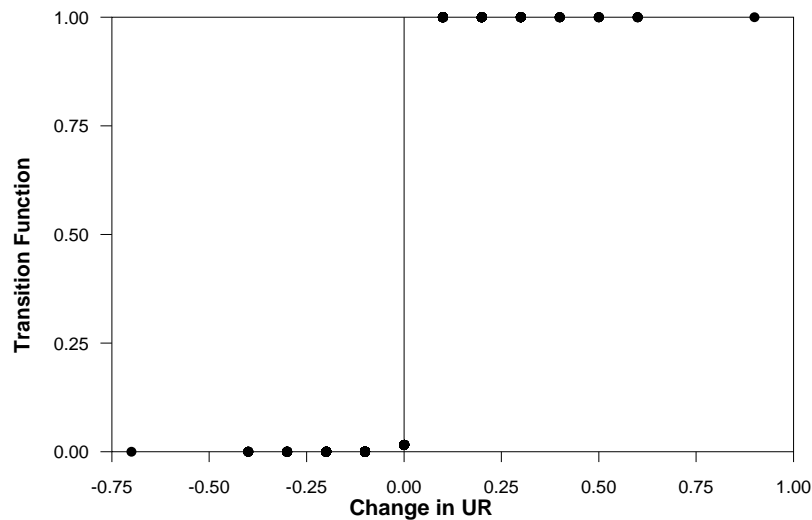


Figure 6.1: Transition Function in LSTAR for One-Digit Unemployment Rate

Table 6.1: Test for STAR in series DUR

Test for STAR in series DUR
 AR length 4
 Delay 1

Test	F-stat	Signif
Linearity	3.7227	0.0000
H01	3.8249	0.0044
H02	4.5925	0.0012
H03	2.5771	0.0366
H12	4.2525	0.0001

sition parameters γ and c aren't identified. Instead, Terasvirta (and colleagues) in a series of papers proposed a battery of LM tests based upon a Taylor expansion of G . Under the null that there is no STAR effect, there should be zero coefficients on a set of interaction terms between the regressors and powers of the transition variable Z_{t-d} . These are computed by the procedure `@STARTest`. The output for this for the unemployment rate series (using the first lag as the threshold variable) is in Table 6.1. This is part of Example 6.1.

The Linearity test includes all the interaction terms through the 3rd power of the transition variable, and serves as a general test for a STAR effect. H01, H02 and H03 are tests on the single power individually, while H12 is a joint test with the first and second powers only. For an LSTAR model, all of these should be significant. For an ESTAR, because of symmetry, the 3rd power shouldn't enter, so you should see H12 significant and H03 insignificant. For the ESTAR, you would also likely reject on the Linearity line, but it won't have as much power as H12, since it includes the 3rd power terms as well.

A common recommendation is to choose the delay d on the threshold based upon the test which gives the strongest rejection. In this example, all delays

from 1 to 4 are examined, and $d = 1$ has by far the largest test statistic.

@STARTEST is strictly for STAR models. More generally, there are STR models (for Smooth Transition Regression) which allow for more general regressions, and for a transition variable that isn't a lag of the dependent variable. For instance, there are models with a smooth transition structural change, that is, the threshold variable is time and instead of a sharp break as in Chapter 3, there's a smooth transition. The testing procedure is similar, and is done by the **@RegSTRTest** procedure. This is a regression post-processor, so you run the base regression first, then apply **@RegSTRTest** after it. For instance,

```
linreg DLR
# constant z{1} DCPI{1} dlr{1}
@RegSTRTest(threshold=z,d=1)
```

tests for a smooth transition break in an error correction model (z is the error correction term) with the presumed break in z_1 (THRESHOLD= Z with D (delay) = 1). The output is similar to that from **@STARTEST** and is interpreted the same way.

A word of caution: the ESTAR model, in particular, can very easily be fooled by outliers (affecting not just the estimators but also the testing procedures). If an outlier has a unique value of the threshold variable (that is, no normal data point shares the same value), then making c equal to that unique value and γ very large will make the second term in (6.2) near 1.0 at all points except the outlier where it would be 0.0, so the outlier will, in effect, have its own regime. This isn't as much of an issue with an LSTAR, since it can't split off a single point like that unless it's associated with the maximum or minimum value for the threshold. This should serve as another reminder not to confuse the null and alternative in this type of test—the null is that the non-breaking model is correct while the alternative *suggests* that a smooth transition might be in order, but not that a smooth transition is *correct*.

6.2 Estimation

For the unemployment data, given the choice of delay (1), the STAR model is set up with

```
nonlin(parmset=starparms) gamma c
frml glstar = %logistic(gamma*(dur{1}-c),1.0)
stats(noprint) dur
compute c=0.0,gamma=2.0/sqrt(%variance)
equation standard x
# constant dur{1 to 4}
equation transit x
# constant dur{1 to 4}
```

This allows for four lags in each AR branch. This guesses $c = 0$ (probably a good guess even if we hadn't already fit a sharp-cutoff TAR model) and γ as a scale of the reciprocal standard deviation. The following transforms the two linear equations into FRML's, with the VECTOR PHI1 as the free parameters for the STANDARD formula and PHI2 for the TRANSIT formula.

```
frml(equation=standard,vector=phi1,parmset=stdparms) phi1f
frml(equation=transit ,vector=phi2,parmset=trnparms) phi2f
frml star dur = g=glstar,phi1f+g*phi2f
```

As suggested above, this estimates first the model holding the two STAR parameters fixed at their guess values, then estimates with the full parameter set producing Table 6.2.

```
compute regparms=stdparms+trnparms
nonlin(parmset=starparms) gamma c
nlls(parmset=regparms,frml=star) dur
nlls(parmset=regparms+starparms,frml=star) dur
```

This graphs (Figure 6.2) the transition function against the threshold. This is an important step, as some failures of the model can be immediately apparent in this graph without being recognizable in the output. This is what you would hope to see in an LSTAR—a smooth transition from a 0 value at the left to a 1 value at the right. It's possible that you can get a very steep, almost step function, which means that the transition appears to be more “sharp” (as in Chapter 4) than smooth, which is certainly possible, as that's a special case. However, you might want to check some different guess values (for c) to make sure you're not finding a local mode. A more pathological pattern is for the transition to never reach zero at the left or one at the right. That's means that c is either near the limits of or possibly even outside the observed values of the threshold. That sometimes means that there is no indication of an actual transition—you should try some other guess values for c to determine if this is just a local mode.

```
set test = glstar
set xtest = dur{1}
scatter(style=dots,hlabel="Change in UR",$
  ylabel="Transition Function",$
  footer="Transition Function in LSTAR for Unemployment Rate")
# xtest test
```

6.3 Forecasts and Impulse Responses

For STAR models, forecasting and impulse response calculations are done using very similar calculations to those for TAR models in Sections 4.3 and 4.4. Example 6.2 adds the calculation of error bands. You may have noticed that

Table 6.2: STAR Estimates for Unemployment Rate

Nonlinear Least Squares - Estimation by Gauss-Newton

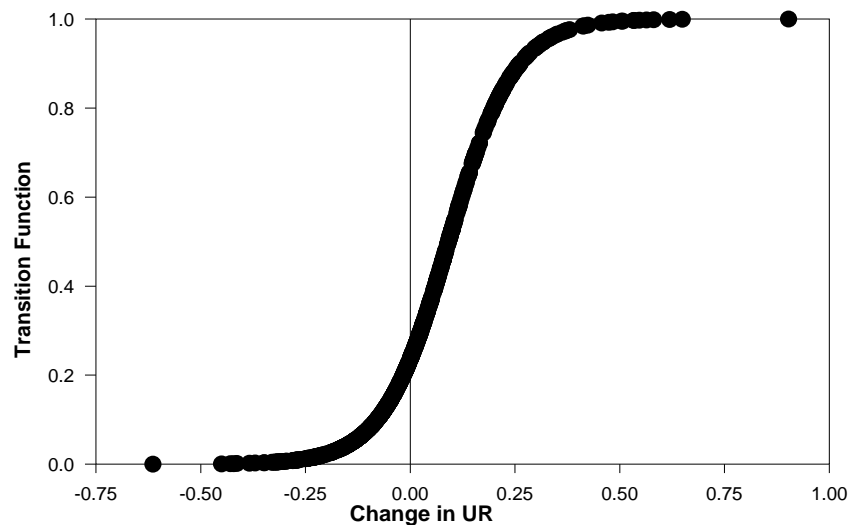
Convergence in 28 Iterations. Final criterion was 0.0000042 \leq 0.0000100

Dependent Variable DUR

Monthly Data From 1960:02 To 2013:06

Usable Observations	637
Degrees of Freedom	625
Skipped/Missing (from 641)	4
Centered R^2	0.1974
$R\text{-Bar}^2$	0.1832
Uncentered R^2	0.1978
Mean of Dependent Variable	0.0038
Std Error of Dependent Variable	0.1717
Standard Error of Estimate	0.1552
Sum of Squared Residuals	15.0491
Regression F(11,625)	13.9710
Significance Level of F	0.0000
Log Likelihood	289.0626
Durbin-Watson Statistic	1.9979

	Variable	Coeff	Std Error	T-Stat	Signif
1.	PHI1(1)=Constant	0.0162	0.0472	0.3433	0.7315
2.	PHI1(2)=DUR{1}	0.0056	0.1595	0.0353	0.9719
3.	PHI1(3)=DUR{2}	0.0982	0.0911	1.0787	0.2811
4.	PHI1(4)=DUR{3}	0.1914	0.0614	3.1166	0.0019
5.	PHI1(5)=DUR{4}	0.1256	0.0585	2.1473	0.0322
6.	PHI2(1)=Constant	-0.1083	0.0956	-1.1336	0.2574
7.	PHI2(2)=DUR{1}	0.4037	0.1552	2.6017	0.0095
8.	PHI2(3)=DUR{2}	0.3093	0.1505	2.0545	0.0403
9.	PHI2(4)=DUR{3}	-0.1195	0.1200	-0.9953	0.3200
10.	PHI2(5)=DUR{4}	0.0057	0.1173	0.0489	0.9610
11.	GAMMA	12.8746	11.2948	1.1399	0.2548
12.	C	0.0911	0.0843	1.0812	0.2800

**Figure 6.2:** LSTAR Transition Function for Unemployment Rate

the examples of impulse responses in Section 4.4 did not include these. The simulations there were required just to estimate the “point” values of the IRF’s, values which can be computed exactly for a linear model. Error bands are designed to take into account that the model’s parameters aren’t known with certainty. Thus, we need to do an entire other loop enclosing the calculation of the NL-IRF to generate different parameters so we can see how this affects the responses. Because the model itself is non-linear, the draws for the parameters isn’t as simple as it is for a linear model like a VAR. The choices are to do bootstrapping or some form of MCMC (Appendix D). Here, we choose MCMC because it can more easily handle the uncertainty in c and γ parameters. Note that this is *not* simple—if this is your first exposure to Metropolis-Hastings, you might want to try a simpler example first. (The RATS *Bayesian Econometrics* e-source is a good place to start).

Example 6.2 estimates a model on the well-known Canadian lynx data (Figure 6.3). This has annual data on the population of lynx (a small wild cat) in a region of Canada from 1821 to 1934. This is a standard data set in non-linear time series modeling because it has a fairly obvious “cycle”, but the down parts of the cycle are shorter and steeper than the up parts.

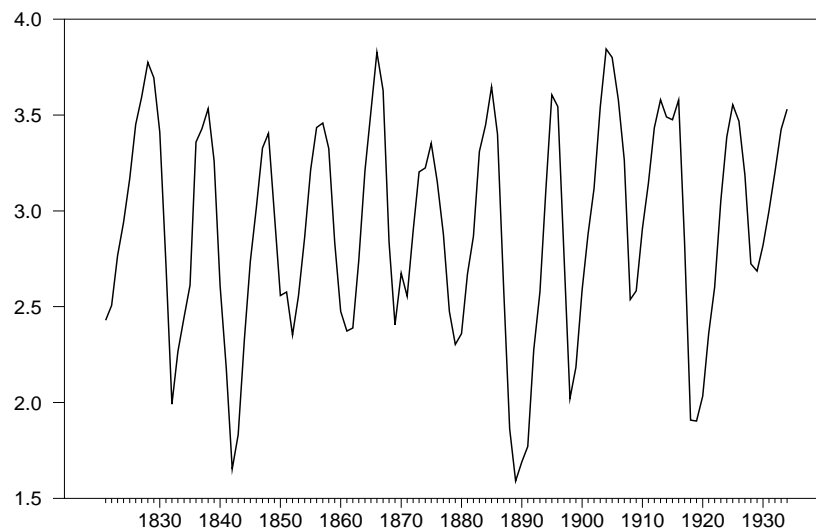


Figure 6.3: Canadian Lynx Data

The estimation of the STAR model for this is covered in detail in the RATS *User’s Guide* (example `TARMODEL.RPF`). We’ll start with the estimation of the chosen model, which is an LSTAR using a delay of lag 3, with an AR(1) on the left branch and a transition AR with a broken set of lags. Note that this uses a rescaling of γ using an approximate standard error of the threshold series (as was done in the original Terasvirta paper).

```

stats x
compute scalef=1.8
nonlin(parmset=starparms) gamma c
frml flstar = %logistic(scalef*gamma*(x{3}-c),1.0)
compute c=%mean,gamma=2.0

equation standard x
# x{1}
equation transit x
# x{2 3 4 10 11}
frml(equation=standard,vector=phi1,parmset=stdparms) phi1f
frml(equation=transit ,vector=phi2,parmset=trnparms) phi2f
frml star x = f=flstar,phi1f+f*phi2f
compute regparms=stdparms+trnparms
*
nlls(parmset=regparms,frml=star) x
nlls(parmset=regparms+starparms,frml=star) x

```

A NL-IRF to a positive one-standard deviation shock starting in 1925:1 is computed using

```

group starmodel star
*
compute ndraws=5000
compute nsteps=40
compute istart=1925:1
compute iend =istart+nsteps-1
*

compute stddev=sqrt(%seesq)
compute delta =1.0*stddev
*
set nlirf istart iend = 0.0

do draw=1,ndraws
  set shocks istart iend = %ran(stddev)
  forecast(paths,model=starmodel,from=istart,to=iend,$
    results=basesims)
  # shocks
  compute ishock=delta
  compute shocks(istart)=ishock
  forecast(paths,model=starmodel,from=istart,to=iend,$
    results=sims)
  # shocks
  set nlirf istart iend = nlirf+(sims(1)-basesims(1))
end do draw

```

For doing confidence bands, we'll use Independence Metropolis-Hastings, which seems to work well in this case. The proposal density is based upon the non-linear least squares estimates of the model, with a mean equal to the point estimates, and a multivariate t density using the estimated covariance matrix off NLLS and 10 degrees of freedom. (FXX is the Cholesky factor of the estimated covariance matrix, which is what we need for drawing from a multivariate t density).

```
compute allparms=regparms+starparms
compute fxx=%decomp(%seesq*%xx)
compute nuxx=10
```

We'll use a flat prior on the parameters themselves, with a loose scaled inverse chi-squared prior (Appendix F.4) on the variance of the residuals.

```
compute s2prior=1.0/10.0
compute nuprior=5.0
```

We'll start the sampler at the NLLS estimates, so the log kernel of the proposal density starts at 0.

```
compute bdraw=%beta
compute logqlast=0.0
```

The final step in the setup is to create a place for the draws. We'll save the entire simulated IRF for each draw, so we can do percentiles.

```
dec series[vect] nlirfs
gset nlirfs 1 ndraws = %zeros(nsteps,1)
```

The pseudo-code for the draw loop is

```
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Independence MH"
do draw=-nburn,ndraws
  draw residual precision conditional on current parameters
  draw parameters from the proposal density
  do the Metropolis test to see if proposal is accepted
  update the infobox
  if draw>0
    compute and save the IRF at the current parameters
  end do draw
infobox(action=remove)
```

One major issue here is that the NL-IRF at each set of draws is subject to its *own* simulation error. While the point estimates of the NL-IRF with 5000 draws take a small amount of time, putting that many inside a loop over (in this example) 6000 draws for the coefficients (5000 plus 1000 burn-in) will take considerably

longer. What you would like is to make (just) enough draws that the simulation error in the NL-IRF given the parameters is really small compared to the differences in the responses themselves due to the draws for the parameters. In this case, it seems that 500 “subdraws” is sufficient for that.

The details of the calculation inside the loop follow. This does a standard draw for the residual precision (reciprocal of the variance) given the current draws for the parameters.

```
compute %parmspoke(allparms,bdraw)
sstats rstart rend (x-star(t))^2>>rssbeta
compute rssplus=nuprior*s2prior+rssbeta
compute hdraw  =%ranchisqr(nuprior+%nobs)/rssplus
```

This draws a proposal, saves the log kernel of the draw, and computes the sum of squared residuals at the draw into RSSTEST. The log kernel of the posterior (since the parameters have been given a flat prior and the residuals are assumed to be Normal) is $-\frac{RSS}{2\sigma^2}$ since everything else cancels out in comparing two posteriors (note, by definition, the precision $h = \sigma^{-2}$). ALPHA is then the Metropolis acceptance probability for an independence chain.

```
compute btest=%beta+%ranmvt(fxx,nuxx)
compute logqtest=%ranlogkernel()
compute %parmspoke(allparms,btest)
sstats rstart rend (x-star(t))^2>>rsstest
compute logptest=-.5*hdraw*rsstest
compute logplast=-.5*hdraw*rssbeta
compute alpha  =exp(logptest-logplast+logqlast-logqtest)
```

This determines whether or not we accept the proposal. If so, the ACCEPT variable is updated, BDRAW is replace with the proposal and the log kernel of the last draw is replaced with the log kernel of the proposal. If it is not accepted, the old values just fall through.

```
if %ranflip(alpha) {
  compute bdraw=btest, accept=accept+1
  compute logqlast=logqtest
}
```

This updates the info box with a running calculation of the probability of acceptance. For independence Metropolis, a high acceptance rate is good—this runs in the 30-40% range, which is more than adequate.

```
infobox(current=draw) %strval(100.0*accept/(draw+nburn+1), "##.##")
if draw<=0
  next
```

This is pretty much a repeat of the one-off calculation of the NL-IRF. Note that this is only necessary once we’re past the burn-in (thus the test for DRAW<=0

in the code above. Because the “shock” is being defined as a one standard deviation shock (not a fixed size), the δ adjusts to the current draw for the precision.

```
compute stddev=sqrt(1.0/hdraw)
compute delta=1.0*stddev
set nlirf istart iend = 0.0
do sim=1,nsims
  set shocks istart iend = %ran(stddev)
  forecast(paths,model=starmodel,results=basesims,$
    from=istart,to=iend)
  # shocks
  compute ishock=delta
  compute shocks(istart)=ishock
  forecast(paths,model=starmodel,results=sims,$
    from=istart,to=iend)
  # shocks
  set nlirf istart iend = nlirf+(sims(1)-basesims(1))
end do sims
```

And this “flattens” this estimate of the NL-IRF into the DRAW element in the full history. (Flattening takes the separate elements from the series and packs them into a VECTOR).

```
ewise nlirfs(draw)(i)=nlirf(i+istart-1)/nsims
```

That finishes the process of drawing the NL-IRF’s. This does the post-processing to pull out the full history of the draws for the responses at each step (into the VECTOR called WORK), computing the three desired percentiles (16-50-84) and putting them into the proper location in the series for the lower (16%-ile), upper(84%-ile) and median.

```
set median istart iend = 0.0
set lower  istart iend = 0.0
set upper  istart iend = 0.0
*
dec vect work(ndraws)
do time=istart,iend
  ewise work(i)=nlirfs(i)(time+1-istart)
  compute ff=%fractiles(work,||.16,.50,.84||)
  compute lower(time)=ff(1)
  compute upper(time)=ff(3)
  compute median(time)=ff(2)
end do time
```

The graph (Figure 6.4) is done with

```
graph(number=0, footer="NL-IRF with 16-84% confidence band") 3
# median istart iend
# lower  istart iend 2
# upper  istart iend 2
```

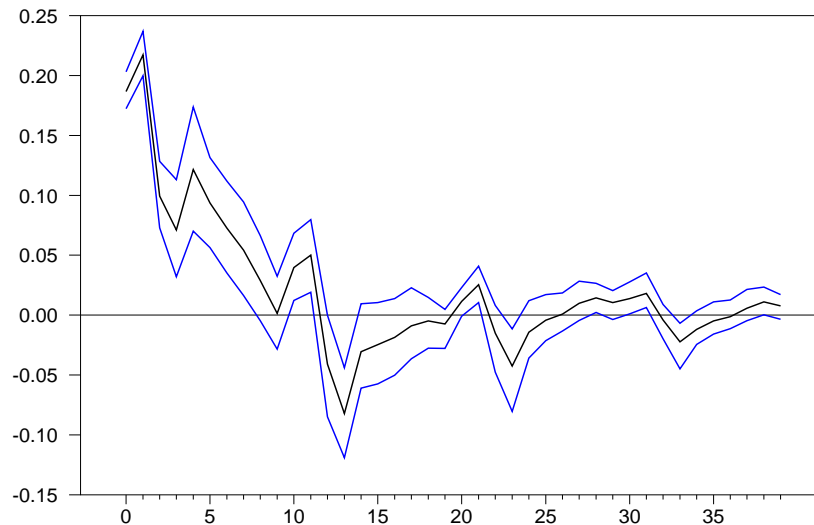


Figure 6.4: Non-linear IRF for STAR Model with Confidence Bands

6.4 More Complicated Models

An alphabet soup of smooth transition models such as STVAR (vector autoregression), STVECM (vector error correction model), STGARCH (GARCH) and combinations such as STAR-STGARCH (smooth transition mean model with smooth transition GARCH variance model) have been proposed in the literature. As a general rule, these have not been particularly successful, and most can be probably best be described as an estimator in search of an application. If you have an interest in these, you might want to take a look at Chan & McAleer (2003), for which we have a replication, which is an (unfortunately) rare paper which shows how complicated “trendy” models can fail, and fail badly. A major problem with this whole literature is that most of these models have multiple transition functions, for instance, the standard description of the STVAR has a separate smooth transition for each equation (different threshold values, sometimes even different threshold variables). However, there’s no reason to suspect that each equation will actually need that, or, if it has a transition, that it should be smooth. Either failure on any single equation can make it impossible to estimate the model.

Example 6.1 LSTAR Model: Testing and Estimation

This fits an LSTAR model to the change in the unemployment rate—the series treated using a sharp-cutoff TAR in Example 4.1. Technical information is in Sections 6.1 (for testing) and 6.2 (for estimation).

```

calendar(m) 1960:1
open data unrate.xls
data(format=xls,org=columns) 1960:01 2013:6
*
set dur = unrate-unrate{1}
*
linreg dur
# constant dur{1 2 3 4}
*
do d=1,4
  @startest(d=d,p=4,print) dur
end do d
*
nonlin(parmset=starparms) gamma c
frml glstar = %logistic(gamma*(dur{1}-c),1.0)
stats(noprint) dur
compute c=0.0,gamma=2.0/sqrt(%variance)
equation standard x
# constant dur{1 to 4}
equation transit x
# constant dur{1 to 4}
*
frml(equation=standard,vector=phi1,parmset=stdparms) phi1f
frml(equation=transit ,vector=phi2,parmset=trnparms) phi2f
frml star dur = g=glstar,phi1f+g*phi2f
*
compute regparms=stdparms+trnparms
nonlin(parmset=starparms) gamma c
nlls(parmset=regparms,frml=star) dur
nlls(parmset=regparms+starparms,frml=star) dur
*
* Graph the transition function against the threshold.
*
set test = glstar
set xtest = dur{1}
scatter(style=dots,hlabel="Change in UR",vlabel="Transition Function",$
  footer="Transition Function in LSTAR for Unemployment Rate")
# xtest test

```

Example 6.2 LSTAR Model: Impulse Responses

This is based upon example 3 from Terasvirta (1994). It estimates the model chosen in the RATS `TARMODELS.RPF` replication file, then computes the non-linear IRF's and and confidence bands for it. This is discussed in Section 6.2.

```
cal 1821
open data lynx.dat
data(org=cols) 1821:1 1934:1 lynx
set x = log(lynx)/log(10)
graph(footer="Canadian Lynx Data (in logs)")
# x
*
* This uses the restricted model already selected.
*
stats x
compute scalef=1.8
*
nonlin(parmset=starparms) gamma c
frml flstar = %logistic(scalef*gamma*(x{3}-c),1.0)
compute c=%mean,gamma=2.0
*
equation standard x
# x{1}
equation transit x
# x{2 3 4 10 11}
frml(equation=standard,vector=phi1,parmset=stdparms) phi1f
frml(equation=transit ,vector=phi2,parmset=trnparms) phi2f
frml star x = f=flstar,phi1f+f*phi2f
compute regparms=stdparms+trnparms
*
nlls(parmset=regparms,frml=star) x
nlls(parmset=regparms+starparms,frml=star) x
*
compute rstart=12,rend=%regend()
*
* One-off NL-IRF
*
group starmodel star
*
compute ndraws=5000
compute nsteps=40
compute istart=1925:1
compute iend =istart+nsteps-1
*
compute stddev=sqrt(%seesq)
compute delta =1.0*stddev
*
set nlirf istart iend = 0.0
do draw=1,ndraws
    set shocks istart iend = %ran(stddev)
    forecast(paths,model=starmodel,from=istart,to=iend,$
        results=basesims)
```



```

    # shocks
    compute ishock=delta
    compute shocks(istart)=ishock
    forecast(paths,model=starmodel,from=istart,to=iend,$
             results=sims)
    # shocks
    set nlirf istart iend = nlirf+(sims(1)-basesims(1))
end do draw
*
set nlirf istart iend = nlirf/ndraws
graph(number=0,footer="NL-IRF for Lynx at "+%datelabel(istart))
# nlirf istart iend
*
* NL-IRF with confidence bands
*
* Independence Metropolis-Hastings. Draw from multivariate t centered at
* the NLLS estimates. In order to do this most conveniently, we set up a
* VECTOR into which we can put the draws from the standardized
* multivariate t.
*
compute accept=0
compute ndraws=5000
compute nburn =1000
compute nsims =500
*
* Proposal density for the regression parameters
*
compute allparms=regparms+starparms
compute fxx =%decomp(%seesq*%xx)
compute nuxx=10
*
* Prior for variance. We use a flat prior on the coefficients.
*
compute s2prior=1.0/10.0
compute nuprior=5.0
*
* Since we're starting at %BETA, the kernel of the proposal density is 1.
*
compute bdraw=%beta
compute logqlast=0.0
*
dec series[vect] nlirfs
gset nlirfs 1 ndraws = %zeros(nsteps,1)
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Independence MH"
do draw=-nburn,ndraws
  *
  * Draw residual precision conditional on current bdraw
  *
  compute %parmspoke(allparms,bdraw)
  sstats rstart rend (x-star(t))^2>>rssbeta
  compute rssplus=nuprior*s2prior+rssbeta
  compute hdraw =%ranchisqr(nuprior+%nobs)/rssplus

```

```

*
* Independence chain MC
*
compute btest=%beta+%ranmvt(fxx,nuxx)
compute logqtest=%ranlogkernel()
compute %parmspoke(allparms,btest)
sstats rstart rend (x-star(t))^2>>rsstest
compute logptest=-.5*hdraw*rsstest
compute logplast=-.5*hdraw*rssbeta
compute alpha =exp(logptest-logplast+logqlast-logqtest)
if %ranflip(alpha) {
    compute bdraw=btest,accept=accept+1
    compute logqlast=logqtest
}
infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.#")
if draw<=0
    next
compute stddev=sqrt(1.0/hdraw)
compute delta=1.0*stddev
set nlirf istart iend = 0.0
do sim=1,nsims
    set shocks istart iend = %ran(stddev)
    forecast(paths,model=starmodel,results=basesims,$
        from=istart,to=iend)
    # shocks
    compute ishock=delta
    compute shocks(istart)=ishock
    forecast(paths,model=starmodel,results=sims,$
        from=istart,to=iend)
    # shocks
    set nlirf istart iend = nlirf+(sims(1)-basesims(1))
end do sims
*
* Flatten this estimate of the NL-IRF to save as the <<draw>> element
* in the full history.
*
ewise nlirfs(draw)(i)=nlirf(i+istart-1)/nsims
end do draw
infobox(action=remove)
*
set median istart iend = 0.0
set lower istart iend = 0.0
set upper istart iend = 0.0
*
dec vect work(ndraws)
do time=istart,iend
    ewise work(i)=nlirfs(i)(time+1-istart)
    compute ff=%fractiles(work,||.16,.50,.84||)
    compute lower(time)=ff(1)
    compute upper(time)=ff(3)
    compute median(time)=ff(2)
end do time
*
graph(number=0,footer="NL-IRF with 16-84% confidence band") 3

```

```
# median istart iend  
# lower  istart iend 2  
# upper  istart iend 2
```

Mixture Models

Suppose that we have a data series y which can be in one of several possible (unobservable) regimes.¹ If the regimes are independent across observations, we have a (simple) *mixture model*. These are quite a bit less complicated than Markov mixture or Markov switching models (Chapter 8), where the regime at one time period depends upon the regime at the previous one, but illustrate many of the problems that arise in working with the more difficult time-dependent models. Simple mixture models are used mainly in cross-section data to model unobservable heterogeneity, though they can also be used in an error process to model outliers or other fat-tailed behavior.

To simplify the notation, we'll use just two regimes. We'll use S_t to represent the regime of the system at time t ,² and p will be the (unconditional) probability that the system is in regime 1. There's no reason that p has to be fixed, and generalizing it to be a function of exogenous variables isn't difficult.

If we write the likelihood under regime i as $f_{(i)}(y_t|X_t, \Theta)$, where X_t are exogenous and Θ are parameters, then the log likelihood element for observation t is

$$\log \{p f_{(1)}(y_t|X_t, \Theta) + (1 - p) f_{(2)}(y_t|X_t, \Theta)\} \quad (7.1)$$

Each likelihood element is a probability-weighted average of the likelihoods in the two regimes. This produces a sample likelihood which can show very bad behavior, such as multiple peaks. In the most common case where the two regimes have the same structure but with different parameter vectors, the regimes become interchangeable. The “labeling” of the regimes isn't defined by the model itself, so there are (in an n regime model) $n!$ identical likelihood modes—one for each permutation of the regimes. If the model is estimated by maximum likelihood, you will end up at one of these, and you can usually (but not always) control which you get by your choices of guess values. However, the problem of *label switching* is a very serious issue with *Bayesian* estimation.

There are three main ways to estimate a model like this: conventional maximum likelihood (ML), expectation-maximization (EM), and Bayesian Markov Chain Monte Carlo (MCMC). These have in common the need for the values for

¹We'll use *regime* rather than *state* for this to avoid conflict with the term state in the state-space model.

²We'll number these as 1 and 2 since that will generalize better to more than two regimes than a 0-1 coding.

$f_{(i)}(y_t|X_t, \Theta)$ across i for each observation. Our suggestion is that you create a **FUNCTION** to do this calculation, which will make it easier to make changes. The return value of the **FUNCTION** should be a **VECTOR** with size equal to the number of regimes. Remember that these are the likelihoods, not the more commonly used *log* likelihoods. If it's more convenient doing the calculation in log form, make sure that you *exp* the results before you return. As an example:

```
function RegimeF time
type vector RegimeF
type integer time
local integer i
dim RegimeF(2)
ewise RegimeF(i)=$
    exp(%logdensity(sigsq,%eqnrvalue(xeq,time,phi(i))))
end
```

In addition to the problem with label switching, there are other pathologies that may occur in the likelihood and which affect both ML and EM. One of the simplest cases of a mixture model has the mean and variance different in each regime. The log likelihood for observation t is

$$\log \{ p f_N(x_t - \mu_1, \sigma_1^2) + (1 - p) f_N(x_t - \mu_2, \sigma_2^2) \} \quad (7.2)$$

where $f_N(x, \sigma^2)$ is the normal density at x with variance σ^2 . At $\mu_1 = x_1$ (or any other data point), the likelihood can be made arbitrarily high by making σ_1^2 very close to 0. The other data points will, of course, have a zero density for the first term, but will have a non-zero value for the second, and so will have a finite log likelihood value. Thus, the likelihood function has very high “spikes” around values equal to the data points.

This is a pathology which occurs because of the combination of both the mean and variance being free, and will not occur if either one is common to the regimes. For instance, an “outlier” model would typically have all branches with a zero mean and thus can’t push the variance to zero at a single point. This problem with the likelihood was originally studied in Kiefer & Wolfowitz (1956) and further examined in Kiefer (1978). The latter paper shows that an interior solution (with the variance bounded away from zero) has the usual desirable properties for a maximum likelihood estimator. To estimate the model successfully, we have to somehow eliminate any set of parameters with unnaturally small variances.

In addition to the narrow range of values at which the likelihood is unbounded, it is possible (and probably likely) that there will be multiple modes. With both ML and EM, it’s important to test alternative starting values.

We’ll employ the three methods to estimate a model of fish sizes used in Fruehwirth-Schnatter (2006). This has length measurements on 256 fish which are assumed to have sizes which vary with an (unobservable) age. Models with three and four categories are examined in most cases.

7.1 Maximum Likelihood

The log likelihood function is just the sum of (7.1) across observations. The parameters can be estimated by **MAXIMIZE**. There are three main issues: first, as mentioned above, the regimes aren't really defined by the model if the density functions take the same form. Generic guess values thus won't work—you have to *force* a separation by giving different guess values to the branches. Also, there's nothing in the log likelihood function that forces p to remain in the interval $[0, 1]$. With two regimes, that usually doesn't prove to be a problem, but constraining the p parameter might be necessary in more complicated models. The usual way of doing that is to model it as a logistic, with $p = 1 - (1 + \exp(\theta))^{-1}$ for the two branch model. The third issue is the avoidance of the zero-variance spikes, which can be done by using the **REJECT** option to test for very small values of the variances.

Maximum likelihood is always the simplest of the three estimation methods to set up. However, in most cases, it's the slowest (unless you use a very large number of draws on the Bayesian method). The best idea is generally to try maximum likelihood first, and go to the extra trouble of EM only if the slow speed is a problem.

For a model with just means and variances, the following are the parameters for **NCATS** categories:

```
dec vect mu(ncats) sigma(ncats) p(ncats)
dec vect theta(ncats-1)
nonlin(parmset=mixparms) mu sigma theta
```

The function for the regime-dependent likelihoods (which will change slightly from one application to another) is:

```
function RegimeF time
type vector RegimeF
type integer time
dim RegimeF(ncats)
ewise RegimeF(i)=exp(%logdensity(sigma(i)^2,fish(time)-mu(i)))
end
```

THETA is the vector of logistic indexes for the probabilities, which will map to the working **P** vector. Because it's possible for $\exp x$ to overflow in cases where the final regime has a true probability near zero, the mapping function subtracts off the maximum value of θ from all calculations before taking the \exp :

```

function %MixtureP theta
type vect theta %MixtureP
local real maxtheta
*
dim %MixtureP(%rows(theta)+1)
*
* Avoid overflow problems by using deviations from the maximum
* theta.
*
compute maxtheta=%maxvalue(theta)
ewise %MixtureP(i)=%if(i<=%rows(theta), exp(theta(i)-maxtheta), $
    exp(-maxtheta))
compute %MixtureP=%MixtureP/%sum(%MixtureP)
end

```

Given those building blocks, the log likelihood is the very simple:

```
frml mixture = fp=%dot(p, RegimeF(t)), log(fp)
```

The estimation is done with **MAXIMIZE** with something like:

```

maximize(start=(p=%MixtureP(theta)), parmset=mixparms, $
    reject=%minvalue(sigma)<sigmalimit, $
    pmethod=simplex, peters=5) mixture gstart gend

```

The **START** option transforms the free parameters in **THETA** to the working **P** vector. The **REJECT** option prevents the smallest value of **SIGMA** from getting too close to zero. Because the likelihood goes unbounded only in a very narrow zone around the zero variance, the limit can be quite small—in this case, we made it a very small fraction of the interquartile range.

```

stats(fractiles) fish
compute sigmalimit=.00001*(%fract75-%fract25)

```

Example 7.1 estimates both a three regime model (Table 7.1) and a four regime model (Table 7.2). The three regime model seems to be capable of identifying the first two cohorts of young fish (fairly low uncertainty about the mean, and a small estimated variance), while the remainder are lumped together into a broad, relatively poorly defined group. The four regime model picks out a third cohort, leaving a smaller, and even more poorly defined remainder.

Note that, as with models in previous sections, the likelihood ratio test for three regimes vs four will not have a standard asymptotic chi-squared distribution because the four regime model isn't identified under the null that the three regime model is correct. There's a thread in the econometrics literature which applies the Davies (1987) adjusted significance level to this, though it's not at

Table 7.1: Three regime mixture model

MAXIMIZE - Estimation by BFGS					
Convergence in 32 Iterations. Final criterion was 0.0000090 <= 0.0000100					
Usable Observations		256			
Function Value		-494.6748			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	MU(1)	3.2092	0.0622	51.5585	0.0000
2.	MU(2)	5.1610	0.0705	73.1733	0.0000
3.	MU(3)	7.2087	0.2798	25.7608	0.0000
4.	SIGMA(1)	0.2455	0.0496	4.9494	0.0000
5.	SIGMA(2)	0.4914	0.0736	6.6783	0.0000
6.	SIGMA(3)	1.8155	0.1359	13.3591	0.0000
7.	THETA(1)	-1.8371	0.2976	-6.1737	0.0000
8.	THETA(2)	-0.3805	0.2882	-1.3205	0.1867

all clear that it's appropriate.³ There are methods for estimating the number of regimes which depend upon simulation methods to compute the marginal likelihoods of each possibility. Among methods which can be and are used that require only the log likelihoods, the information criteria (generally AIC or BIC) are the simplest choices. Using those, AIC slightly favors the four regime, and BIC/SBC slightly favors the three.

If you want to compute the actual probabilities of the regimes, you can use **SUMMARIZE** to compute them and their standard errors with (after the three regimes):

```
summarize(parmset=mixparms,title="P(S=1)", $
  numerical) %MixtureP(theta)(1)
summarize(parmset=mixparms,title="P(S=2)", $
  numerical) %MixtureP(theta)(2)
summarize(parmset=mixparms,title="P(S=3)", $
  numerical) %MixtureP(theta)(3)
```

In cases like this (where the classification of a given observation into one of the regimes is rather clear), the probabilities are almost exactly the fraction of the observations which would classify into each regime.

7.2 EM Estimation

The EM algorithm (Appendix B) is able to simplify the calculations quite a bit in these models. The augmenting parameters x are the regimes. In the M step,

³Davies' theorem never mentions adjustments for asymptotic chi-squared tests, only for exact finite sample tests. And even then, it's not clear how it would apply to any test beyond one regime vs two.

Table 7.2: Four Regime Mixture Model

MAXIMIZE - Estimation by BFGS
Convergence in 32 Iterations. Final criterion was 0.0000000 <= 0.0000100

Usable Observations	256
Function Value	-488.5858

	Variable	Coeff	Std Error	T-Stat	Signif
1.	MU(1)	3.2130	0.0568	56.5310	0.0000
2.	MU(2)	5.2596	0.0712	73.8561	0.0000
3.	MU(3)	7.4428	0.1126	66.0710	0.0000
4.	MU(4)	8.6183	1.1978	7.1949	0.0000
5.	SIGMA(1)	0.2513	0.0440	5.7084	0.0000
6.	SIGMA(2)	0.6274	0.0636	9.8714	0.0000
7.	SIGMA(3)	0.4538	0.1459	3.1113	0.0019
8.	SIGMA(4)	1.7589	0.4933	3.5656	0.0004
9.	THETA(1)	-0.7377	0.6538	-1.1284	0.2592
10.	THETA(2)	0.9954	0.6962	1.4297	0.1528
11.	THETA(3)	-0.1515	1.0072	-0.1504	0.8804

we maximize over Θ the simpler

$$E_{\{S_t|\Theta_0\}} \log f(y_t, S_t|X_t, \Theta) = E_{\{S_t|\Theta_0\}} \{\log f(y_t|S_t, X_t, \Theta) + \log f(S_t|X_t, \Theta)\} \quad (7.3)$$

For the typical case where the underlying models are linear regressions, the two terms on the right use separate sets of parameters, and thus can be maximized separately. Maximizing the sum of the first just requires a probability-weighted linear regression, while maximizing the second estimates p as the average of the probabilities of the regimes. p is constructed to be in the proper range, so we don't have to worry about that as we might with maximum likelihood. The value of the E step is that it allows us to work with sums of the more convenient log likelihoods rather than logs of the sums of the likelihoods as in (7.1).

In Example 7.2, we use a `SERIES[VECT]` to hold the estimated probabilities of the regimes using the previous parameter settings. For a two-regime model, we *could* get by with just a single series with the probabilities of regime 1, knowing that regime 2 would have one minus that as the probability. However, the more general way of handling it is, in fact, simpler to write. The setup for this is

```
dec series[vector] pt_t
gset pt_t gstart gend = %fill(ncats,1,1.0/ncats)
```

The second line just initializes all the elements—the values don't really matter because the first step in EM is to compute the probabilities of these anyway.

The E step just uses Bayes rule to fill in those probabilities. This is computed using the previous parameter settings for the means and variances and for the unconditional probabilities of the branches:

```
gset pt_t gstart gend = f=RegimeF(t), (f.*p)/%dot(f,p)
```

The M step is, in this case, most conveniently done by using the **SSTATS** instruction to compute probability-weighted sums and sums of squares along with the sum of the probabilities themselves. This requires a separate calculation for each of the possible regimes. Because `PT_T` is a `SERIES[VECT]`, you first have to reference the time period (thus `PT_T(T)`) and then further reference the element within that `VECTOR`, which is why you use `PT_T(T)(I)` to get the probability of regime `I` at time `T`.

```
do i=1,ncats
  sstats gstart gend pt_t(t)(i)>>sumw pt_t(t)(i)*fish>>sumwm $
    pt_t(t)(i)*fish^2>>sumwmsq
  compute p(i)=sumw/%nobs
  compute mu(i)=sumwm/sumw
  compute sigma(i)=%max(sqrt(sumwmsq/sumw-mu(i)^2),sigmalimit)
end do i
```

Note that, like ML, this also has a limit on how small the variance can be. The EM iterations can “blunder” into one of the small variance spikes if nothing is done to prevent it.

The full EM algorithm requires repeating those steps, so this is enclosed in a loop. At the end of each step, the log likelihood is computed—this should increase with each iteration, generally rather quickly at first, then slowly crawling up.

```
sstats gstart gend log(%dot(RegimeF(t),p))>>logl
disp "Iteration" emits logl p mu sigma
```

In Example 7.2, we do 50 EM iterations to improve the guess values, then switch to maximum likelihood. With straight ML, it takes 32 iterations of ML after some simplex preliminary iterations; with the combination of EM plus ML, it takes just 9 iterations of ML to finish convergence, and less than half the execution time. The EM iterations also tend to be a bit more robust to guess values, although if there are multiple modes (which is likely) there’s no reason that EM can’t home in on one with a lower likelihood.

The point estimates are effectively identical—the final standard errors are somewhat different because Example 7.1 uses BFGS and Example 7.2 used BHHH as it’s starting with nearly converged estimates and thus doesn’t have enough iterations to estimate the curvature.

7.3 Bayesian MCMC

As with EM, the regime variables are now treated as parameters. However, now we *draw* values for them as part of a Gibbs sampler. The repeated steps are (in some order) to draw Θ given S_t and p , then S_t given Θ and p , then p given S_t .

In Example 7.3, the first step is to draw the sigma given the regimes and the means. The standard prior for a variance is a scaled inverse chi-squared (Appendix F.4).⁴ In the Fruehwirth-Schnatter book, the choice was a very loose prior with one degree of freedom (NUDF is the variable for that) and a mean of one (NUSUMSQR is the mean times NUDF). You can even use a non-informative prior with zero degrees of freedom, as there is effectively a zero probability of getting a draw with the variance so small that you catch a spike. **SSTATS** is used to compute the sum of squared residuals (and the number of covered observations as a side effect); these are combined with the prior to give the parameters for an inverse chi-squared draw for the variance (Appendix F.4).

```
do i=1,ncats
  sstats(smpl=(s==i)) gstart gend (fish-mu(i))^2>>sumsqr
  compute sigma(i)=sqrt((sumsqr+nusumsqr)/%ranchisqr(%nobs+nudf))
end do i
```

Next up is drawing the means given the variances and the regimes. In this example, we use a flat prior on the means—we'll discuss that choice in Section 7.3.1. Given the standard deviations just computed, the sum and observation count are sufficient statistics for the mean, which are drawn as normals.

```
do i=1,ncats
  sstats(smpl=(s==i)) gstart gend fish>>sum
  compute mu(i)=sum/%nobs+%ran(sigma(i)/sqrt(%nobs))
end do i
```

For now, we'll skip over the next step in the example (relabeling) and take that up in section 7.3.1. We'll next look at drawing the regimes given the other parameters. Bayes formula gives us the relative probabilities of regime i at time t as the product of the unconditional probability $p(i)$ times the likelihood of regime i at t . The **%RANBRANCH** function is designed precisely for drawing a random index from a vector of relative probability weights.⁵ A single instruction does the job:

```
set s gstart gend = fxp=RegimeF(t).*p,%ranbranch(fxp)
```

Finally, we need to draw the unconditional probabilities given the regimes. For two regimes, this can be done with a beta distribution (Appendix F.2), but for

⁴See the first page in Appendix C for the derivation of this.

⁵Note that you don't need to divide through by the sum of f times p —**%RANBRANCH** takes care of the normalization.

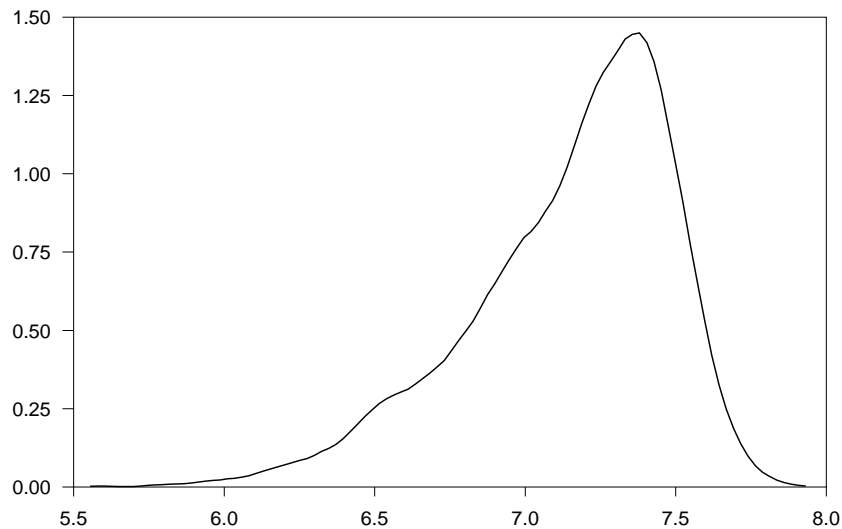


Figure 7.1: MCMC Density for 3rd category

more than that, we need the more general Dirichlet distribution (Appendix F.3). For an n component distribution, this takes n input shapes and returns an n vector with non-negative components summing to one. The counts of the number of the regimes are combined with a weak Dirichlet prior (all components are 4; the higher the values, the tighter the prior). An uninformative prior for the Dirichlet would have input shapes that are all zeros. However, that isn't recommended, as it is possible for a sweep to generate no data points in a particular regime—a non-zero prior makes sure that the unconditional probability doesn't also collapse to zero.

```
do i=1,ncats
  sstats(smpl=(s==i)) gstart gend 1>>shapes(i)
end do i
compute p=%randirichlet(shapes+prior)
```

The MCMC loop saves the simulated values of the means and variances from each of the kept draws. Outside the loop, the program generates density graphs for the mean parameters. All of them are in line with what you would expect from the ML estimates except for the 3rd regime (Figure 7.1) which has a considerable skew. The maximum likelihood estimates have this at 7.44 with a standard error of .112, where the latter seems rather optimistic given the density graph.

An interesting graph shows the simulated mean and variance combinations for the different regimes (Figure 7.2). As you can see from this, the first regime is rather well-separated, which means that it's relatively easy to identify a year-old fish by its size. The second regime is not quite as well-defined, and the third regime isn't well-defined at all. A likely reason for this is that it's not clear with four regimes whether the third regime is 3 and 4 year old fish, with the fourth then being 5 and up, or whether the third regime is 3 years old only and 4 and

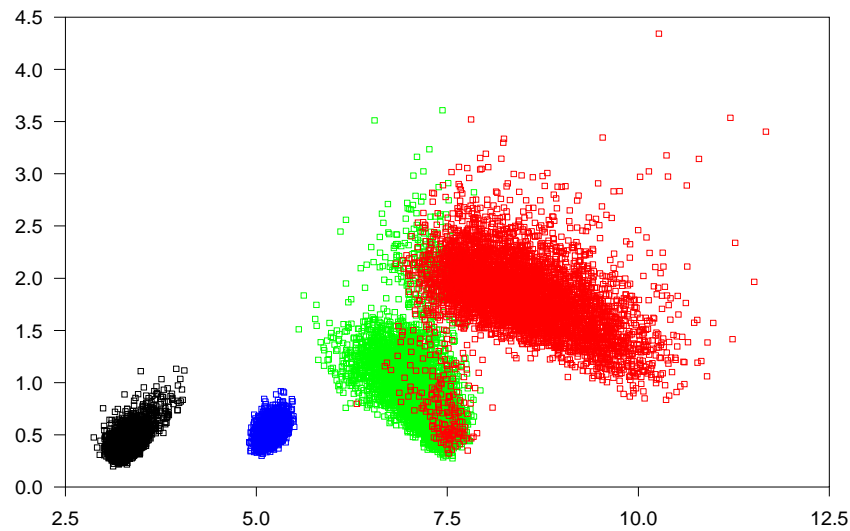


Figure 7.2: MCMC Mean-Variance Combinations by Regime

up are in the fourth regime—by the time you get into age 3 fish, it seems that the individual differences are starting to get large relative to the differences due to age.

7.3.1 Label Switching

The problem of label switching in Gibbs sampling in this type of model has been underappreciated. One way to avoid it is to use a very tight prior on the means in the regimes to make draws which switch positions nearly impossible. However, you can't make it completely impossible with *any* Normal prior, and a prior tight enough to prevent label switching is probably more informative than we would generally prefer. It's possible to use a non-Normal prior which ensures that the regime two mean is greater than regime one, and regime three greater than regime two, etc., but that will be considerably more complicated since it's not the natural prior for Normally distributed data. It also forces a difference which might not actually exist (we could be allowing for more regimes than are needed) so the different apparent modes might be an artifact of the prior.

An alternative is described in Fruehwirth-Schnatter (2006). Instead of trying to force the desired behavior through the prior (which probably won't work properly), it uses a prior which is identical across regimes, thus making the posterior identical for all permutations. Then the definitions of the regimes is corrected in each sweep to achieve a particular ordering. This requires swapping the values of the regimes, the probability vectors, variances and means (or regression coefficient vectors in general). The relabeling code is fairly simple in this case, because the `%INDEX` function can be used to get a sorting index for the mean vector. After `SWAPS=%INDEX(MU)`, `SWAPS(1)` has the index of the smallest element of `MU`, `SWAPS(2)` the index of the second smallest, etc. The

following corrects the ordering of the `MU` vector—similar operations are done for `SIGMA` and `P`. Note that this copies `MU` to a temporary `VECTOR` first—you can't directly switch `MU` values around as **EWISE** operates an element at a time, so the original `MU(1)` would get overwritten before being used for its new location.

```
compute swaps=%index(mu)
compute temp=mu
ewise mu(i)=temp(swaps(i))
```

Because of the positioning of the relabeling step (after the draws for `MU`, `SIGMA` and `P`, but before the draws for the regimes), there is no need to switch the definitions of the regimes, since they will naturally follow the definitions of the others.

This type of correction is quite simple in this case since the regression part is just the single parameter, which can be ordered easily. With a more general regression, it might be more difficult to define the interpretations of the regimes.

Example 7.1 Mixture Model-Maximum Likelihood

Estimation of a mixture model by maximum likelihood, discussed in detail in Section 7.1. This estimates both three and four regime models.

```
open data fish_data.txt
data(format=prn,org=cols) 1 256 fish
*
* Use the sample quantiles to get guess values. Also get a lower
* limit on the sigma to prevent convergence to a spike.
*
stats(fractiles) fish
compute sigmalimit=.00001*(%fract75-%fract25)
compute gstart=1,gend=256
*
* 3 category Normal mixture model - maximum likelihood
*
compute ncats=3
dec vect mu(ncats) sigma(ncats) p(ncats)
dec vect theta(ncats-1)
nonlin(parmset=mixparms) mu sigma theta
*
* Give guess values which spread the means out. Also, make the
* initial guesses for sigma relatively small.
*
compute mu=||%fract05,%median,%fract95||
compute sigma=%fill(ncats,1,.1*sqrt(%variance))
compute theta=%zeros(ncats-1,1)
*****
function RegimeF time
type vector RegimeF
type integer time
dim RegimeF(ncats)
ewise RegimeF(i)=exp(%logdensity(sigma(i)^2,fish(time)-mu(i)))
end
*****
function %MixtureP theta
type vect theta %MixtureP
local real maxtheta
*
dim %MixtureP(%rows(theta)+1)
*
* Avoid overflow problems by using deviations from the maximum
* theta.
*
compute maxtheta=%maxvalue(theta)
ewise %MixtureP(i)=%if(i<=%rows(theta),exp(theta(i)-maxtheta),
exp(-maxtheta))
compute %MixtureP=%MixtureP/%sum(%MixtureP)
end
*****
frml mixture = fp=%dot(p,RegimeF(t)),log(fp)
*
```

```

maximize (start=(p=%MixtureP(theta)), parmset=mixparms, $
    reject=%minvalue(sigma)<sigmalimit, $
    pmethod=simplex, peters=5) mixture gstart gend
@regcrits(title="Three regime model")
*
summarize (parmset=mixparms, title="P (S=1) ", $
    numerical) %MixtureP(theta) (1)
summarize (parmset=mixparms, title="P (S=2) ", $
    numerical) %MixtureP(theta) (2)
summarize (parmset=mixparms, title="P (S=3) ", $
    numerical) %MixtureP(theta) (3)
*
* 4 category Normal mixture model - maximum likelihood
*
compute ncats=4
dec vect mu(ncats) sigma(ncats) p(ncats)
dec vect theta(ncats-1)
stats(fractiles) fish
compute mu=||%fract05,%fract25,%fract75,%fract95||
compute sigma=%fill(ncats,1,.1*sqrt(%variance))
compute theta=%zeros(ncats-1,1)
maximize (start=(p=%MixtureP(theta)), parmset=mixparms, $
    reject=%minvalue(sigma)<sigmalimit, $
    pmethod=simplex, peters=5) mixture gstart gend
@regcrits(title="Four regime model")

```

Example 7.2 Mixture Model-EM

Estimation of a mixture model by EM. Details are in Section 7.2. This does the three regime model only.

```

open data fish_data.txt
data(format=prn,org=cols) 1 256 fish
*
* Use the sample quantiles to get guess values. Also get a lower
* limit on the sigma to prevent convergence to a spike.
*
stats(fractiles) fish
compute sigmalimit=.00001*(%fract75-%fract25)
compute gstart=1,gend=256
*
* 3 category Normal mixture model - EM
*
compute ncats=3
dec vect mu(ncats) sigma(ncats) p(ncats)
*
compute mu=||%fract10,%median,%fract90||
compute sigma=%fill(ncats,1,.1*sqrt(%variance))
compute p      =%fill(ncats,1,1.0/ncats)
*****
function RegimeF time

```



```

type vector  RegimeF
type integer time
dim RegimeF(ncats)
ewise RegimeF(i)=exp(%logdensity(sigma(i)^2,fish(time)-mu(i)))
end

*****
*
* pt_t has the estimated probabilities of the regimes at each time
* period.
*
dec series[vect] pt_t
gset pt_t gstart gend = %fill(ncats,1,1.0/ncats)
*
do emits=1,50
  *
  * E-step given current guess values for sigma and mu (compute
  * probabilities of the regimes for each time period).
  *
  gset pt_t gstart gend = f=RegimeF(t),(f.*p)/%dot(f,p)
  *
  * M-step for probabilities, means and variances
  *
  do i=1,ncats
    sstats gstart gend pt_t(t)(i)>>sumw pt_t(t)(i)*fish>>sumwm $
      pt_t(t)(i)*fish^2>>sumwmsq
    compute p(i)=sumw/%nobs
    compute mu(i)=sumwm/sumw
    compute sigma(i)=%max(sqrt(sumwmsq/sumw-mu(i)^2),sigmalimit)
  end do i
  sstats gstart gend log(%dot(RegimeF(t),p))>>logl
  disp "Iteration" emits logl
end do emits

*****
*
* Maximum likelihood to polish estimates
*
function %MixtureP theta
type vect theta %MixtureP
local real maxtheta
*
dim %MixtureP(%rows(theta)+1)
*
* Avoid overflow problems by using deviations from the maximum theta.
*
compute maxtheta=%maxvalue(theta)
ewise %MixtureP(i)=%if(i<=%rows(theta),exp(theta(i)-maxtheta),
  exp(-maxtheta))
compute %MixtureP=%MixtureP/%sum(%MixtureP)
end

*****
*
* Because this is using variational methods to estimate the
* parameters, we use the logistic index for the probabilities.
*

```

```

dec vect theta(ncats-1)
nonlin(parmset=catparms) theta
nonlin(parmset=regparms) mu sigma
ewise theta(i)=log(p(i))-log(p(ncats))
*
* This is the standard log likelihood for ML
*
frml mixture = fp=%dot(p,RegimeF(t)),log(fp)
*
maximize(start=p=%MixtureP(theta),parmset=regparms+catparms,$
  reject=%minvalue(sigma)<sigmalimit,method=bhhh) $
  mixture gstart gend

```

Example 7.3 Mixture Model-MCMC

Estimation of a mixture model by Markov Chain Monte Carlo. Details in Section 7.3. This works with the four regime model only.

```

open data fish_data.txt
data(format=prn,org=cols) 1 256 fish
*
* Use the sample quantiles to get guess values. Also get a lower
* limit on the sigma to prevent convergence to a spike.
*
stats(fractiles) fish
compute gstart=1,gend=256
*
* 4 category model - MCMC
*
compute ncats=4
dec vect mu(ncats) sigma(ncats) p(ncats)
compute mu=||%fract10,%fract25,%median,%fract90||
compute sigma=%fill(ncats,1,.1*sqrt(%variance))
compute p=%fill(ncats,1,1.0/ncats)
*****
function RegimeF time
type vector RegimeF
type integer time
dim RegimeF(ncats)
ewise RegimeF(i)=exp(%logdensity(sigma(i)^2,fish(time)-mu(i)))
end
*****
dec vect shapes(ncats) priord(ncats) fxp(ncats)
set s gstart gend = fxp=RegimeF(t).*p,%ranbranch(fxp)
*
* Prior for unconditional probabilities
*
compute priord=%fill(ncats,1,4.0)
*
* Prior for variances
*

```

```

compute nusumsqr=1.0,nudf=1
*
* "Flat" prior is used for means
*
compute nburn=2000,ndraws=5000
*
* Bookkeeping arrays
*
dec vect[series] mus(ncats) sigmas(ncats)
do i=1,ncats
  set mus(i) 1 ndraws = 0.0
end do i
do i=1,ncats
  set sigmas(i) 1 ndraws = 0.0
end do i
*
infobox(action=define,lower=-nburn,upper=ndraws,progress) $
  "Gibbs Sampling"
do draw=-nburn,ndraws
  *
  * Draw sigma's given mu's and regimes
  *
  do i=1,ncats
    sstats(smpl=(s==i)) gstart gend (fish-mu(i))^2>>sumsqr
    compute sigma(i)=sqrt((sumsqr+nusumsqr)/%ranchisqr(%nobs+nudf))
  end do i
  *
  * Draw mu's given sigma's and regimes.
  *
  do i=1,ncats
    sstats(smpl=(s==i)) gstart gend fish>>sum
    compute mu(i)=sum/%nobs+%ran(sigma(i)/sqrt(%nobs))
  end do i
  *
  * Relabel if necessary
  *
  compute swaps=%index(mu)
  *
  * Relabel the mu's
  *
  compute temp=mu
  ewise mu(i)=temp(swaps(i))
  *
  * Relabel the sigma's
  *
  compute temp=sigma
  ewise sigma(i)=temp(swaps(i))
  *
  * Relabel the probabilities
  *
  compute temp=p
  ewise p(i)=temp(swaps(i))
  *
  * Draw the regimes, given p, the mu's and the sigma's

```

```

*
set s gstart gend = fxp=RegimeF(t).*p,%ranbranch(fxp)
*
* Draw the probabilities
*
do i=1,ncats
    sstats(smpl=(s==i)) gstart gend 1>>shapes(i)
end do i
compute p=%randirichlet(shapes+prior)
infobox(current=draw)
if draw>0 {
    *
    * Do the bookkeeping
    *
    do i=1,ncats
        compute mus(i)(draw)=mu(i)
        compute sigmas(i)(draw)=sigma(i)
    end do i
}
end do draw
infobox(action=remove)
*
density(smoothing=1.5) mus(1) 1 ndraws xf ff
scatter(footer="Density Estimate for mu(1)",style=line)
# xf ff
density(smoothing=1.5) mus(2) 1 ndraws xf ff
scatter(footer="Density Estimate for mu(2)",style=line)
# xf ff
density(smoothing=1.5) mus(3) 1 ndraws xf ff
scatter(footer="Density Estimate for mu(3)",style=line)
# xf ff
density(smoothing=1.5) mus(4) 1 ndraws xf ff
scatter(footer="Density Estimate for mu(4)",style=line)
# xf ff
*
scatter(footer="Mean-Sigma combinations") 4
# mus(1) sigmas(1) 1 ndraws
# mus(2) sigmas(2) 1 ndraws
# mus(3) sigmas(3) 1 ndraws
# mus(4) sigmas(4) 1 ndraws

```

Markov Switching: Introduction

With time series data, a model where the (unobservable) regimes are independent across time will generally be unrealistic for the process itself, and will typically be used only for modeling residuals. Instead, it makes more sense to model the regime as a Markov Chain. In a Markov Chain, the probability that the process is in a particular regime at time t depends only upon the probabilities of the regimes at time $t - 1$, and not on earlier periods as well. This isn't as restrictive as it seems, because it's possible to define a system of regimes at t which includes not just S_t , but the tuple $S_t, S_{t-1}, \dots, S_{t-k}$, so the “memory” can stretch back for k periods. This creates a feasible but more complicated chain, with M^{k+1} combinations in this augmented regime, where M is the number of possibilities for each S_t .

8.1 Markov Switching Models

The type of model generated by having separate regimes on observable data controlled by an unobservable Markov Chain are known variously as Markov Switching Models, Regime Switching Models or Hidden Markov Models. In econometrics, the term Markov Switching is by far the most common, with techniques typically described as MSxxx where xxx is the underlying model conditional on the regime, such as MSVAR for a Markov Switching VAR, or MS-GARCH for a Markov Switching GARCH.

Though it is rarely used in econometrics, the “Hidden Markov” terminology is actually more descriptive. The regime process *is* hidden. While you can do inference on its probability (using its effect on the observable y), you can't tell for certain which regime holds at a given point in time. If you *know* where the regimes are, this is not the type of model to use. Except in pathological cases, a Markov model switches both ways—if you can switch from regime 1 to regime 2, there should also be the probability of switching back. If you have a model where the switch only goes one way, this is not the type of model to use.

Note well that switching and Markov switching are *not* the same things (which makes the use of “Regime Switching” as a synonym for MS rather misleading). The structural breaks in Chapters 1, 2 and 3 are also switching models at a specific (possibly unknown) time and the TAR models from Chapters 4 and 5 are switching models under the control of an observable criterion. While there may be some uncertainty as to the precise location of the regime switches in

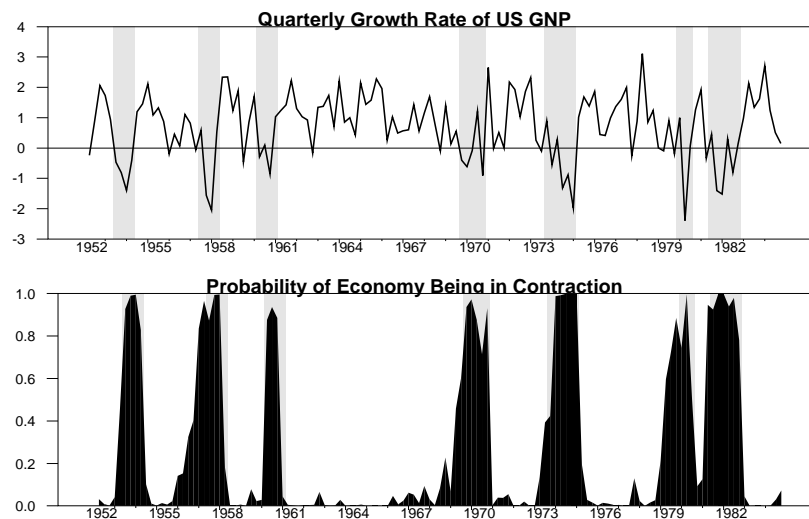


Figure 8.1: Regime Probabilities in Original Hamilton Model

those, the fact remains that given the model estimates, the process is known to be in a specific regime. In the Markov model, the regime cannot be known for certain, and often there can be considerable confusion about what regime holds at a given entry (more than one regime can have a non-trivial probability).

Markov switching models can be *notoriously* difficult to fit. Even if you are able to get a model to converge, they can be very difficult to interpret. Without getting into the specifics of how this type of model is estimated, consider the paper which introduced this type of model to the economics profession: Hamilton (1989). This posits a Markov switching model of GNP growth, with regimes labeled as low growth (“recession”) and high growth (“expansion”). With a relatively simple model, when estimated through 1984:4, this gives remarkably good results (Figure 8.1) as it comes very close to matching the NBER business cycle dates (background shading)—there’s a bit of uncertainty in the entrances to some of the recessions (though not really the exits) and that’s not unexpected as the NBER dates are hard 0-1’s when there was likely some real uncertainty in the committee about the precise entry dates in some of these.

It’s now been well-established that the good behavior of the simple Hamilton model doesn’t survive when applied to a longer range of U.S. data. Figure 8.2 shows the results from real GDP growth starting in 1952, but now ending in 2016:4. The model now completely misses several recessions and has phantom “positive” signals at odd locations, such as late 1977.

Just looking at the raw data (top panels) can give a clue as to what happened—after 1982, recessions have been less frequent and overall growth has been much steadier. In the original model, the mean in the low growth regime is -0.36 and in the high growth, it’s $+1.16$; with the updated data, those are now -1.16 and $+0.86$.¹ Thus, it’s now picking out much more extreme low growth peri-

¹These are quarter to quarter growth, not the more commonly quoted annualized growth

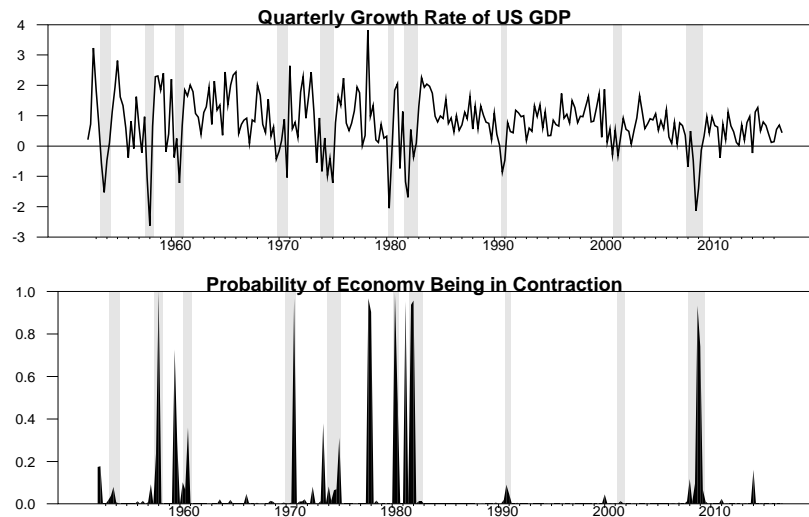


Figure 8.2: Regime Probabilities in Updated Hamilton Model

ods, which are much less persistent (and thus not behaving like business cycle recessions). The model could probably be rehabilitated with some adjustments after 1982 to reduce the variance and allow different transition probabilities, but in some ways that defeats the whole purpose of letting the model itself locate the “regimes” instead of imposing them through some form of dummies. This is an example of how Markov Switching models can produce results which may not be what you expect or desire. So before you start, make sure that it’s actually the correct model to answer the question you are asking, and be ready for disappointment if it doesn’t work out the way you want.

8.2 Common Concepts

As with the mixture models, the likelihood of the data (at t) given the regime is written as $f_{(i)}(y_t | X_t, \Theta)$. For now, we’ll just assume that this can be computed² and concentrate on the common calculations for all Markov Switching models which satisfy this assumption.

If the process is in regime j at $t-1$, the probability of moving to regime i at t can be written p_{ij} , where $\sum_{i=1}^M p_{ij} = 1$.³ This *transition probability matrix* can be time-invariant (the most common assumption), or can depend upon some exogenous variables. Typically, it is considered to be unknown, and its values must be estimated. Because of the adding-up constraint, there are only $M - 1$ free rates, so multiply by 4 to annualize them.

²There are important models for which the likelihood depends upon the entire history of regimes up through t , and thus can’t be written in this form.

³This can also be written as the transpose of this, so columns are the new regime and rows are the current regime. The formulas tend to look more natural with the convention that we’re using.

values in each column. In the RATS support routines for Markov Chains, the free parameters in this are represented by an $(M - 1) \times M$ matrix. Where there are just two regimes, it is often parameterized based upon the two probabilities of “staying” (p_{11} and p_{22} in our notation), but that doesn’t generalize well to more than two regimes, so we won’t use it in any of our examples.

Given the vector of (predicted) probabilities of being at regime i at t , the likelihood function is computed as it is with mixture models, and the steps in EM and in MCMC for estimating or drawing the regime-specific parameters for the models controlled by the switching are also the same as they are for the mixture models. What we need that we have not seen before are the rules of inference on the regime probabilities:

1. The *predicted* probabilities at t given data through $t - 1$. Calculating this is known as the *prediction step*.
2. The *filtered* probabilities at t given data through t . Calculating this is known as the *update step*.
3. The *smoothed* probabilities at t given data through the end of sample T .
4. Simulated regimes for $1, \dots, T$ given data through the end of sample T .

The support routines for this are in the file **MSSETUP.SRC**. You pull this in with:

```
@MSSETUP(regimes=# of regimes,lags=# of lags)
```

The **LAGS** option is used when the likelihood depends upon (a finite number of) lagged regimes. We’ll talk about that later, but for most models we’ll have (the default) of **LAGS=0**. Note that this is not necessarily the number of lags in the underlying model—dependence of the likelihood on lagged (observable) *data* generally doesn’t create a dependence on lagged *regimes*.

@MSSETUP defines quite a few standard variables which will be used in implementing these models. Among them are **NREGIMES** and **NLAGS** (number of regimes and number of lagged regimes needed), **P** and **THETA** (transition matrix and logistic indexes for it), **PT_T1**, **PT_T** and **PSMOOTH** (series of probabilities of regimes at various points). The only one that might raise a conflict with a variable in your own program would probably be **P**, so if you run across an error about a redefinition of **P**, you probably need to rename your original variable.

8.2.1 Prediction Step

Assuming that we have a vector of probabilities at $t - 1$ given data through $t - 1$ (which we’ll call $p_{t-1|t-1}$) and the transition matrix $\mathbf{P} = [p_{ij}]$, the predicted probabilities for t (which we’ll call $p_{t|t-1}$) are given by $p_{t|t-1} = \mathbf{P}p_{t-1|t-1}$. The function from **MSSETUP.SRC** which does this calculation is **%MCREGIME(P,PSTAR)**. In its argument list, **PSTAR** is the previous vector of probabilities and **P** can either

be the full $M \times M$ transition matrix or the $(M - 1) \times M$ parameterized version of it, whichever is more convenient in a given situation. It returns the VECTOR of predicted probabilities.

8.2.2 Update Step

This invokes Bayes rule and is identical to the calculation in the simpler mixture model, combining the predicted probabilities with the vector of likelihood values across the regimes:

$$p_{t|t}(i) = \frac{p_{t|t-1}(i)f(i)(y_t | X_t, \Theta)}{\sum_{i=1}^M p_{t|t-1}(i)f(i)(y_t | X_t, \Theta)} \quad (8.1)$$

This is computed with the function `%MSUPDATE(F,PSTAR,FPT)`. `F` is the VECTOR of likelihoods (at t) given the regime, `PSTAR` is (now) the vector of predicted probabilities. The function returns the VECTOR of updated probabilities, and also returns in `FPT` the likelihood of the observation, which is the denominator in (8.1). The product of the values of the `FPT` across t gives the likelihood (not logged) of the full sample using the standard sequential conditioning argument that

$$f(y_1, \dots, y_T) = f(y_1)f(y_2 | y_1) \cdots f(y_T | y_{T-1}, \dots, y_1)$$

The combination of prediction and update through the data set is known as (forward) filtering. Since most models will use exactly the same sequence of prediction and update steps, there is a separate function which combines all of this together. `%MSPROB(T,F)` takes as input the current time period (which will generally just be `T` since it's almost always used in a formula), and the vector of likelihoods `F`, does the prediction step and update steps and returns the likelihood.

8.2.3 Smoothing

This is covered by the result in Appendix A. In this application, we define

- x is the regime at t
- y is the regime at $t + 1$
- \mathcal{I} is the data through t
- \mathcal{J} is the data through T

The key assumption (A.1) holds because knowing the *actual* regime y at $t + 1$ provides better information about the regime at t than all the data from $t + 1, \dots, T$ that are added in moving from \mathcal{I} to \mathcal{J} . To implement this, we need to retain the series of predicted probabilities (which will give us $f(y|\mathcal{I})$, when we look at the predicted probabilities for $t + 1$) and filtered probabilities ($f(x|\mathcal{I})$).

At the end of the filtering step through the full data set, we have the filtered probabilities for S_T , which are (by definition) the same as the smoothed probabilities. (A.2) is then applied recursively, to generate the smoothed probability at $T - 1$, then $T - 2$, etc. For a Markov Chain, the $f(y|x, \mathcal{I})$ in (A.2) is just the probability transition matrix from t to $t + 1$. Substituting in the definitions for the Markov Chain, we get

$$p_{t|T}(j) = p_{t|t}(j) \sum_{i=1}^M p_{ij} \frac{p_{t+1|T}(i)}{p_{t+1|t}(i)}$$

This calculation needs to be “failsafed” against divide by zeros. The probability ratio

$$\frac{p_{t+1|T}(i)}{p_{t+1|t}(i)}$$

should be treated as zero if the denominator alone is zero. (The numerator should also be zero in that case).

In order to handle easily cases with more than two regimes, the various probability vectors are best handled by defining them as `SERIES[VECTOR]`. By convention, we call the inputs to the smoothing routines `PT_T` for the filtered probabilities, `PT_T1` for the predicted probabilities and `PSMOOTH` for the smoothed probabilities. The procedure on `MSSETUP.SRC` for calculating the smoothed probabilities is `@MSSMOOTHED`. Its syntax is:

```
@MSSMOOTHED start end PSMOOTH
```

This assumes that `PT_T` and `PT_T1` have been defined as described above, which is what the `%MSPROB` function will do. The output from `@MSSMOOTHED` is the `SERIES[VECT]` of smoothed probabilities.

To extract a series of a particular component from one of these `SERIES[VECT]`, use `SET` with something like

```
set p1 = psmooth(t) (1)
```

The result of this is the smoothed probability of regime 1.

8.2.4 Simulation of Regimes

The most efficient way to draw a sample of regimes is using the Forward Filter-Backwards Sampling (FFBS) algorithm described in Chib (1996). This is also known as Multi-Move Sampling since the algorithm samples the entire history at one time. The Forward Filter is exactly the same as used in the first step of smoothing. Backwards Sampling can also be done using the result in Appendix A. First, draw the regime at T from the filtered distribution. Then compute the distribution from which to sample $T - 1$ applying (A.2) with the $f(y|\mathcal{J})$ (which is,

in this case, $f(S_T|T)$ a unit vector at the sampled value for S_T . Walk backwards though the data range to get the full sampled distribution.

The procedure on `mssetup.src` that does the sampling is `@MSSAMPLE`. This has a form similar to the smoothing procedure. It's

```
@MSSAMPLE start end REGIME
```

The inputs are the same, while the output `REGIME` is a `SERIES[INTEGER]` which takes the sampled values between 1 and M . `@MSSETUP` includes a definition of a `SERIES[INTEGER]` called `MSRegime` which we use in most examples that require this.

Single-Move Sampling

To use Multi-Move Sampling, you need to be able to do the filtering (and smoothing) steps. This won't always be possible. The assumption made in (8.1) was that the likelihood at time t was a function only of the regime at t . The filtering and smoothing calculations can be extended to situations where the likelihood depends upon a *fixed* number of previous regimes by defining an augmented regime using the tuple $S_t, S_{t-1}, \dots, S_{t-k}$. However, a GARCH model has an unobservable "state" variable—the lagged variance—which depends upon the precise sequence of regimes that preceded it. The likelihood at t has M^t branches, which rapidly becomes too large to enumerate. Similarly, most state-space models have unobservable state variables which also depend upon the entire sequence of earlier regimes.⁴

An alternative form of sampling which can be used in such cases is *Single-Move Sampling*. This samples S_t taking $S_1, \dots, S_{t-1}, S_{t+1}, \dots, S_T$ as given. The joint likelihood of the full data set and the full set of regimes (conditional on all other parameters in the model) can be written as

$$f(\mathbf{Y}, \mathbf{S} \mid \Theta) = f(\mathbf{Y} \mid \mathbf{S}, \Theta) f(\mathbf{S} \mid \Theta) \quad (8.2)$$

Using the shorthand $\mathbf{S} - S_t$ for the sequence of regimes other than S_t , Bayes rule gives us

$$p(S_t = i \mid \mathbf{Y}, \mathbf{S} - S_t, \Theta) \propto f(\mathbf{Y} \mid \mathbf{S} - S_t, S_t = i, \Theta) f(\mathbf{S} - S_t, S_t = i \mid \Theta) \quad (8.3)$$

Using the Markov property on the chain, the second factor on the right can be written sequentially as:

$$f(\mathbf{S} \mid \Theta) = f(S_1 \mid \Theta) f(S_2 \mid S_1, \Theta) \dots f(S_T \mid S_{T-1}, \Theta)$$

In doing inference on S_t alone, any factor in this which doesn't include S_t will cancel in doing the proportions in (8.3), so we're left with:

⁴There are, however, state-space models for which states at t are known given the data through t , and such models can be handled using the regime filtering and smoothing.

- for $t = 1$, $f(S_1 | \Theta)f(S_2 | S_1, \Theta)$
- for $t = T$, $f(S_T | S_{T-1}, \Theta)$
- for others, $f(S_t | S_{t-1}, \Theta)f(S_{t+1} | S_t, \Theta)$

Other than $f(S_1 | \Theta)$ (which will be discussed on page 157), these will just be the various transition probabilities between the (assumed fixed) regimes at times other than t and the choice for S_t that we're evaluating.

At each t and for each value of i , we need to compute $f(Y | S - S_t, S_t = i, \Theta)$. For the types of models for which Single-Move Sampling is most commonly applied, evaluating the sample likelihood requires a complete pass through the data. Thus, to do just one sweep of Single-Move Sampling, we need to make $O(MT^2)$ calculations of likelihood elements. By contrast, Multi-Move Sampling requires $O(MT)$. With $T = 1000$ (not uncommon for a GARCH model) that means it will take roughly 1000 times longer to use Single-Move Sampling than it would Multi-Move Sampling on a similar type of model (such as an ARCH) for which the latter can be used. Plus, Multi-Move Sampling is much more efficient as a step in a Gibbs sampler because it samples all the regimes together. If you have a pair of data points (say t and $t + 1$) that are very likely to be in the same regime, but it's uncertain which one, Single-Move Sampling will tend to get stuck in one of the two since given S_t, S_{t+1} will generally be the same as S_t and then, given S_{t+1}, S_t will usually be the same as S_{t+1} . By contrast, Multi-Move Sampling will sample S_{t+1} in a nearly unconditional fashion, then sample S_t based upon that.

The following is an example of the process for Single-Move Sampling:

```
compute pstar=%mcergodic(p)
do time=gstart,gend
  compute lastregime=MSRegime(time)
  do i=1,nstates
    if MSRegime(time)==i
      compute logptest=logplast
    else {
      compute MSRegime(time)=i
      sstats gstart gend msgarchlogl>>logptest
    }

    compute pleft =%if(time==gstart,pstar(i), $
                                     p(i,MSRegime(time-1)))
    compute pright=%if(time==gend ,1.0 , $
                                     p(MSRegime(time+1),i))
    compute fps(i)=pleft*pright*exp(logptest-logplast)
    compute logp(i)=logptest
  end do i
  compute MSRegime(time)=%ranbranch(fps)
  compute logplast=logp(MSRegime(time))
end do time
```

This loops over `TIME` from the start to the end of the data range, drawing values for `MSREGIME(TIME)`. To reduce the calculation time, this doesn't compute the log likelihood function value at the *current* setting since that will be just be the value carried over from the previous time period—the variable `LOGPLAST` keeps the log likelihood at the current set of regimes. The only part of this that's specific to an application is the calculation of the log likelihood (into `LOGPTEST`) given a test set of values of `MSREGIME`, which is done here using the **SSTATS** instruction to sum the `MSGARCHLOGL` formula across the data set. Because the calculation needs (in the end) the likelihood itself (not the log likelihood), it's important to be careful about over- or underflows when *exp*'ing the sample log likelihoods. Since relative probabilities are all that matter, the `LOGPLAST` value is subtracted from all the `LOGPTEST` values before doing the *exp*.⁵

The `FPS` and `LOGP` vectors need to be set up before the loop with:

```
dec vect fps logp
dim fps(nregimes) logp(nregimes)
```

`FPS` keeps the relative probabilities of the test regimes, and `LOGP` keeps the log likelihoods for each so we don't have to recompute once we've chosen the regime.

Single-Move with Metropolis

Many of the rather time-consuming evaluations of the sample likelihood in Single-Move Sampling can be avoided by using Metropolis within Gibbs. This general technique is described in Appendix D. We're already avoiding doing a function evaluation for the current setting, but we still need to do the calculation for the other choice(s) at each time period. The relative probabilities of the regimes are a product of the (relative) likelihoods and the probabilities of moving to and from its neighbors for the regime being examined. If the chain is quite persistent, the "move" probabilities will often dominate the decision. For instance, if the probability of staying in 1 is .8 and of staying in 2 is .9, the probability of the sequence 1,1,1 is 32 times higher than 1,2,1 in a two-regime chain.⁶

Instead, we can use Metropolis sampling where we use the easy-to-compute transition probabilities as the proposal distribution. Draw from that. If we're

⁵If the log likelihoods are (for instance) -1000 and -1010, $\exp(-1000)$ and $\exp(-1010)$ are both true zeros in machine arithmetic, so we can't compute relative probabilities. By subtracting either -1000 or -1010 from each log likelihood before *exp*'ing, we get the proper roughly 22000:1 relative probabilities. Note that we haven't had to worry about this previously because the only likelihood needed in filtering is for just one data point at a time, not a whole sample.

⁶ $(.8 \times .8) / (.2 \times .1)$, where the denominator is the probability of moving from 1 to 2 times the probability of moving from 2 to 1.

looking at the combination above with $S_{t-1} = 1$ and $S_{t+1} = 1$, then roughly 32 times out of 33 we'll get a test value of $S_t = 1$ and in 1 out of 33, we'll get $S_t = 2$. We then only need to do a function evaluation if that test value is *different* from the current one; in a high percentage of cases, it will be the same, so we just stay where we are and move on to the next time period. If it *is* different, the Metropolis acceptance probability is just the ratio of the function value at the test settings to the function value at the current settings.⁷

The following is the general structure for this, with again, the only application-specific code being the evaluation of the log likelihood.

```
compute pstar=%mcergodic(p)
do time=gstart,gend
  compute oldregime=MSRegime(time)
  do i=1,nstates
    compute pleft=%if(time==gstart,pstar(i), $
                                     p(i,MSRegime(time-1)))
    compute pright=%if(time==gend, 1.0, $
                                     p(MSRegime(time+1), i))
    compute qp(i)=pleft*pright
  end do i

  compute candidate=%ranbranch(qp)
  if MSRegime(time)<>candidate {
    compute MSRegime(time)=candidate
    sstats gstart gend msgarchlog1>>logptest
    compute alpha=exp(logptest-logplast)
    if %ranflip(alpha)
      compute logplast=logptest
    else
      compute MSRegime(time)=oldregime
  }
end do time
```

This needs the following before the loop:

```
dec vect qp
dim qp(nregimes)
```

8.2.5 Pre-Sample Regime Probabilities

The first sentence in the description of the prediction step began “Assuming that we have a vector of probabilities at $t - 1$ given data through $t - 1$.” We have not addressed what happens when $t = 1$. There are two statistically justifiable

⁷Just the likelihoods, because the transition probabilities cancel, as they're the proposal distribution.

ways to handle the $p_{0|0}$. One is to treat them as free parameters, adding to the parameter set an M vector of non-negative values summing to one. This is by far the simplest way to handle the pre-sample if you use the EM algorithm, since it can compute this probability vector directly as part of the smoothing process. The other is to set them to the “ergodic” probabilities, which are the long-run average probabilities for the Markov Chain given the values for the transition matrix. For a time-invariant Markov Chain, this ergodic probability exists except in rare circumstances, such as an absorbing state (if, for instance, there’s a permanent break in the process). If you use ML, *this* is the most convenient way to handle the pre-sample.⁸ The two methods aren’t the same, and give rise to slightly different log likelihoods (the estimated probability, of necessity, giving the higher value).

8.2.6 Pathologies

Not all Markov Chains can work properly with standard treatments. Suppose we make the mistake of trying to fit a MS model to the data with the structural break in the mean shown in Figure 1.1. In a MS framework, this has two regimes, but there’s only one switch—once you move from the low mean regime to the high one, there is no (observed) switch back. That means that $p_{2,2}$ is 1. What about the transition from regime 1? You have 100 periods where 1 is followed by 1, and one period where 1 is followed by 2, so you would expect to see a $p_{1,1}$ of around 100/101. Whether that can be a reasonable description of the process will depend upon the situation—basically, that means that there’s a random exogenous process that determines the time of the transition. The Markov Chain here has what’s known as an *absorbing state* (regime 2)—if you compute the ergodic probabilities, it’s regime 2 with probability 1, as in the “long-run” eventually you will transition to 2 and then never leave. Clearly, you can’t use the ergodic probability to initialize the filter, since it doesn’t allow for regime 1. In fact, the “correct” initialization here would be regime 1 with probability 1, which would have to be forced into the model.

Another situation which, while not as serious a problem statistically, can cause somewhat odd behavior is when the the “Markov” part doesn’t really apply. The mixture model of Chapter 7 is a special case of a Markov Switching model where the transition probabilities are the same for each current regime, that is, the probability of the regime at t is independent of what regime holds at $t - 1$. You might end up with something close to this if there in fact is only one regime, or you could also see this if the data are dominated by a few scattered outliers.

⁸Maximum likelihood tends to be slow enough without having to deal with the extra parameters.

8.3 Estimation

As with mixture models, there are three basic methods of estimation: maximum likelihood, EM and MCMC. And as with mixture models, ML is usually the simplest to set up and EM is the quickest to execute. However, there are several types of underlying models where (exact) maximum likelihood isn't feasible and even more where EM isn't. MCMC is the only method which works in almost all cases.

For all types of models, you need to be able to compute a `VECTOR` of likelihood elements for each regime at each time period. It's the inability to construct this that makes ML and EM infeasible for certain types of models—while the switching mechanism may be a Markov Chain with short memory, the controlled process might depend upon the entire history of the regimes, which rapidly becomes too large to enumerate. MCMC avoids the problem because it always works with just one sample path at a time for the regimes rather than a weighted average across all paths. There are other models for which EM is not practical because the M step for the model parameters has no convenient form.

8.3.1 Simple Example

We'll look at a simple example to illustrate the three estimation methods. This has zero mean and Markov Switching variances. It is taken from Kim & Nelson (1999).⁹ The data are excess stock returns, monthly from 1926 to 1986. This is proposed as an alternative to ARCH or GARCH models as way to explain clustering of large changes—there is serial correlation in the variance regime through a Markov Switching process.

The full-sample mean is extracted from the data, so the working data are assumed to be mean zero. Thus, the only parameters are the variances in the branches and the parameters governing the switching process. The model is fit with three branches—we'll use the variable `NREGIMES` in all examples for the number of regimes. The variances will be in a `VECTOR` called `SIGMAS`. This will make it easy to change the number of regimes.

For all estimation methods, we need a **FUNCTION** which returns a `VECTOR` of likelihoods across regimes at a given time period. We'll use the following in this example, which can take any number of variance regimes:

⁹Their Application 3 in Section 4.6.


```

function RegimeF time
type vector          RegimeF
type integer         time
*
local integer        i
*
dim RegimeF(nregimes)
ewise RegimeF(i)=exp(%logdensity(sigmaz(i),ew_excs(time)))
end

```

8.3.2 Maximum Likelihood

With the **FUNCTION** returning the likelihoods written, an (almost) complete setup for maximum likelihood estimation with a model with time-invariant transition probabilities, including calculation of the smoothed probabilities is:

```

@MSSetup(regimes=3)
nonlin(parmset=modelparms) ....
nonlin(parmset=msparms) p
frml markov = f=RegimeF(t),fpt=%MSPProb(t,f),log(fpt)
@MSFilterInit
maximize(start=(pstar=%msinit()),$
  parmset=msparms+modelparms) markov start end
@MSSmoothed %regstart() %regend() psmooth

```

The **@MSFilterInit** procedure takes care of the specific setup that is needed for the forward filtering. The **%MSINIT** function takes care of the required initialization of a single filtering pass through the data (returning the pre-sample probabilities) and the **%MSPProb** function does the prediction and update steps returning the (non-logged) likelihood.

The above is the structure for direct estimation of the transition probabilities. We can also choose the logistic parameterization. With three (or more) choices, the logistic is generally the most reliable because the 1 to 3 and 3 to 1 probabilities can often be effectively zero, and we'll use this for the three-regime example in this section. Example 8.1 does maximum likelihood. We need to create the **THETA** matrix of logistic indices and give it guess values. It's generally easiest to first input the guess values in **P** form and use the **%MSPLOGISTIC** function to map it to the corresponding logistics.

```

input p
.6 .2 .1
.3 .6 .3
compute theta=%msplogistic(p)

```

We set up and initialize the variance vector with:

```

dec vect sigmas(nregimes)
stats ew_excs
compute sigmas(1)=0.2*%variance
compute sigmas(2)=1.0*%variance
compute sigmas(3)=5.0*%variance

```

We're trying to steer the estimates towards the labeling with 1 being the lowest variance and 3 the highest. There's no guarantee that we'll be successful, but this will probably work—you just don't want any of the guess values to be so extreme that the probabilities are effectively zero for all of them at all data points.

If you use the logistic parameterization, the **MAXIMIZE** instruction will have a slightly different **START** option, because we need to transform the logistic **THETA** into the probability matrix **P**. The **START** option will always have the **PSTAR=%MSINIT()** calculation; however, the transformation always needs to be done first. The **%(...)** enclosing the two parts of the **START** is needed because without it a “,” would be a separator for the instruction options.

```

maximize(start=%(p=%mslogisticp(theta),pstar=%msinit(p)), $
  parmset=msparms+modelparms, $
  method=bfgs, iters=400, pmethod=simplex, pters=5) markov * 1986:12

```

One thing to note is that the logistic mapping, while it makes estimation simpler when a transition probability is near the boundary, does not fix the problem at the boundary itself. In order to get a true zero probability, you need an index of $-\infty$ at the slot which needs to be zero, or of $+\infty$ on the others in the column if the zero needs to be at the bottom of the column. That's apparent in the output in Table 8.1, where the probability of moving from 1 to 3 is (effectively) zero—in order to represent that, the 1,1 and 1,2 elements need to be quite large.

8.3.3 EM

EM generally provides the quickest estimation, but requires some specialized calculations in all cases.

In the notation of Appendix B, let y represent the observed data $\{Y_t : t = 1, \dots, T\}$ and x the full record of the regimes $\{S_t : t = 0, \dots, T\}$, including the pre-sample regime. The E-step in the EM algorithm takes the form

$$E_x(\log f(x, y | \Theta) | y, \Theta_0)$$

Given the structure of x , this can be written

$$\sum_x (\log f(x, y | \Theta)) p(x | y, \Theta_0) \quad (8.4)$$

Table 8.1: Output from maximum likelihood estimation

MAXIMIZE - Estimation by BFGS					
Convergence in 65 Iterations. Final criterion was 0.0000013 <= 0.0000100					
Monthly Data From 1926:01 To 1986:12					
Usable Observations		732			
Function Value		1001.8949			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	THETA(1,1)	66.6724	12.9739	5.1390	0.0000
2.	THETA(2,1)	63.0229	13.0615	4.8251	0.0000
3.	THETA(1,2)	0.5093	0.8653	0.5886	0.5561
4.	THETA(2,2)	4.4164	0.6382	6.9198	0.0000
5.	THETA(1,3)	-58.0512	26.6287	-2.1800	0.0293
6.	THETA(2,3)	-2.9595	0.6797	-4.3542	0.0000
7.	SIGMAS(1)	0.0012	0.0002	6.8449	0.0000
8.	SIGMAS(2)	0.0040	0.0005	8.4234	0.0000
9.	SIGMAS(3)	0.0311	0.0057	5.4624	0.0000

where the sum is over all possible histories of the regimes. Since

$$f(x, y|\theta) = f(y|x, \theta)f(x|\theta)$$

we can also usefully decompose (8.4) as

$$\sum_x (\log f(y|x, \Theta)) p(x|y, \Theta_0) + \sum_x (\log f(x|\Theta)) p(x|y, \Theta_0) \quad (8.5)$$

The first term in this is relatively straightforward. Given x , $\log f(y|x, \Theta)$ is just the standard log likelihood for the model evaluated at that particular set of regimes. The overall term is the probability-weighted average of the log likelihood, where the weights are the probabilities of the state histories evaluated at Θ_0 . In most cases, this will end up requiring a probability-weighted regression of some form. Because the weighting probabilities are based upon Θ_0 rather than the Θ over which the M step optimizes, this term can be maximized separately from the second one—the first depends upon the regression parameters in Θ , but not the transition parameters, while the second is the reverse.

Where there are no parameters shared across regimes, this part of the M step can generally be done by looping over the regimes, using a standard estimation instruction with the `WEIGHT` option, where the `WEIGHT` is the series of smoothed probabilities for that regime. If there *are* shared parameters (for instance, in a regression, a common variance), the calculation is quite a bit more complicated.

In the second term in (8.5), using the standard trick of sequential conditioning gives

$$f(x|\Theta) = f(S_0 | \theta) f(S_1|S_0, \theta) \dots f(S_T|S_{T-1}, S_{T-2}, \dots, S_0, \Theta) \quad (8.6)$$

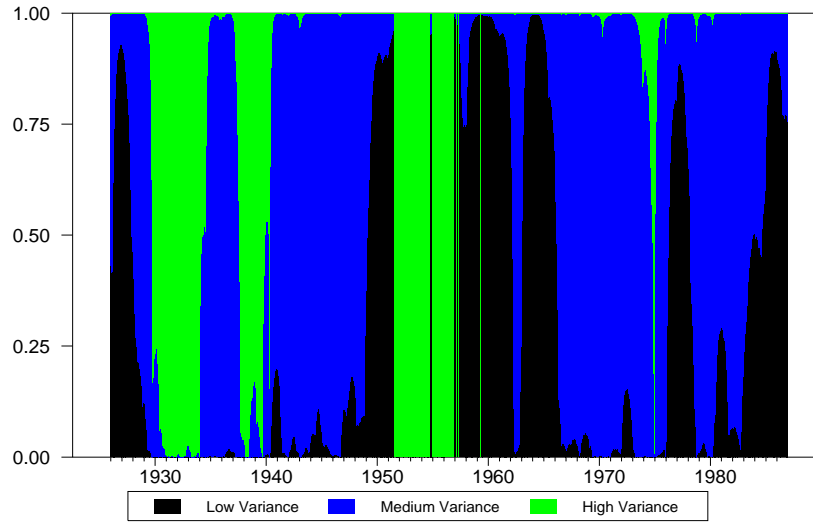


Figure 8.3: Smoothed Probabilities of Variance Regimes

By the Markov property, the conditional densities can be reduced to conditioning on just one lag. The second term can thus be rewritten as

$$(\log f(S_0 | \Theta)) p(S_0 | y, \Theta_0) + \sum_{t=1}^T (\log f(S_t | S_{t-1}, \Theta)) p(S_t, S_{t-1} | y, \Theta_0) \quad (8.7)$$

The first term in (8.7) is quite inconvenient, and is generally ignored, with the assumption that it will be negligible as a single term compared to the T element sum that follows. That does, however, mean that EM won't converge (exactly) to maximum likelihood, but will only be close.

The probability weights in the sum, $p(S_t, S_{t-1} | y, \Theta_0)$, are *smoothed* probabilities of the *pair* (S_t, S_{t-1}) computed at Θ_0 . They have to be the smoothed estimates because they are conditioned on the full y record. This requires a specialized filtering and smoothing calculation operating on pairs of regimes, but it's the same calculation for all underlying model types. We'll use the abbreviation:

$$\hat{p}_{ij,t} = P(S_t = i, S_{t-1} = j | T)$$

Given those, the maximizer for the sum in (8.7) for a fixed transition matrix is

$$\hat{p}_{ij} = \frac{\sum_{t=1}^T \hat{p}_{ij,t}}{\sum_{k=1}^M \sum_{t=1}^T \hat{p}_{ik,t}}$$

in effect, the empirical estimate of the transition probabilities from the smoothed estimates. This completes the M step, so we return to the E step and continue until convergence or we complete the desired number of passes.

The procedures and functions for handling most of the EM for the Markov Switching model are in the file `msemsetupstd.src`. These will be pulled in when you do

```
@MSEMSetupStd(regimes=# of regimes)
```

This includes all the functions from `mssetup.src`, plus additional ones for the EM calculations in Markov Switching models. These will do almost everything except the calculation of your model-specific likelihood functions, and the M step for the model-specific parameters. In Example 8.2, we do 50 iterations of the EM algorithm. Each iteration starts with:

```
@MSEMFiterInit
do time=gstart,gend
    @MSEMFiterStep time RegimeF(time)
end do time
disp "Iteration" ### emits * "Log Likelihood" %logl
```

This executes the filter step on the pairs of current and lagged regimes.¹⁰ As a side-effect, this combination computes the log likelihood into `%LOGL`. This will increase from one iteration to the next—usually quickly at first, then more slowly.

The above generates predicted and filtered versions of the probabilities of the regime pairs. The next part computes the smoothed probabilities of the regime pairs and their marginal to the current regime:

```
@MSEMSmooth
gset psmooth gstart gend = MSEMmarginal(MSEMpt_sm(t))
```

These two steps are basically the same for any model, other than the calculation of the `REGIMEF` function. The next part is where the model will matter. In the case of the switching variances, the M step for the variances does probability-weighted averages of the squares of the data:

```
do i=1,nregimes
    sstats gstart gend psmooth(t)(i)*ew_excs^2>>wsumsq $
        psmooth(t)(i)>>wts
    compute sigmas(i)=wsumsq/wts
end do i
```

The final procedure from the support routines does the M step for the transitions:

```
@MSEMDoPMatrix
```

¹⁰This is needed for the inference on the transitions, and there's no advantage to doing a separate filter/smooth operation just on the current regime, since those probabilities are just marginals of the regime pair calculation.

8.3.4 MCMC (Gibbs Sampling)

This treats the regimes as a separate set of parameters. There are three basic steps in this:

1. Draw the regimes given the model parameters and the transition parameters.
2. Draw the model parameters given the regimes (transition parameters generally don't enter).
3. Draw the transition parameters given the regimes (model parameters generally don't enter).

The first step was already discussed above (Section 8.2.4). In practice, we may need to reject any draws which produce too few entries in one of the regimes to allow safe estimation of the model parameters for that regime. The second step will require a standard set of techniques; it's just that they will be applied to subsamples determined by the draws for the regimes. The only thing (slightly) new here will be the third step. This will be similar to the analogous step in the mixture model except that the analysis has to be carried out separately for each value of the "source" regime. In other words, for each j , we look at the probabilities of moving from j at $t - 1$ to i at t . We count the number of cases in each "bin", combine it with a weak Dirichlet prior and draw column j in the transition matrix from a Dirichlet distribution. Each column will generally need its own settings for the prior, as standard practice is for a weak prior but one which favors the process staying in its current regime. In our examples, we will represent the prior as a `VECT[VECT]` with a separate vector for each regime. For instance, in Example 8.3, we use

```
dec vect[vect] gprior(nregimes)
compute gprior(1)=||8.0,1.0,1.0||
compute gprior(2)=||1.0,8.0,1.0||
compute gprior(3)=||1.0,1.0,8.0||
```

which has a mean of the probability of staying in the current regime as .8. The obvious initial values for the `P` matrix are the means:

```
ewise p(i,j)=gprior(j)(i)/%sum(gprior(j))
```

The draw for the `P` matrix can be done by using the procedure `@MSDrawP`, which takes the input `GPRIOR` in the form we've just described, along with the sampled values of `MSRegime`, and produces a draw for `P`. This same line will appear in the MCMC estimation in each of the chapters on Markov switching:

```
@MSDrawP(prior=gprior) gstart gend p
```

In addition, MCMC has the issue of label switching, which isn't shared by ML and EM, both of which simply pick a particular set of labels. For models with

an obvious ordering, we can proceed as described in Section 7.3.1. However, it's not always immediately clear how to order a model. In an AR model, you might, for instance, need to compute the process means as a function of the coefficients and order on that.

A final thing to note is that Markov Switching models can often have multiple modes aside from simple label switches—you could have modes with switches between high and low mean, between high and low variances, between normal and outlier data points. ML and EM will generally find one (though not always the one with the highest likelihood). MCMC may end up “visiting” several of these, so simple sample statistics like the mean of a parameter across draws might not be a good description. It's not a bad idea to also include estimated densities.

Kim and Nelson estimate this model using Gibbs Sampling as their Application 1 in Section 9.2. We will do several things differently than is described there. Most of the changes are designed to make the algorithm easier to modify for a different number of regimes. First, K&N draw the transition probabilities by using a sequence of draws from the beta, first drawing the probability of staying vs moving, then dividing the probability of moving between the two remaining regimes. This is similar to, but not the same as, drawing the full column at one time using the Dirichlet (which is what we'll do), and is *much* more complicated. Second, they draw the variances sequentially, starting with the smallest, working towards the largest, enforcing the requirement that the variances stay in increasing order by rejecting draws which put the variances out of order. This will work adequately¹¹ as long as all the regimes are all well-populated. However, if the regime at the end (in particular) has a fairly small number of members, the draws for its variance in their scheme can be quite erratic.

Instead of their sequential method, we'll use a (symmetrical) hierarchical prior as described in Appendix C and switch labels to put the variances in the desired order. The draws for the hierarchical prior take the following form: this draws the common variance taking the ratios as given¹² then the regime variances given `SCOMMON`.

¹¹Though it becomes more complicated as the number of regimes increases.

¹²This never actually computes the ratios, but instead uses the implied value of `SIGMAS(i) / SCOMMON`.

```

sstats gstart gend ew_excs^2*scommon/sigmas(MSRegime(t))>>sumsq
compute scommon=(sumsq+nucommon*s2common)/$
                    %ranchisqr(%nobs+nucommon)
do i=1,nregimes
    sstats(smpl=MSRegime(t)==i) gstart gend $
        ew_excs^2/scommon>>sumsq
    compute sigmas(i)=scommon*(sumsq+nuprior(i))/$
        %ranchisqr(%nobs+nuprior(i))
end do i

```

The second stage in this requires the degrees of freedom for the prior for each component (the `NUPRIOR` vector), which is 4 for all regimes in our example. The first stage allows for informative priors on the common variance (using `NUCOMMON` and `S2COMMON`), but we're using the non-informative zero value for `NUCOMMON`.

With the order in which the parameters are drawn in this example (regimes, then variances, then transition probabilities), the label switching needs to correct the sigmas and the regimes, but doesn't have to fix the transitions since they get computed afterwards:

```

compute swaps=%index(sigmas)
*
* Relabel the sigmas
*
compute temp=sigmas
ewise sigmas(i)=temp(swaps(i))
*
* Relabel the regimes
*
gset MSRegime gstart gend = swaps(MSRegime(t))

```

The sections of the MCMC loop which draw the regimes and draw the transition matrices are effectively the same for all models.

Example 8.1 Markov Switching Variances-ML

Estimation of a model with Markov Switching variances by maximum likelihood. This is described in detail in Section 8.3.2.

```

open data ew_excxs.prn
calendar(m) 1926:1
data(format=free,org=columns) 1926:01 1986:12 ew_excxs
*
* Extract the mean from the returns.
*
diff(center) ew_excxs
*
@MSSetup(regimes=3)
*
* Provide guess values for p, then translate into equivalent thetas.
*
input p
.6 .2 .1
.3 .6 .3
compute theta=%msplogistic(p)
*
dec vect sigmas(nregimes)
stats ew_excxs
compute sigmas(1)=0.2*%variance
compute sigmas(2)=1.0*%variance
compute sigmas(3)=5.0*%variance
*
*****
*
* RegimeF returns a vector of likelihoods for the various regimes at
* <<time>>. The likelihoods differ in the regimes based upon the
* values of sigmas.
*
function RegimeF time
type vector RegimeF
type integer time
*
local integer i
*
dim RegimeF(nregimes)
ewise RegimeF(i)=exp(%logdensity(sigmas(i),ew_excxs(time)))
end
*****
nonlin(parmset=modelparms) sigmas
nonlin(parmset=msparms) theta
*
frml markov = f=RegimeF(t),fpt=%MSProb(t,f),log(fpt)
*
@MSFilterInit
maximize(start=(p=%mslogisticp(theta),pstar=%MSInit()),$
parmset=msparms+modelparms,$
method=bfgs, iters=400, pmethod=simplex, pitters=5) markov

```

```

*
@MSSmoothed %regstart() %regend() psmooth
set p1 = psmooth(t) (1)
set p2 = psmooth(t) (2)
set p3 = psmooth(t) (3)
graph(style=stacked,maximum=1.0,picture="##.##", $
    header="Smoothed Probabilities of Variance Regimes",key=below,$
    klables=||"Low Variance","Medium Variance","High Variance"||) 3
# p1
# p2
# p3
*
set variance = p1*sigmas(1)+p2*sigmas(2)+p3*sigmas(3)
graph(footer=$
    "Figure 4.10 Estimated variance of historical stock returns")
# variance
*
set stdu = ew_excsc/sqrt(variance)
graph(footer="Figure 4.11a Plot of standardized stock returns")
# stdu

```

Example 8.2 Markov Switching Variances-EM

Estimation of a model with Markov Switching variances by EM. See Section 8.3.3 for more information.

```

open data ew_excsc.prn
calendar(m) 1926:1
data(format=free,org=columns) 1926:01 1986:12 ew_excsc
*
* Extract the mean from the returns.
*
diff(center) ew_excsc
*
@MSEMSSetupStd(regimes=3)
*
* Set up the parameters, and provide guess values. We'll use the
* full size transition matrix.
*
dec rect p(nregimes,nregimes)
ewise p(i,j)=%if(i==j,.8,.2/(nregimes-1))
*
dec vect sigmas(nregimes)
stats ew_excsc
compute sigmas(1)=0.2*%variance
compute sigmas(2)=1.0*%variance
compute sigmas(3)=5.0*%variance
*
*****
* RegimeF returns a vector of likelihoods for the various regimes
* at <<time>>. The likelihoods differ in the regimes based upon the
* values of sigmas.

```

```

*
function RegimeF time
type vector    RegimeF
type integer   time
*
local integer  i
*
dim RegimeF(nregimes)
ewise RegimeF(i)=exp(%logdensity(sigmaz(i),ew_excstime))
end
*****
compute gstart=1926:1,gend=1986:12
do emits=1,50
  *
  * Do the "E" step, computing smoothed probabilities for the pairs
  * of current and lagged regimes given the variances and the
  * transition probabilities.
  *
  * Filter through the data, saving the predicted and filtered
  * probabilities.
  *
  @MSEMFiterInit gstart gend
  do time=gstart,gend
    @MSEMFiterStep time RegimeF(time)
  end do time
  *
  disp "Iteration" ### emits "Log Likelihood" * %logl
  @MSEMSmooth
  *
  * Get the psmoothed current probabilities by marginalizing the
  * combined regime probabilities.
  *
  gset psmooth gstart gend = MSEMMarginal(MSEMFpt_sm(t))
  *****
  *
  * M-step for the variances. In this case, these will be
  * probability-weighted averages of the sums of squares.
  *
  do i=1,nregimes
    sstats gstart gend psmooth(t)(i)*ew_excstime^2>>wsumsq $
      psmooth(t)(i)>>wts
    compute sigmaz(i)=wsumsq/wts
  end do i
  *
  * M-step for the transitions.
  *
  @MSEMDoPMatrix
end do emits
*
disp "P=" p
disp "Sigma=" sigmaz
*
set p1 = psmooth(t)(1)
set p2 = psmooth(t)(2)

```

```

set p3 = psmooth(t) (3)
graph(style=stacked,maximum=1.0,picture="##.##", $
  header="Smoothed Probabilities of Variance Regimes",key=below,$
  klabels=||"Low Variance","Medium Variance","High Variance"||) 3
# p1
# p2
# p3
*
set variance = p1*sigmas(1)+p2*sigmas(2)+p3*sigmas(3)
graph(footer=$
  "Figure 4.10 Estimated variance of historical stock returns")
# variance
*
set stdu = ew_excs/sqrt(variance)
graph(footer="Figure 4.11a Plot of standardized stock returns")
# stdu

```

Example 8.3 Markov Switching Variances-MCMC

Estimation of a model with Markov Switching variances by Markov Chain Monte Carlo. The details are in Section 8.3.4.

```

open data ew_excs.prn
calendar(m) 1926:1
data(format=free,org=columns) 1926:01 1986:12 ew_excs
*
* Extract the mean from the returns.
*
diff(center) ew_excs
*
linreg ew_excs
# constant
*
compute gstart=%regstart(),gend=%regend()
@MSSSetup(regimes=3)
*
dec vect sigmas(nregimes)
*
* This uses the full n x n matrix for the probabilities.
*
dec rect p(nregimes,nregimes)
*
* Prior for transitions.
*
dec vect[vect] gprior(nregimes)
dec vect tcounts(nregimes) pdraw(nregimes)
compute gprior(1)=||8.0,1.0,1.0||
compute gprior(2)=||1.0,8.0,1.0||
compute gprior(3)=||1.0,1.0,8.0||
*
* Initial values for p.

```

```

*
ewise p(i,j)=gprior(j) (i)/%sum(gprior(j))
*
* Initialize the sigmas
*
stats ew_excs
compute scommon =%variance
compute sigmas(1)=0.2*scommon
compute sigmas(2)=1.0*scommon
compute sigmas(3)=5.0*scommon
*
* Hierarchical prior for sigmas
* Uninformative prior on the common component.
*
compute nucommon=0.0
compute s2common=0.0
*
* Priors for the relative variances
*
dec vect nuprior(nregimes)
dec vect s2prior(nregimes)
compute nuprior=%fill(nregimes,1,4.0)
compute s2prior=%fill(nregimes,1,1.0)
*****
*
* RegimeF returns a vector of likelihoods for the various regimes
* at <<time>>. The likelihoods differ in the regimes based upon the
* values of sigmas.
*
function RegimeF time
type vector    RegimeF
type integer   time
*
local integer  i
*
dim RegimeF(nregimes)
ewise RegimeF(i)=exp(%logdensity(sigmas(i),ew_excs(time)))
end
*****
@MSFilterSetup
*
* Initialize the regimes
*
gset MSRegime gstart gend = 1
*
* This is a smaller number of draws than would be desired for a
* final result.
*
compute nburn=2000,ndraws=10000
*
* For convenience in saving the draws for the parameters, create a
* non-linear PARMSET with the parameters we want to save. We can use
* the %PARMSPEEK function to get a VECTOR with the values for saving
* in the BGIBBS SERIES of VECTORS.

```

```

*
nonlin(parmset=mcmcparms) p sigmas
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %parmspeek(mcmcparms)
*
* Keeps track of the sum of the dummies for the regimes, for
* computing the probabilities.
*
dec vect[series] sregimes(nregimes)
clear(zeros) sregimes
*
* Keeps track of the sum of the current estimated variance for each
* time period, for computing the average variance.
*
set estvariance = 0.0
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
"Gibbs Sampling"
do draw=-nburn,ndraws
  *
  * Draw S given means and variances using forward filter-backwards
  * sampling algorithm.
  *
  * Forward filter
  *
  @MSFilterInit
  do time=gstart,gend
    @MSFilterStep time RegimeF(time)
  end do time
  *
  * Backwards sample
  *
  @MSSample gstart gend MSRegime
  *****
  *
  * Draw the common variance factor given the relative variances
  * and regimes.
  *
  sstats gstart gend ew_excs^2*scommon/sigmas(MSRegime(t))>>sumsq
  compute scommon=(sumsq+nucommon*s2common)/$
    %ranchisqr(%nobs+nucommon)
  *
  * Draw the relative variances, given the common variances and the
  * regimes.
  *
  do i=1,nregimes
    sstats(smpl=MSRegime(t)==i) gstart gend $
      ew_excs^2/scommon>>sumsq
    compute sigmas(i)=scommon*(sumsq+nuprior(i))/$
      %ranchisqr(%nobs+nuprior(i))
  end do i
  *
  * Handle label switching
  *

```

```

compute swaps=%index(sigmas)
*
* Relabel the sigmas
*
compute temp=sigmas
ewise sigmas(i)=temp(swaps(i))
*
* Relabel the regimes
*
gset MSRegime gstart gend = swaps(MSRegime(t))
*****
*
* Draw p's
*
@MSDrawP(prior=gprior) gstart gend p
infobox(current=draw)
*
* Once we're past the burn-in, save results.
*
if draw>0 {
  *****
  *
  * Combine p and the sigma vector into a single vector and save
  * it for each draw.
  *
  compute bgibbs(draw)=%parmspeek(mcmcparms)
  *
  * Update the sum of the occurrence of each regime, and the sum
  * of the variance of each entry.
  *
  do i=1,nregimes
    set sregimes(i) = sregimes(i)+(MSRegime(t)==i)
  end do i
  set estvariance = estvariance+sigmas(MSRegime(t))
  *****
}
end do draw
infobox(action=remove)
*
* Compute means and standard deviations from the Gibbs samples
*
@mcmcpostproc(ndraws=ndraws,mean=bmeans,stderrs=bstderrs) bgibbs
*
* Put together a report similar to table 9.2. Note that we're
* including the 3rd row in the saved statistics, so we have to skip
* over a few slots to get just the ones from the table.
*
report(action=define)
report(atrow=1,atcol=2,tocol=3,span) "Posterior"
report(atrow=2,atcol=2) "Mean" "SD"
report(atrow=3,atcol=1) "$p_{11}$"          bmeans(1) bstderrs(1)
report(atrow=4,atcol=1) "$p_{12}$"          bmeans(2) bstderrs(2)
report(atrow=5,atcol=1) "$p_{21}$"          bmeans(4) bstderrs(4)
report(atrow=6,atcol=1) "$p_{22}$"          bmeans(5) bstderrs(5)

```

```

report(atrow=7,atcol=1) "$p_{31}$"          bmeans(7) bstderrs(7)
report(atrow=8,atcol=1) "$p_{32}$"          bmeans(8) bstderrs(8)
report(atrow=9,atcol=1) "$\sigma_1^2$"      bmeans(10) bstderrs(10)
report(atrow=10,atcol=1) "$\sigma_2^2$"     bmeans(11) bstderrs(11)
report(atrow=11,atcol=1) "$\sigma_3^2$"     bmeans(12) bstderrs(12)
report(action=format,picture="###.####")
report(action=show)
*
* Convert sums to averages and graph
*
do i=1,nregimes
    set sregimes(i) = sregimes(i)/ndraws
end do i
set estvariance = estvariance/ndraws
*
graph(footer="Figure 9.2a Probability of low-variance regime")
# sregimes(1)
graph(footer="Figure 9.2b Probability of medium-variance regime")
# sregimes(2)
graph(footer="Figure 9.2c Probability of high-variance regime")
# sregimes(3)
graph(footer="Figure 9.3 Estimated variance of historical returns")
# estvariance
*
garch(nomean,hseries=h) / ew_excs
graph(footer="Comparison of GARCH and MS Variance Models") 2
# h
# estvariance

```


Markov Switching Regressions

In the example in Chapter 8, each regime had only one free parameter: its variance. This makes it very easy to control the interpretation of the regimes. It's still certainly possible to have more than one non-trivial local mode for the likelihood, but each mode will place the regimes in an identifiable order.

Things are considerably more complicated with the subject of this chapter, which is a Markov switching linear regression. Even in the simplest case where the only explanatory variable is a constant, if the variance also switches with regime, there are two parameters governing each regime. Unless either the mean or the variance is rather similar across regimes, you won't know for certain whether the regimes represent mainly a difference in variance, a difference in means, or some combination of the two. In fact, in the example that we are using in this chapter, the interpretation desired by the author (that the regimes separate based largely upon the mean level of the process) doesn't seem to be the one supported by the data, and instead, it's the variance.

This difficulty arises because the regime can only be inferred from the estimates, unlike a TAR or STAR model, where a computable condition determines which regime holds. It's not difficult to *estimate* a fully-switching linear regression—it just may be hard to make sense of the results.

9.1 MSREGRESSION procedures

The support routines for Markov switching linear regressions are in the file `MSREGRESSION.SRC`. You'll generally pull this in by executing the `@MSRegression` procedure:

```
@MSRegression( options ) depvar
# list of regressors
```

The `options` are `REGIMES=#` of regimes, with a default of 2, and `SWITCH=[C]/CH/H`, which determines what switches—`SWITCH=C` (the default) is coefficients only, `SWITCH=CH` means coefficients and variances and `SWITCH=H` is variances only. You can also have some coefficients switching and some fixed—to do this, use the option `NFIX=#` of fixed coefficients together with `SWITCH=C` or `SWITCH=CH`, and arrange the regressor list to have the fixed coefficients first. There's also an `EQUATION` option to input the form of

the regression—if you use that, you can leave out the *list of regressors* supplementary card.

The `@MSRegression` pulls in all the required functions and procedures for ML and EM, including the `FUNCTION` which returns the likelihoods across regimes: this is `%MSRegFVec(time)`. The switching coefficients for the regression are in a `VECT[VECT]` called `BETAS` where `BETAS(I)` is the coefficient vector under regime `I`. The fixed coefficients are in a `VECTOR` called `GAMMAS`. The variances are either in the `REAL` variable `SIGSQ` if the variances don't switch or in the `VECTOR` `SIGSQV` if they do.

9.2 The example

The example that we'll use is based upon the application from section 4.5 in Tsay (2010).¹ This is a two-lag autoregression on the change in the unemployment rate, estimated using two regimes. It is basically the same underlying model (with a slightly different data set) used in Example 4.1 as a TAR. Unlike the “Hamilton” model described in Chapter 22 of Hamilton (1994) and in Chapter 11 of this course, this allows the intercept, lag coefficients and variance all to switch, while Hamilton heavily constrains the autoregression to allow only the mean to switch. The Hamilton approach is much more complicated to implement, but makes for easier interpretation by allowing only the one switching parameter per regime.

The ML and EM programs will be quite similar and relatively simple for all the models covered by this procedure, once you've read in the data and executed the procedure. The Bayesian estimates will naturally be somewhat different from application to application because of the need for a model-specific prior.

For this example, the setup code, which will be used in all three estimation methods is:

```
@MSRegression(switch=ch, regimes=2) dx
# constant dx{1 2}
```

9.2.1 Maximum Likelihood

As in Chapter 8, you must choose in which form to estimate the transition probabilities. With just two regimes, we'll use the simpler direct estimate of the `P` matrix. The parameter sets for the regression part and the Markov switching part are:

```
@MSRegParmset(parmset=regparms)
nonlin(parmset=msparms) p
```

¹This is also in the 2nd edition of that book.

The `@MSRegParmset` pulls together all the parameters/parameter vectors needed for the model as you've set it up—here it will be the `BETAS VECTOR` of coefficient `VECTORS` and the `SIGSQV VECTOR` of variances.

Before we can estimate, we need to do something to distinguish the regimes in the guess values. The procedure `@MSRegInitial` does this by estimating a single linear regression, and altering the intercept in the regimes (from low to high as the regime number increases). If you expect the switch to be (for instance) more on a slope coefficient, you would have to use some other method to set the initial guess values for the `BETAS` vectors. For instance, `@MSRegInitial` has a `GUESSREGIMES` option which can be used if you have a fairly strong prior on how the regimes might split and can create a dummy variable which approximates that—we do that in Section 9.2.3.

```
@MSRegInitial
compute gstart=%regstart(),gend=%regend()
```

Since we're allowing free estimation of both the regression coefficients and the variances, we need to avoid the problem with zero variance spikes (page 132). To that end, we compute a lower limit on the variance that we can use for the `REJECT` option on the `MAXIMIZE`. Here, we make that a small multiple of the smaller guess value for the variance. Since the problem comes only when the variance is (effectively) zero, any small, but finite, bound should work without affecting the ability to locate the interior solution. The function `%MSRegInitVariances` is needed at the start of each function evaluation to make sure the fixed vs switching variances are handled properly; as a side effect, it returns the minimum of the variances.

```
compute sigmalimit=1.e-6*%minvalue(sigsqv)
*
frml logl = f=%MSRegFVec(t),fpt=%MSProb(t,f),log(fpt)
@MSFilterInit
maximize(start=(sigmatest=%MSRegInitVariances(),pstar=%msinit()),$
  parmset=regparms+mparams,$
  reject=sigmatest<sigmalimit,$
  method=bfgs,pmethod=simplex,piters=5) logl gstart gend
```

As you can see, this is very similar to the estimation code in the previous chapter. The results are in Table 9.1. The two regimes have very similar (almost identical) persistence—both around .92, so the expected time to a switch is $1/(1-.92)$ or roughly 12 quarters.

In all three examples, we'll include a graph of the estimated (smoothed) probabilities of regime 1. Each estimation technique will use a different method to obtain this. With ML, we need to do the extra step of smoothing the filtered estimates, since those aren't needed in the course of the estimation itself.

Table 9.1: Output from MS Linear Regression

MAXIMIZE - Estimation by BFGS					
Convergence in 14 Iterations. Final criterion was 0.0000057 <= 0.0000100					
Quarterly Data From 1948:04 To 1993:04					
Usable Observations		181			
Function Value		-23.5124			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	BETAS(1)(1)	-0.0555	0.0197	-2.8192	0.0048
2.	BETAS(1)(2)	0.3447	0.1094	3.1511	0.0016
3.	BETAS(1)(3)	0.0137	0.1017	0.1350	0.8926
4.	BETAS(2)(1)	0.0560	0.0462	1.2106	0.2261
5.	BETAS(2)(2)	0.8541	0.1072	7.9706	0.0000
6.	BETAS(2)(3)	-0.3294	0.1087	-3.0297	0.0024
7.	SIGSQV(1)	0.0199	0.0039	5.0861	0.0000
8.	SIGSQV(2)	0.1737	0.0287	6.0429	0.0000
9.	P(1,1)	0.9206	0.0355	25.9680	0.0000
10.	P(1,2)	0.0801	0.0388	2.0618	0.0392

```
@MSSmoothed gstart gend psmooth
set plsmooth = psmooth(t) (1)
graph(footer="Smoothed Probabilities of Regime 1",max=1.0,min=0.0)
# plsmooth
```

Finally, this computes the conditional means across regimes by looking at the implied process means of the AR(2) models. Note that this is a calculation which is specific to this model (an AR(2) with intercept).

```
do i=1,2
  disp "Conditional Mean for Regime" i $
    betas(i) (1) / (1-betas(i) (2)-betas(i) (3))
end do i
```

Conditional Mean for Regime 1	-0.08647
Conditional Mean for Regime 2	0.11780

The desire of the author was to have the regimes switch based on the mean. It's certainly the case that the mean is lower in regime 1 than in 2. Note, however, that in the regression results, the *variance* is also substantially higher in regime 2 than in 1, and in the graph of the probabilities (Figure 9.1), regime 2 largely coincides with recessions after 1962, but not before then. So there are some real signs that this might not be producing the desired results.

The full program is Example 9.1.

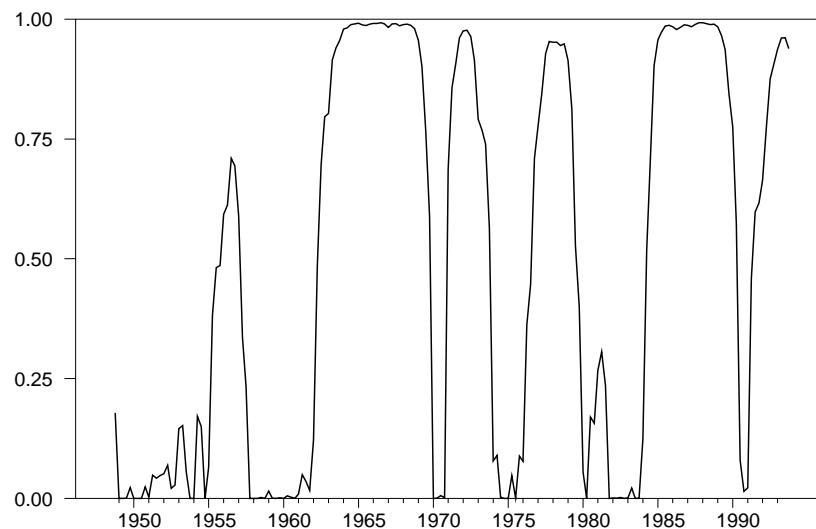


Figure 9.1: Smoothed Probability of Regime 1

9.2.2 EM

Most of the specialized calculations required for EM estimation are included in a set of procedures within the **MSREGRESSION.SRC** file. As a result, after the model has been set up, all that is required is the following:

```
@MSRegEMGeneralSetup
do emits=1, 50
    @MSRegEMStep gstart gend
    disp "Iteration" emits "Log likelihood" %logl
end do emits
```

which does 50 EM iterations. In the example, we follow this by doing 10 BHHH iterations to polish the estimates and get the covariance matrix.² Because EM uses a full-size **P** matrix and ML uses one without the bottom row, **P** needs to be truncated before we can apply **MAXIMIZE**.

```
compute p=%xsubmat(p,1,nregimes-1,1,nregimes)
@MSRegParmset (parmset=regparms)
nonlin (parmset=msparms) p
frml logl = f=%MSRegFVec(t), fpt=%MSProb(t,f), log(fpt)
@MSFilterInit
maximize(start=%(%MSRegInitVariances()), pstar=%msinit()), $
    parmset=regparms+msparms, $
    method=bhhh, iters=10) logl gstart gend
```

²This uses BHHH rather than BFGS because there will only be minor adjustments due to the slight difference in likelihoods between EM and ML. This won't give BFGS enough iterations to give a reliable estimate of the covariance matrix of the coefficients.

EM needs the smoothed regime probabilities as part of its calculations, so we can just pull the information directly out of the `PSMOOTHED SERIES` of `VECTORS`.

```
set plsmooth = psmooth(t) (1)
graph(footer="Smoothed Probability of Regime 1",max=1.0,min=0.0)
# plsmooth
```

The full program is Example 9.2.

9.2.3 MCMC (Gibbs Sampling)

A quick comparison of the two previous examples with Example 9.3 would confirm that this is quite a bit harder to set up for this case than ML or EM. At least in comparison with EM, this is somewhat misleading, because there's quite a bit of calculation in the `@MSRegEMStep` procedure—it's just that it takes the same form for all switching linear regressions, while MCMC requires more model-specific calculations.

The great advantage of Gibbs sampling is that it can give you a better idea of the shape of the likelihood. For instance, in this model, the intended interpretation of the two regimes is that one is a low mean, and other a high mean. However, a careful look at the results from MCMC shows that it's more of a low variance-high variance split.

The Gibbs sampler requires you to specify an order in which you do the blocks. In this case, we'll start by conditioning on the regime and drawing the other parameters conditional on that. If the switching behavior is fairly clear, you can even start with random assignments for the regimes, and it will eventually work itself out. However, it doesn't hurt to start with something closer to what you actually expect. Here, the regime is set to be 1 for negative values of `DX` and 2 for positive ones. The other parameters are initialized based upon that (though that doesn't matter as much here as with the other estimation methods, since those other parameters are being drawn right away conditional on the regime assignment).

```
gset MSRegime = fix(%if(dx<0.0,1,2))
@MSRegInitial(guessregimes=MSRegime)
compute gstart=%regstart(),gend=%regend()
```

As in Section 8.3.4, we need priors for the transition probabilities:

```
dec vect[vect] gprior(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
dec vect tcounts(nregimes)
```

and a hierarchical prior for the variances:

```

compute nucommon=0.0
compute s2common=0.0
*
dec vect nuprior(nregimes)
dec vect s2prior(nregimes)
compute nuprior=%fill(nregimes,1,4.0)
compute s2prior=%fill(nregimes,1,1.0)

```

What we need now that we didn't before is a prior for the regression coefficients. In order to keep the posterior symmetric in permutations of the regimes, we use the same (Normal) prior for both. This is specified using a mean `VECTOR` and precision (inverse covariance) `SYMMETRIC` matrix. In this case, the mean is zero on all coefficients; the precision is 0 on the constant (which means a flat prior) and 4 (for a standard deviation of $1.0/\sqrt{4} = .5$) on the two lag coefficients. This is a fairly loose prior for an autoregression where we would expect the persistence to be fairly low, since it's on the first differences.

```

dec symm hprior
dec vect bprior
compute hprior=%diag(||0.0,4.0,4.0||)
compute bprior=%zeros(%nreg,1)

```

Inside the simulation loop, the first step is drawing the variances. The procedure `@MSRegResids` computes the regime-specific residuals that are needed for this. Given those, the draws for the common variance factor and the regime-specific variances is basically the same as in the previous chapter:

```

sstats gstart gend resid^2*scommon/sigsqv(MSRegime(t))>>sumsqr
compute scommon=(sumsqr+nucommon*s2common)/$
                    %ranchisqr(%nobs+nucommon)
do i=1,nregimes
    sstats(smpl=MSRegime(t)==i) gstart gend resid^2/scommon>>sumsqr
    compute sigsqv(i)=scommon*(sumsqr+nuprior(i))/$
                    %ranchisqr(%nobs+nuprior(i))
end do i

```

The draws for the coefficients are complicated a bit by this being an autoregression. Conventionally, we don't want either AR polynomial to be explosive. Unlike a SETAR model, where a regime can be explosive because the threshold will cause the process to leave that regime once the value gets too large, the Markov switching process doesn't react to the level of the dependent variable.³ If we make the prior Normal as above, but truncated to the stable region on the AR process, we can implement this by drawing from the posterior Normal and rejecting a draw if it has unstable roots. `%MSRegARIsUnstable` takes a

³However, the process *can* be stable even with an explosive regime if the persistence of the explosive regime isn't high. We'll see that in Chapter 10.

VECTOR of coefficients for an autoregression and returns 1 if they have an unstable (unit or explosive) root. Here, the first coefficient in each of the `BETAS` vectors is for the constant, so we test the VECTOR made up of the second and third. If you have a linear regression for which there are no stability issues like this, you would just delete the `IF` and `GOTO` lines.

```
:redrawbeta
do i=1,nregimes
  cmom(smpl=(MSRegime==i),equation=MSRegEqn) gstart gend
  compute betas(i)=%ranmvpostcmom($
    %cmom,1.0/sigsqv(i),hprior,bprior)
  if MSRegARIsUnstable(%xsubvec(betas(i),2,3))
    goto redrawbeta
end do i
```

We now have draws for all the regression parameters. We want to re-label so as to make the process *means* go from low to high. In other applications, you might re-label based upon the intercept or one of the slope coefficients, or possibly the variance. The means are functions of the regression coefficients:

```
do i=1,nregimes
  compute mu(i)=betas(i)(1)/(1-betas(i)(2)-betas(i)(3))
end do i
compute swaps=%index(mu)
```

We need to swap coefficient vectors, variances and transition rows and columns in the transition matrix based upon the sorting index for the means. Most of this is done by the `@MSRegRelabel` procedure which takes care of relabeling of anything that's a standard part of the MS regression. However, the regime means are derived statistics, so we have to switch *them* manually:

```
if %MSSwapCheck(swaps) {
  disp "Draw" draw "Executing swap"
  @MSRegRelabel swaps
  compute tempmu=mu
  ewise mu(i)=tempmu(swaps(i))
}
```

We now need to draw the regimes, which is relatively simple using the `@MSRegFilter` procedure to do the filtering step, then the standard `@MSSAMPLE` procedure to sample the regimes. This returns a `VECT[INT]` of counts of observations in each regime (called `TCOUNTS` here), which is used in the next step.

```
@MSRegFilter gstart gend
:redrawregimes
@MSSample(counts=tcounts) gstart gend MSRegime
```


This also includes a check that the number of entries in each regime is at or above a minimum level (here `minsize` is made equal to 5). While not strictly necessary,⁴ this is a common step.

```
if %minvalue(tcounts)<minsize {
    disp "Draw" draw "Redrawing regimes with regime of size" $
        %minvalue(tcounts)
    goto redrawregimes
}
```

Note that the program includes messages whenever there's a redraw or swap, which helps you see how smoothly the simulations are going. If you run this, you will note that there are quite a few swaps to put the means back in order. That's a sign that the regimes aren't particularly well-described based upon their means. We also include an alternative where the labels are switched based upon the variance. That turns out to be much more successful from the sampling standpoint, though it might not be as interesting a result. The only difference between the two samplers is that the second one uses

```
compute swaps=%index(sigsqv)
```

instead of

```
compute swaps=%index(mu)
```

The “bookkeeping” that's done as part of the sampler is to compute a running sum of times that each entry is in regime 1 (which converts to an estimated probability when divided by the number of draws) and a full set of estimated parameters.

```
set regime1 gstart gend = regime1+(MSRegime==1)
compute bgibbs(draw)=%parmspeek(mcmcparms)
```

Note that the `MCMCPARMS` `PARMSET` is never actually used in any actual estimation—it is simply to organize the information we want to save. It's defined before the first Gibbs loop as:

```
nonlin(parmset=mcmcparms) betas mu sigsqv p
```

This includes the three standard parameters (`BETAS`, `SIGSQV` and `P`), but also the derived statistics `MU` for the process means. The post-processing uses the `@MCMCPOSTPROC` procedure to compute the means and standard errors from the saved draws for the parameters, which are then displayed in a `REPORT` (Table 9.2, which is from the second sampler. Note that this depends upon random numbers.)

⁴We have an informative prior on the coefficients and on the variance, so the estimates can't collapse to a singular matrix or zero variance.

Table 9.2: MCMC Estimation of MS Regression

BETAS(1)(1)	-0.054	0.025
BETAS(1)(2)	0.359	0.133
BETAS(1)(3)	-0.002	0.111
BETAS(2)(1)	0.058	0.051
BETAS(2)(2)	0.820	0.108
BETAS(2)(3)	-0.299	0.111
MU(1)	-0.082	0.034
MU(2)	0.122	0.113
SIGSQV(1)	0.024	0.006
SIGSQV(2)	0.182	0.033
P(1,1)	0.903	0.039
P(2,1)	0.097	0.039
P(1,2)	0.112	0.047
P(2,2)	0.888	0.047

```
@mcmcpostproc(ndraws=ndraws,mean=bmeans,stderrs=bstderrs) bgibbs
*
report(action=define)
report(atrow=1,atcol=1,fillby=cols) %parmlabels(mcmcparms)
report(atrow=1,atcol=2,fillby=cols) bmeans
report(atrow=1,atcol=3,fillby=cols) bstderrs
report(action=format,picture="*.###")
report(action=show)
```

While this was done with the labeling based upon variances, even the labeling on means produces a very similar result that regime 2 isn't really defined well by its mean, but instead by the variance.

Finally, the raw sums of counts of regime 1 are converted to estimated probabilities and graphed. This graph is largely consistent with what we had from the point estimates.

```
set regime1 gstart gend = regime1/ndraws
graph(header="MCMC Probability of Low Variance Regime")
# regime1
```

Example 9.1 MS Linear Regression: ML Estimation

This estimates a Markov switching linear regression by maximum likelihood. This is discussed in Section 9.2.1.

```

open data q-unemrate.xls
calendar(q) 1948
data(format=xls,org=columns) 1948:1 1993:4 unemp
*
set dx = unemp-unemp{1}
*
* Markov-Switching model. While this is an autoregression, it can be
* handled using the simpler @MSRegression procedures since the entire
* coefficient vector (and the variances) switch.
*
@MSRegression(switch=ch,regimes=2) dx
# constant dx{1 2}
*
@MSRegParmset(parmset=regparms)
nonlin(parmset=msparms) p
*
@MSRegInitial
compute gstart=%regstart(),gend=%regend()
*
* Compute a lower bound on the permitted values of the variance to
* prevent convergence to a zero-variance spike.
*
compute sigmalimit=1.e-6*%minvalue(sigsqv)
*
frml logl = f=%MSRegFVec(t),fpt=%MSProb(t,f),log(fpt)
@MSFilterInit
maximize(start=%(sigmatest=%MSRegInitVariances(),pstar=%msinit()),$
    parmset=regparms+msparms,$
    reject=sigmatest<sigmalimit,$
    method=bfgs,pmethod=simplex,piters=5) logl gstart gend
*
@MSSmoothed gstart gend psmooth
*
* Smoothed probabilities of the regimes
*
set plsmooth = psmooth(t)(1)
graph/footer="Smoothed Probabilities of Regime 1",max=1.0,min=0.0)
# plsmooth
*
do i=1,2
    disp "Conditional Mean for Regime" i $
        betas(i)(1)/(1-betas(i)(2)-betas(i)(3))
end do i

```

Example 9.2 MS Linear Regression: EM Estimation

This estimates a Markov switching linear regression by EM. The details are in Section 9.2.2.

```
open data q-unemrate.xls
calendar(q) 1948
data(format=xls,org=columns) 1948:1 1993:4 unemp
*
set dx = unemp-unemp{1}
*
* Markov-Switching model. While this is an autoregression, it can be
* handled using the simpler MSRegression procedures since the entire
* coefficient vector (and the variances) switch.
*
@MSRegression(switch=ch,regimes=2) dx
# constant dx{1 2}
*
@MSRegInitial
compute gstart=%regstart(),gend=%regend()
@MSRegEMGeneralSetup
do emits=1,50
    @MSRegEMStep gstart gend
    disp "Iteration" emits "Log likelihood" %log1
end do emits
*
* Polish estimates with 10 iterations of BHHH
*
compute p=%xsubmat(p,1,nregimes-1,1,nregimes)
@MSRegParmset(parmset=regparms)
nonlin(parmset=msparms) p
frml log1 = f=%MSRegFVec(t),fpt=%MSProb(t,f),log(fpt)
@MSFilterInit
maximize(start=(%MSRegInitVariances()),pstar=%msinit()),$
    parmset=regparms+msparms,$
    method=bhhh,itors=10) log1 gstart gend
*
* Smoothed probabilities of the regimes. (The EM procedures compute
* PSMOOTH as a side effect).
*
set plsmooth = psmooth(t)(1)
graph(footer="Smoothed Probability of Regime 1",max=1.0,min=0.0)
# plsmooth
```

Example 9.3 MS Linear Regression: MCMC Estimation

This estimates a Markov switching linear regression by Gibbs sampling. This does two samplers: one labeling based upon process means, the other based upon regime variances. The results are similar, but the variance labeling requires no relabels, while the mean labeling requires frequent relabels, indicating that the model actually is better described as low variance vs high variance. This is discussed in Section 9.2.3.

```
open data q-unemrate.xls
calendar(q) 1948
data(format=xls,org=columns) 1948:1 1993:4 unemp
*
set dx = unemp-unemp{1}
*
* Markov-Switching model. While this is an autoregression, it can be
* handled using the simpler MSRegression procedures since the entire
* coefficient vector (and the variances) switch.
*
@MSRegression(switch=ch,regimes=2) dx
# constant dx{1 2}
gset MSRegime = fix(%if(dx<0.0,1,2))
@MSRegInitial(guessregimes=MSRegime)
compute gstart=%regstart(),gend=%regend()
compute p=%mspexpand(p)
*
* Prior for transitions. Weak Dirichlet priors with preference for
* staying in a given regime.
*
dec vect[vect] gprior(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
dec vect tcounts(nregimes)
*
* Hierarchical prior for sigmas
* Uninformative prior on the common component.
*
compute nucommon=0.0
compute s2common=0.0
*
* Priors for the relative variances
*
dec vect nuprior(nregimes)
dec vect s2prior(nregimes)
compute nuprior=%fill(nregimes,1,4.0)
compute s2prior=%fill(nregimes,1,1.0)
*
* Start the common factor at one
*
compute scomon=1.0
*
* Loose prior for the regressions. The intercept has zero precision;
```

```

* the lag coefficients have 0 mean and standard error .5.
*
dec symm hprior
dec vect bprior
compute hprior=%diag(||0.0,4.0,4.0||)
compute bprior=%zeros(%nreg,1)
*
compute nburn=2000,ndraws=5000
*
dec vect mu(nregimes)
*
* This is used only for bookkeeping purposes
*
nonlin(parmset=mcmcparms) betas mu sigsqv p
*
* For relabeling
*
dec vect tempmu(nregimes)
*
* Minimum size of a regime (has to be at least as large as the number of
* coefficients in an equation, usually at least a bit more than that.)
*
compute minsize=5
*
* Bookkeeping arrays
*
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %parmspeek(mcmcparms)
set regimel gstart gend = 0.0
infobox(action=define,lower=-nburn,upper=ndraws,progress) $
  "Gibbs Sampling"
do draw=-nburn,ndraws
  *
  * Draw sigma's given beta's and regimes
  *
  @MSRegResids(regime=MSRegime) resids gstart gend
  *
  * Draw the common variance factor given the relative variances and
  * regimes.
  *
  sstats gstart gend resids^2*scommon/sigsqv(MSRegime(t))>>sumsqr
  compute scommon=(sumsqr+nucommon*s2common)/$
    %ranchisqr(%nobs+nucommon)
  *
  * Draw the relative variances, given the common variances and the
  * regimes.
  *
  do i=1,nregimes
    sstats(smpl=MSRegime(t)==i) gstart gend resids^2/scommon>>sumsqr
    compute sigsqv(i)=scommon*(sumsqr+nuprior(i))/$
      %ranchisqr(%nobs+nuprior(i))
  end do i
  *
  * Draw beta's given sigma's and regimes

```

```

*
:redrawbeta
do i=1,nregimes
    cmom(smpl=(MSRegime==i),equation=MSRegEqn) gstart gend
    compute betas(i)=%ranmvpostcmom($
        %cmom,1.0/sigsqv(i),hprior,bprior)
    if %MSRegARIsUnstable(%xsubvec(betas(i),2,3))
        goto redrawbeta
end do i
*
* Relabel if necessary. They are ordered based upon the process
* means.
*
do i=1,nregimes
    compute mu(i)=betas(i)(1)/(1-betas(i)(2)-betas(i)(3))
end do i
*
compute swaps=%index(mu)
if %MSSwapCheck(swaps) {
    disp "Draw" draw "Executing swap"
    @MSRegRelabel swaps
    *
    * Because the means are derived and are not part of the
    * controlled set of variables, we have to manually switch
    * them.
    *
    compute tempmu=mu
    ewise mu(i)=tempmu(swaps(i))
}
*
* Draw the regimes
*
@MSRegFilter gstart gend
:redrawregimes
@MSSample(counts=tcounts) gstart gend MSRegime
if %minvalue(tcounts)<minsize {
    disp "Draw" draw "Redrawing regimes with regime of size" $
        %minvalue(tcounts)
    goto redrawregimes
}
*
* Draw p's
*
@MSDrawP(prior=gprior) gstart gend p
infobox(current=draw)
if draw>0 {
    *
    * Do the bookkeeping
    *
    set regime1 gstart gend = regime1+(MSRegime==1)
    compute bgibbs(draw)=%parmspeek(mcmcparms)
}
end do draw
infobox(action=remove)

```

```

*
@mcmcpostproc (ndraws=ndraws, mean=bmeans, stderrs=bstderrs) bgibbs
*
report (action=define)
report (atrow=1, atcol=1, fillby=cols) %parmlabels (mcmcparms)
report (atrow=1, atcol=2, fillby=cols) bmeans
report (atrow=1, atcol=3, fillby=cols) bstderrs
report (action=format, picture="*.###")
report (action=show)
*
set regime1 gstart gend = regime1/ndraws
graph(header="MCMC Probability of Low Mean Regime")
# regime1
*****
*
* Same thing but with labels based upon variance, not mean
*
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %parmspeek (mcmcparms)
set regime1 gstart gend = 0.0
infobox (action=define, lower=-nburn, upper=ndraws, progress) $
"Gibbs Sampling"
do draw=-nburn, ndraws
  @MSRegResids (regime=MSRegime) resids gstart gend
  *
  * Draw the common variance factor given the relative variances and
  * regimes.
  *
  sstats gstart gend resids^2*scommon/sigsqv (MSRegime (t)) >>sumsq
  compute scommon=(sumsq+nucommon*s2common)/$
    %ranchisqr (%nobs+nucommon)
  *
  * Draw the relative variances, given the common variances and the
  * regimes.
  *
  do i=1, nregimes
    sstats (smp1=MSRegime (t)==i) gstart gend resids^2/scommon >>sumsq
    compute sigsqv (i)=scommon*(sumsq+nuprior (i))/$
      %ranchisqr (%nobs+nuprior (i))
  end do i
  *
  * Draw beta's given sigma's and regimes
  *
:redrawbeta
  do i=1, nregimes
    cmom (smp1=(MSRegime==i), equation=MSRegEqn) gstart gend
    compute betas (i)=%ranmvpostcmom ($
      %cmom, 1.0/sigsqv (i), hprior, bprior)
    if %MSRegARIsUnstable (%xsubvec (betas (i), 2, 3))
      goto redrawbeta
  end do i

  do i=1, nregimes
    compute mu (i)=betas (i) (1)/(1-betas (i) (2)-betas (i) (3))

```



```

end do i
*
* This time do the swaps based upon the variances
*
compute swaps=%index(sigsqv)
if %MSSwapCheck(swaps) {
    disp "Draw" draw "Executing swap"
    @MSRegRelabel swaps
    compute tempmu=mu
    ewise mu(i)=tempmu(swaps(i))
}
*
* Draw the regimes
*
@MSRegFilter gstart gend
:redrawregimes
@MSSample(counts=tcounts) gstart gend MSRegime
if %minvalue(tcounts)<minsize {
    disp "Draw" draw "Redrawing regimes with regime of size" $
        %minvalue(tcounts)
    goto redrawregimes
}
*
* Draw p's
*
@MSDrawP(prior=gprior) gstart gend p
infobox(current=draw)
if draw>0 {
    *
    * Do the bookkeeping
    *
    set regime1 gstart gend = regime1+(MSRegime==1)
    compute bgibbs(draw)=%parmspeek(mcmcparms)
}
end do draw
infobox(action=remove)
*
@mcmcpostproc(ndraws=ndraws,mean=bmeans,stderrs=bstderrs) bgibbs
*
report(action=define)
report(atrow=1,atcol=1,fillby=cols) %parmslabels(mcmcparms)
report(atrow=1,atcol=2,fillby=cols) bmeans
report(atrow=1,atcol=3,fillby=cols) bstderrs
report(action=format,picture="*.###")
report(action=show)
*
set regime1 gstart gend = regime1/ndraws
graph(header="MCMC Probability of Low Variance Regime")
# regime1

```

Markov Switching Multivariate Regressions

This generalizes the model in Chapter 9 to allow more than one dependent variable. The number of free parameters greatly increases, with, in an n variable model, n times as many regression coefficients and the variance parameter(s) replaced by an $n \times n$ covariance matrices. Needless to say, this greatly increases the difficulty with properly labeling the regimes, since you could, for instance, have equation variances for different components going in different directions among regimes.

The assumption made is that all equations have the same right-side variables. This is appropriate for most MS-VAR's and most MS-VECM's. The MS-VAR support in Chapter 11 is for strict VAR's where there are no explanatory variables except the mean/intercept and the lagged dependent variables—under those circumstances, you can allow for a switch of the process *mean* (as is done in the Hamilton paper). With more general explanatory variables, the process mean is no longer fixed within a regime and so can't be used as the basis for the switch.

The restriction that the explanatory variables are identical makes EM and MCMC quite a bit simpler—if you're willing to stick with the (now much) slower ML, you can relax that, though not by using the support procedures covered in this chapter.

10.1 @MSSYSREGRESSION procedures

The support routines for Markov switching linear systems regressions are in the file **MSSYSREGRESSION.SRC**. You'll generally pull this in by executing the **@MSSysRegression** procedure:

```
@MSSysRegression( options )  
# list of dependent variables (if MODEL option isn't used)  
# list of regressors (if EQUATION/MODEL options aren't used)
```

The main options are **REGIMES=#** of regimes, with a default of 2, and **SWITCH=[C]/CH/H**, which determines what switches—**SWITCH=C** (the default) is coefficients only, **SWITCH=CH** means coefficients and covariance matrices and **SWITCH=H** is covariance matrices only. You can also have some coefficients switching and some fixed—to do this, use the option **NFIX=#** of fixed

coefficients together with SWITCH=C or SWITCH=CH, and arrange the regressor list to have the fixed coefficients first. Note that the coefficients are treated identically across equations—the same set are fixed or are varying in each.

You have two other options which can be used to provide the form of the model. EQUATION=*equation* uses an EQUATION to provide the explanatory variables—if you use it, you only need to provide the first supplementary card (with the dependent variables). MODEL=*model* provides both the dependent variables *and* the explanatory variables, so you don't need either of the supplementary cards. Note that with MODEL, the explanatory variables will still be identical in each equation—that's taken from the first EQUATION in the MODEL.

As a fairly complicated example, this sets up a MS-VECM, where the EQM series is the long-run equilibrium condition. The “short-term” coefficients (on the lagged differenced data) are fixed across regimes (NFI_X=8 covers those—note that the fixed coefficients have to be listed first), with only the CONSTANT and the error correction coefficient (on lagged EQM) allowed to switch.

```
set eqm = %dot(x1r, ||ly, lwp, ln, lns, t ||)
*
set dly  = 100.0*(ly-ly{1})
set dlwp = 100.0*(lwp-lwp{1})
set dlns = 100.0*(lns-lns{1})
set dln  = 100.0*(ln-ln{1})
*
@mssysregression(nfix=8, regimes=2)
# dly dlwp dlns dln
# dly{1 2} dlwp{1 2} dlns{1 2} dln{1 2} constant eqm{1}
```

The FUNCTION which returns the likelihoods across regimes is now called %MSSysRegFVec(*time*) and most other functions and procedures have names similar to those for linear regressions, but with SysReg replacing Reg. The switching coefficients for the regression are in a VECT[RECT] called BETASYS where BETASYS(*S*) is the coefficient vector under regime *S*. Column *J* of BETASYS(*S*) has the coefficients for the equation *j*. The fixed coefficients are in a RECTANGULAR called GAMMASYS, with column *J* again having the coefficients for equation *j*. The variances are either in the SYMMETRIC variable SIGMA if the variances don't switch or in the VECTOR[SYMM] SIGMAV if they do.

10.1.1 Impulse Response Functions

While MS systems regressions can be applied more generally, they will typically be used for dynamic models like VAR's and VECM's. For models like that, there will generally be relatively few estimated parameters that can be interpreted on their own—individual lag coefficients tell you relatively little, as their effect depends upon all the other lag coefficients. In the VAR literature, it's generally

considered to be bad form (and a waste of space) to list the lag coefficients. That goes double (or more) if you have regime-specific lag coefficients. Instead, as in the VAR literature, it makes more sense to present the dynamics in the form of impulse response function graphs.

Impulse responses in a MS model are very different from those in threshold models (whether sharp or smooth transitions). In the threshold models, the regimes are connected by the (observable) switching criterion so the model has to be simulated as a whole. It's possible for one regime to have explosive dynamics as long as it drives the process towards the switch to the other regime. In the MS model, however, the regime-switching process isn't observable and doesn't respond to the values of the endogenous variables. So, instead, you can compute and display regime-specific IRF's, that is, the IRF treating the regime as unchanging. Now, it's still possible for a regime to be explosive, as long as its persistence is low. For instance, if we take as a simple example the one-lag autoregression

$$y_t = \begin{cases} 1.2y_{t-1} + u_t & \text{if } S_t = 1 \\ 0.8y_{t-1} + u_t & \text{if } S_t = 2 \end{cases}$$

and if the probability of being in regime 1 next period is less than the probability of being in regime 2 instead (starting in either regime), then the overall process *isn't* explosive—if you take the expectation across regimes, it has an combined autoregressive coefficient less than 1.

If you want to do *forecasts* of a MS process, you would need to do a joint simulation of the switching process and the regime processes,¹ but you really can't use that to extend the idea of the non-linear impulse response function because the addition of the shock has a very different effect than it would in a model where the regime is observable.

To compute regime-specific impulse responses (or anything similar), use the **@MSSysRegSetModel** procedure. This takes one option: `REGIME=desired regime`, and fills the global `MODEL` variable named `MSSysRegModel` with the coefficients and covariance matrix for that regime. You can then use `MODEL=MSSysRegModel` in, for instance, an **IMPULSE** instruction. If you have a just-identified structural model, you can compute an appropriate factor using (for instance) **CVMODEL** or the **@ShortAndLong** procedure the way you would for a regular VAR, applying it to the regime-specific covariance matrix (and lag sums if needed for long-run restrictions). Note that an *overidentified* structural model isn't so simple—unlike a standard VAR, the overidentification affects the entire estimation process.

¹Because the switching process is exogenous, you can actually simulate it first through the entire forecast period, then work through the forecasts entry by entry taking the simulated regime as given.

10.2 The example

The example that we'll use in this Chapter is based upon the model in Ehrmann et al. (2003).² This is a three variable, three lag vector autoregression on capacity utilization, consumer prices and oil prices. Because all coefficients are switching, this can be handled using the simpler `@MSSYSREGRESSION` procedures rather than `@MSVARESETUP` that will be covered in Chapter 11.

The setup code, which will be used in all three estimation methods, is:

```
system(model=varmodel)
variables logcutil logcpi logpoil
lags 1 to 3
det constant
end(system)
*
@mssysregression(model=varmodel, regimes=2, switch=ch)
```

10.2.1 Maximum Likelihood

As in Chapter 8, you must choose in which form to estimate the transition probabilities. With just two regimes, we'll use the simpler direct estimate of the P matrix. The parameter sets for the regression part and the Markov switching part are:

```
@mssysregparmset (parmset=regparms)
nonlin (parmset=msparms)  p
```

Because there are so many possible ways to jiggle the switching parameters to distinguish the regimes, `@MSSysRegInitial` doesn't attempt to do anything other than initialize the parameters based upon a multivariate regression, with all the `BETASYS` matrices made equal to the same full-sample estimates. In this example, for regime 2, we start with a higher variance for the oil price (third variable) and lower variance for the macro variables:

```
compute gstart=1973:4, gend=2000:12
@mssysreginitial gstart gend
compute sigmav(2)=%diag(%xdia(gsigmav(1)).*|.25,.25,4.0|)
```

`SIGMAV(1)` will be the full-sample estimates, while `SIGMAV(2)` will be a diagonal matrix with the diagonal from `SIGMAV(1)` multiplied by (in order) .25, .25 and 4.0. Since there are likely to be several local modes, you'll probably need to experiment a bit and see how sensitive the estimates are to the choice for the guess values. An alternative approach is to use the `GUESS` option on `@MSSysRegInitial` to do estimates based upon the residuals from a

²The data set is a reconstruction because the authors were not permitted to provide the original data.

least squares VAR to separate the regimes based upon large values of the oil residual—for illustration, we’ll do that with the EM estimation.

With a multivariate regression with switching covariance matrices, we not only have to worry about likelihood spikes with (near) zero variances, but also near-singular matrices with non-zero values. The appropriate protection against this is based upon the log determinant of the covariance matrix—we can’t allow that to get too small for one regime relative to the others. The function `%MSSysRegInitVariances()` returns the minimum log determinant of the covariance matrices. A reasonable lower limit for that is something like 12 times the number of dependent variables (here 3) less than the full-sample log determinant.³ The following is the estimation code for maximum likelihood, including the rejection test for small variances:

```
compute logdetlimit=%MSSysRegInitVariances()-12.0*nvar
*
frml logl = f=%MSSysRegFVec(t), fpt=%MSProb(t,f), log(fpt)
@MSFilterInit
maximize(start=$
    %(logdet=%MSSysRegInitVariances(), pstar=%MSSysRegInit()), $
    parmset=regparms+msparms, reject=logdet<logdetlimit, $
    method=bfgs, iters=500, pmethod=simplex, peters=5) logl gstart gend
```

Note that this takes a *very* long time and hangs up at a local mode, which actually doesn’t show behavior all that much different from the better mode found using EM (Section 10.2.2) which may indicate that there are quite a few local modes.

The full program is Example 10.1. An edited output is provided in Table 10.1. As described above, `BETASYS(1)` are the coefficients on the regressors in regime 1. `BETASYS(1)(i,j)` is the coefficient on regressor *i* in equation *j*. What is shown in the table are the 10 coefficients from equation 1 (for `LOGCUTIL`) (plus 2 more from equation 2). As with any model like this set up with **SYSTEM**, the variables are grouped by lags of the dependent variables, so coefficients 1 to 3 are the lags of `LOGCUTIL`, 4 to 6 are the lags of `LOGCPI`, etc. Again, in a dynamic model like this, these coefficients can be almost impossible to interpret, so it makes little sense to report them. We could do point estimates of impulse response functions based upon these estimates, but we’ll save the IRF’s for the MCMC estimates (Section 10.2.3) which will allow for error bands as well.

By contrast, the `SIGMAV` and the *P*’s *can* be interpreted. What is clear here is that the desired sample split based primarily on the variance of oil seems to have come true—the variance of the oil residual in regime 2 is 95 (`SIGMAV(2)(3,3)`) while it’s just 3.3 in regime 1. The transition probabilities both indicate *very* persistent regimes which can be seen very clearly in

³This allows regime variances to be around 10^{-6} ($\exp(-12) \approx 6.e - 6$) times the full-sample value, which, while obviously quite small, isn’t small enough to allow likelihood spikes.

Table 10.1: Output from Maximum Likelihood Estimation

MAXIMIZE - Estimation by BFGS					
Convergence in 647 Iterations. Final criterion was 0.0000000 <= 0.0000100					
Monthly Data From 1973:04 To 2000:12					
Usable Observations		333			
Function Value		-1158.6866			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	BETASYS(1)(1,1)	1.3069	0.0083	157.2702	0.0000
2.	BETASYS(1)(2,1)	-0.1450	0.0140	-10.3567	0.0000
3.	BETASYS(1)(3,1)	-0.2058	0.0103	-19.9606	0.0000
4.	BETASYS(1)(4,1)	-0.0001	0.0110	-0.0049	0.9961
5.	BETASYS(1)(5,1)	-0.5514	0.0072	-76.3854	0.0000
6.	BETASYS(1)(6,1)	0.5676	0.0167	33.9291	0.0000
7.	BETASYS(1)(7,1)	0.0354	0.0184	1.9258	0.0541
8.	BETASYS(1)(8,1)	-0.0176	0.0057	-3.0763	0.0021
9.	BETASYS(1)(9,1)	-0.0281	0.0216	-1.3020	0.1929
10.	BETASYS(1)(10,1)	15.4911	6.5667	2.3590	0.0183
11.	BETASYS(1)(1,2)	-0.0119	0.0047	-2.5367	0.0112
12.	BETASYS(1)(2,2)	0.0631	0.0064	9.9134	0.0000
	⋮				
61.	SIGMAV(1)(1,1)	0.7739	0.0871	8.8845	0.0000
62.	SIGMAV(1)(2,1)	0.0333	0.0166	2.0129	0.0441
63.	SIGMAV(1)(2,2)	0.0544	0.0065	8.4291	0.0000
64.	SIGMAV(1)(3,1)	0.2159	0.1297	1.6651	0.0959
65.	SIGMAV(1)(3,2)	0.0575	0.0367	1.5660	0.1173
66.	SIGMAV(1)(3,3)	3.2769	0.3969	8.2560	0.0000
67.	SIGMAV(2)(1,1)	0.2913	0.0292	9.9796	0.0000
68.	SIGMAV(2)(2,1)	-0.0009	0.0037	-0.2451	0.8064
69.	SIGMAV(2)(2,2)	0.0081	0.0009	8.6488	0.0000
70.	SIGMAV(2)(3,1)	-0.3833	0.4053	-0.9457	0.3443
71.	SIGMAV(2)(3,2)	0.1402	0.0662	2.1181	0.0342
72.	SIGMAV(2)(3,3)	94.9592	9.7154	9.7741	0.0000
73.	P(1,1)	0.9755	0.0137	71.1757	0.0000
74.	P(1,2)	0.0189	0.0098	1.9250	0.0542

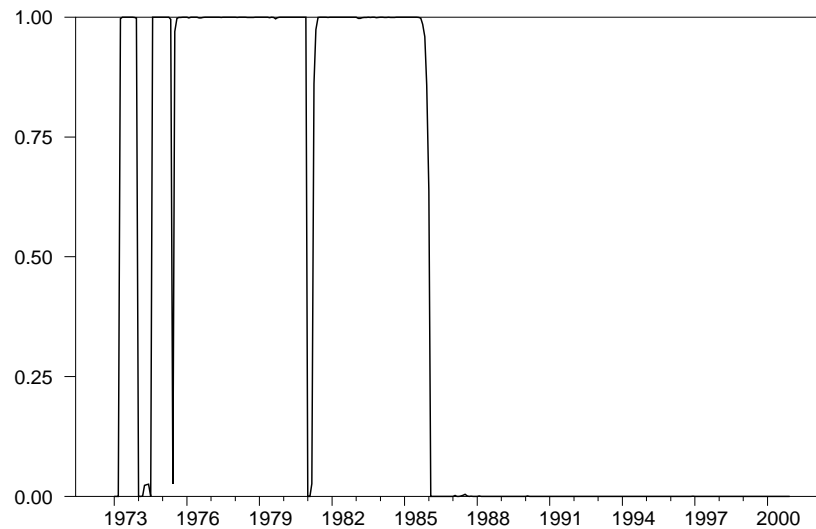


Figure 10.1: Smoothed Probabilities of Regimes

the smoothed estimates of the regimes (Figure 10.1). In fact, this is very close to making the final transition to regime 2 permanent, indicating that a structural break model might be more appropriate. (Note that U.S. domestic oil prices were price-controlled until 1981, so there was, indeed, a break in the legal structure part way through the data range).

10.2.2 EM

For illustration, we'll use a different approach to guess values. If you have a fairly strong prior belief as to how the regimes are likely to break, you can use that information to get guess values. Here, we're expecting the oil price variance to be the main determinant. This estimates the VAR by least squares and sets up a guess for the regimes based upon the size of the oil price residual—entries where the residual is less than one-standard deviation are in regime 1 and those where it's greater are in regime 2.

```
compute oilpos=3
*
estimate(model=MSSysRegModel,resids=varres)
compute gstart=%regstart(),gend=%regend()
*
* Guess the regimes to be 1 and 2 based upon values of
* standardized residuals for oil in the VAR.
*
gset MSRegime = $
    %if(abs(varres(oilpos))/sqrt(%sigma(oilpos,oilpos))<1.0,1,2)
*
@MSSysRegInitial(guess=MSRegime) gstart gend
```


Note that this is a very crude way of separating the regimes as each entry is processed separately, with no regard to its neighbors, while we're expecting the regimes to be more persistent than that. If we wanted to be more exacting (if, for instance, this didn't work very well), using a moving average or exponential smooth of squared residuals would give us a better guess as to the final regime values.

Most of the specialized calculations required for EM estimation are included in a set of procedures within the **MSSYSREGRESSION.SRC** file. As a result, after the model has been set up, all that is required is the following:

```
@MSSysRegEMGeneralSetup
do emits=1,50
    @MSSysRegEMStep gstart gend
    disp "Iteration" emits "Log likelihood" %logl
end do emits
```

which does 50 EM iterations. We again follow this with 10 BHHH iterations to get the covariance matrix.

```
compute p=%xsubmat(p,1,nregimes-1,1,nregimes)
frml logl = f=%MSSysRegFVec(t),fpt=%MSProb(t,f),log(fpt)
@MSFilterInit
maximize(start=$
    %(logdet=%MSSysRegInitVariances(),pstar=%MSSysRegInit()),$
    parmset=regparms+mparms,method=bhhh,itors=10) logl gstart gend
```

The full program is Example 10.2.

10.2.3 MCMC (Gibbs Sampling)

This is several times longer than the ML and EM versions. Part of that is because this includes the calculation and display of IRF's with error bands, but mostly it's because MCMC is quite a bit more complicated than the other methods and MCMC is quite a bit more complicated for multivariate regressions than it is for univariate ones.

This uses the same basic setup as for EM, that is, it uses guess regimes for initializing the parameters. However, here it uses those guess regimes to initialize the chain, first drawing the various other parameters conditional on the initial guess regimes, then redrawing the regimes towards the end.

We'll use the same priors for the transitions as in Example 9.3:

```
dec vect[vect] gprior(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
```

We'll again use a hierarchical prior for the covariance matrices; the extension to Wishart distributions is described on page 312.

```
dec vect nuprior(nregimes)
compute nuprior=%fill(nregimes,1,6.0)
*
dec vect[series] vresids(nvar)
dec vect[symm] uu(nregimes)
dec vect tcounts(nregimes)
```

In this example, we're using a flat prior on the regression coefficients. A "Minnesota" type prior might make more sense, but would be more complicated to set up.

```
dec symm hprior
dec vect bprior
compute hprior=%zeros(nvar*nreg,nvar*nreg)
compute bprior=%zeros(nvar*nreg,1)
```

Inside the simulation loop, the first step is drawing the covariance matrices. The procedure `@MSSysRegResids` computes the `VECT[SERIES]` of residuals, using the appropriate coefficients in the currently sampled regimes. We then need to compute the cross products of those residuals in the subsample for each regime (into `UU(I)`), and the count of the subsample into `TCOUNTS(I)`.

```
@MSSysRegResids(regime=MSRegime) vresids gstart gend
ewise uu(i)=%zeros(nvar,nvar)
ewise tcounts(i)=0.0
do time=gstart,gend
  compute uu(MSRegime(time))=uu(MSRegime(time))+$
    %outerxx(%xt(vresids,time))
  compute tcounts(MSRegime(time))=tcounts(MSRegime(time))+1
end do time
```

The common Σ matrix is then drawn using the following:

```
compute uucommon=%zeros(nvar,nvar)
do k=1,nregimes
  compute uucommon=uucommon+nuprior(k)*inv(sigmax(k))
end do k
compute sigma=%ranwishartf(%decomp(inv(uucommon)),%sum(nuprior))
```

Given the new `SIGMA`, the regime-specific covariance matrices are drawn by:

```
do k=1,nregimes
  compute sigmax(k)=%ranwisharti($
    %decomp(inv(uu(k)+nuprior(k)*sigma)),tcounts(k)+nuprior(k))
end do k
```

The regime-specific coefficients are drawn using:

```
do i=1,nregimes
  cmom(smpl=(MSRegime==i),model=MSSysRegModel) gstart gend
  compute betasys(i)=%reshape(%ranmvkroncmom($
    %cmom,inv(sigmav(i)),hprior,bprior),nreg,nvar)
end do i
```

which is similar to the univariate regression, except we use the multivariate extensions for drawing the simulated coefficients. `MSSysRegModel` is defined when the system is set up originally to provide the information needed by the **CMOMENT** instruction.

The re-labeling in this case will be based upon the variance of the 3rd equation (for oil)—early on, we defined `OILPOS` as the position of oil. We extract the (3,3) component from the covariance matrices and create the swap index from that. The `@MSSysRegRelabel` procedure then rearranges all the switching components of the model (`P`, `SIGMAV` and `BETASYS`) based upon that.

```
ewise voil(i)=sigmav(i)(oilpos,oilpos)
compute swaps=%index(voil)
if %MSSwapCheck(swaps) {
  disp "Draw" draw "Executing swap"
  @MSSysRegRelabel swaps
}
```

We now need to draw the regimes and then the transition matrix. The only difference between this and the univariate model is the very first line, which is

```
@MSSysRegFilter gstart gend
```

rather than the same with `@MSRegFilter`.

The “bookkeeping” for the draws that we keep has three parts: for each entry, we sum up the number of times that it’s in regime 1, which will convert (at the end) to an estimated (smoothed) probability of regime 1.⁴ The entire parameter vector is saved for each draw, so we’ll have complete records of all the draws for all of them, allowing for diagnostics and density estimation if we want those. Finally, we compute and save the impulse responses. As described earlier (Section 10.1.1), the IRF’s for a regime can be computed by extracting the regime-specific model using `@MSSysRegSetModel`, and using the **IMPULSE** instruction. Now the authors did something specific in handling this that you might not want to repeat in a different situation—they standardized Cholesky factor shocks to unit impacts on own variables, and in particular, the oil shock (which is last in the Cholesky order) has a unit impact on oil in each regime. As

⁴Gibbs sampling takes the entire data set as given so it produces smoothed and not filtered results.

we saw in the results, the variance of the oil shock is around 30 times higher in regime 2 than regime 1, so the natural shock size is 5-6 times higher. The idea is to compare the dynamic responses to the shock without swamping the regime 1 effects with the sheer size of the shocks in regime 2. The calculation of `FACTOR` in the below first does the Cholesky factor, then divides the columns by the diagonal elements of the matrix.

```
@MSSysRegSetModel (regime=1)
compute factor=%decomp(sigmav(1)), $
        factor=%ddivide(factor,%xdiag(factor))
impulse (noprint,model=MSSysRegModel,results=impulses1,$
        steps=steps,factor=factor)
```

The same is done for regime 2. Note well that this isn't an actual "factor" of the covariance matrix and can't be used for an error decomposition. If you want an FEVD, you would apply **ERRORS** to the model without the calculated `FACTOR` and just allow the standard unscaled Cholesky factor to apply.⁵

This "interleaves" the two sets of responses so it looks like there are six shocks to the three variables, with the first, third and fifth being shocks from regime 1 and the second, fourth and sixth from regime 2.

```
dim %%responses (draw) (%rows(impulses1)*%cols(impulses1)*2,steps)
ewise %%responses (draw) (i,j)=$
        ix=%vec(%xt(impulses1,j)~~%xt(impulses2,j)),ix(i)
```

The post-processing for the regime probabilities and parameters are basically the same as they were in Example 9.3, so we'll focus on the presentation of the IRF's. Because we put all the responses into the single `%%RESPONSES` array, we can use `@MCGraphIRF` to do the graphs. In this case, we're only interested in the responses to the two oil price shocks, so we use the `ONLYSHOCKS` option to show only shocks 5 and 6 (computed using the position of oil in the model). This produces Figure 10.2.

```
@MCGraphIRF (model=MSSysRegModel,onlyshocks=||2*oilpos-1,2*oilpos||,$
        shocklabels=||"Regime 1","Regime 2"||,$
        varlabels=||"Cap Util","CPI","Oil Price"||)
```

Again, remember that this is standardized to a common own-impact of 1.0. Regime 1 is the "low variance" and regime 2 is the high variance. In both cases, the oil shocks are last in the order, so the impacts on capacity utilization and prices are zero by construction. Clearly, there is a major difference in the dynamics of the responses of the rest of the economy in the second regime compared to the first.

⁵The standardization loses the scale of the shocks and it's the scale that largely determines the error decomposition.

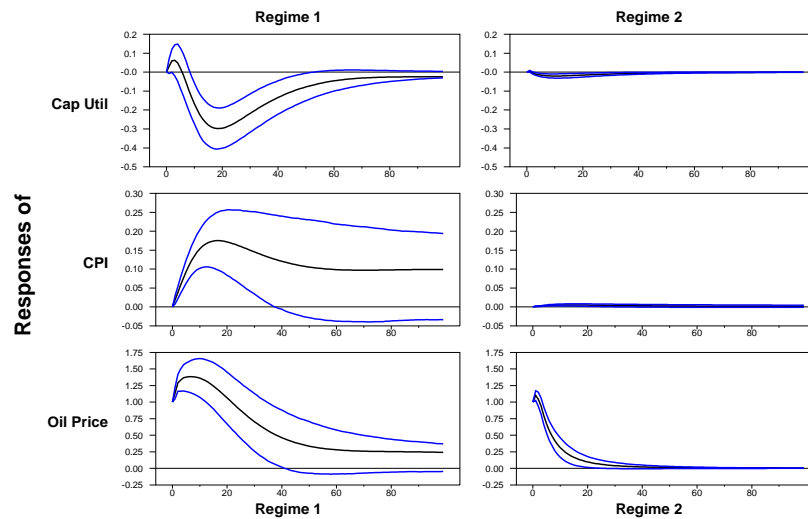


Figure 10.2: Regime-Specific Responses to Standardized Oil Price Shocks

10.3 Systems Regression with Fixed Coefficients

The example used so far in this chapter had all coefficients and the covariance matrices switching. This simplifies both EM and MCMC, since the regression coefficients can be estimated using separate standard system regressions on each regime.⁶ With a systems estimator, once there is any common component, whether it's the covariance matrix or a regression coefficient, the likelihood function can't be "factored" from a full system across regimes down to analysis of separate regimes. That can create very large parameter sets and complicated optimization problems. While this is doable (the combined regression model is still linear), it's simpler, both with EM and with MCMC, to break the optimization problem up into separate parameter sets, treating the fixed and the switching separately. This is fairly standard practice in MCMC, where we always need to block the parameters for convenience. In the case of EM, it means that the M step doesn't maximize the probability-weighted likelihood, but merely improves it. The use of a partial rather than full optimization is known as Generalized EM.

There are three blocks of parameters: the covariance matrices, the fixed regression coefficients and the switching regression coefficients. The simplest process is to do the regression coefficients first. Taking those as given, we can compute the residuals and estimate the covariance matrices using straightforward methods. The switching regression coefficients aren't especially hard given the other two—we subtract off the contribution of the fixed regressors from the dependent variables and treat those partial residuals as if they were the dependent variables in a (now) fully switching model. It's the estimation of

⁶ML is unaffected by the fixed vs switching structure since it simply computes the log likelihood and maximizes it by brute force.

the fixed regression coefficients that's the most complicated of the three. This requires stacking and weighting the equations.

If you use EM and the `@MSSysRegEM...` procedures, all of this is done automatically. It isn't if you do MCMC since you could be using a prior or applying the rejection method to some of your draws. For drawing the fixed coefficients, you need something like:

```
@MSSysRegFixedCMOM(regimes=MSRegime) gstart gend xxfixed
compute gammasys=%reshape(%ranmvpostcmom($
    xxfixed,1.0,hpriorGamma,bpriorGamma),MSSysRegNFix,nvar)
```

The procedure `@MSSysRegFixedCMOM` generates a cross product matrix of the stacked and weighted (using the covariance matrices) regressors and dependent variables. There are `MSSysRegNFix` fixed coefficients in each of `nvar` equations, so this builds the matrix `XXFIXED` which is symmetrical with dimension $(\text{MSSysRegNFix} \times \text{nvar}) + 1$, where the +1 is the stacked and weighted dependent variable. The mean and precision of the prior have to be the proper sizes. For a flat prior, they would be set up with

```
dec symm hpriorGamma
dec vect bpriorGamma
compute hpriorGamma=%zeros(nvar*MSSysRegNFix,nvar*MSSysRegNFix)
compute bpriorGamma=%zeros(nvar*MSSysRegNFix,1)
```

The 1.0 in the second argument of `%RANMVPOSTCMOM` is there because the covariance matrices have already been incorporated into the cross product matrix.

The draws for the switching coefficients is similar to that used when the entire coefficient vector switches. You just need to compute the residuals from the fixed part (done with `@MSSysRegFixResids`) and use a separate procedure for computing the systems cross product matrix (with `@MSSysRegSwitchCMOM`) for the switching coefficients only.

```
@MSSysRegFixResids MSSysRegU gstart gend
do i=1,nstates
    @MSSysRegSwitchCMOM(regimes=MSRegime,s=i,u=MSSysRegU) $
        gstart gend xxswitch
    compute betasys(i)=%reshape(%ranmvkroncmom($
        xxswitch,inv(sigmav(i)),hpriorBeta(i),bpriorBeta(i)), $
        MSSysRegNSwitch,nvar)
end do i
```

The number of switching coefficients per equation is `MSSysRegNSwitch`. A flat prior for each of the equations is set up with:

```
dec vect[symm] hpriorBeta(nvar)
dec vect[vect] bpriorBeta(nvar)
ewise hpriorBeta(i)=%zeros(nvar*MSSysRegNSwitch,$
    nvar*MSSysRegNSwitch)
ewise bpriorBeta(i)=%zeros(nvar*MSSysRegNSwitch,1)
```

Example 10.1 MS Systems Regression: ML Estimation

This estimates a Markov switching linear system by maximum likelihood. This is discussed in Section 10.2.1.

```

open data eev_reconstructed.rat
calendar(m) 1973:1
data(format=rats) 1973:01 2000:12 oilprice pcusle cumfg
*
set logcutil = 100.0*log(cumfg)
set logpoil  = 100.0*log(oilprice)
set logcpi   = 100.0*log(pcusle)
*
@varlagselect(Lags=6,crit=hq)
# logcutil logcpi logpoil
*
system(model=varmodel)
variables logcutil logcpi logpoil
lags 1 to 3
det constant
end(system)
*
@mssysregression(model=varmodel,regimes=2,switch=ch)
*
@mssysregparmset(parmset=regparms)
nonlin(parmset=msparms) p
*
compute gstart=1973:4,gend=2000:12
@mssysreginitial gstart gend
*
* Reset the 2nd covariance matrix to make regime 2 have a high
* variance for oil.
*
compute sigmav(2)=%diag(%xdiag(sigmav(1)).*|.25,.25,4.0|)
compute logdetlimit=%MSSysRegInitVariances()-12.0*3
*
frml logl = f=%MSSysRegFVec(t),fpt=%MSProb(t,f),log(fpt)
@MSFilterInit
maximize(start=$
    %(logdet=%MSSysRegInitVariances(),pstar=%MSSysRegInit()),$
    parmset=regparms+msparms,reject=logdet<logdetlimit,$
    method=bfgs,itors=1000,pmethod=simplex,piters=5) logl gstart gend
*
@msssmoothed gstart gend psmooth
*
* Smoothed probabilities of the regimes
*
set plsmooth = psmooth(t)(1)
graph(footer="Smoothed Probabilities of Regime 1",max=1.0,min=0.0)
# plsmooth

```


Example 10.2 MS Systems Regression: EM Estimation

This estimates a Markov switching linear system by EM. The details are in Section 10.2.2. For illustration, this uses a different method of initializing the parameters based upon guessing the regime breakdown.

```

open data eev_reconstructed.rat
calendar(m) 1973:1
data(format=rats) 1973:01 2000:12 oilprice pcusle cumfg
*
set logcutil = 100.0*log(cumfg)
set logpoil  = 100.0*log(oilprice)
set logcpi   = 100.0*log(pcusle)
*
@varlagselect(Lags=6,crit=hq)
# logcutil logcpi logpoil
*
system(model=varmodel)
variables logcutil logcpi logpoil
lags 1 to 3
det constant
end(system)
*
estimate(model=varmodel,resids=varresids)
*
@mssysregression(model=varmodel,regimes=2,switch=ch)
*
* Position of oil in the model (dependent variable)
*
compute oilpos=3
*
* Estimate the model by least squares to help with the
* initialization of regimes.
*
estimate(model=MSSysRegModel,resids=varres)
compute gstart=%regstart(),gend=%regend()
*
* Guess the regimes to be 1 and 2 based upon values of
* standardized residuals for oil in the VAR.
*
gset MSRegime = $
    %if(abs(varres(oilpos))/sqrt(%sigma(oilpos,oilpos))<1.0,1,2)
*
@mssysregInitial(guess=MSRegime) gstart gend
*
@mssysregEMGeneralSetup
do emits=1,50
    @mssysregEMStep gstart gend
    disp "Iteration" emits "Log likelihood" %logl
end do emits
*
* Polish estimates with 10 iterations of BHHH. Set up the parameter
* sets and the initialize the regime probability matrices (required

```

```

* for ML, but not EM, which uses its own).
*
@MSSysRegParmset (parmset=regparms)
nonlin(parmset=msparms) p
*
* The "P" matrix needs to be truncated to remove the bottom row since
* that's the form used for ML estimation.
*
compute p=%xsubmat(p,1,nregimes-1,1,nregimes)
frml logl = f=%MSSysRegFVec(t),fpt=%MSProb(t,f),log(fpt)
*
@MSFilterInit
maximize(start=$
    %(logdet=%MSSysRegInitVariances(),pstar=%MSSysRegInit()),$
    parmset=regparms+msparms,method=bhhh,itters=200) logl gstart gend
*
* Smoothed probabilities of the regimes. (The EM procedures compute
* PSMOOTH as a side effect).
*
set plsmooth = psmooth(t)(1)
graph(footer="Smoothed Probability of Regime 1",max=1.0,min=0.0)
# plsmooth

```

Example 10.3 MS Systems Regression: MCMC Estimation

This estimates a Markov switching linear system by Gibbs sampling and generates regime-specific impulse responses with error bands. (Error bands require use of either MCMC or bootstrapping, not the point estimates done in the the EM and ML examples). See Section 10.2.3 for technical information.

```

open data eev_reconstructed.rat
calendar(m) 1973:1
data(format=rats) 1973:01 2000:12 oilprice pcusle cumfg
*
set logcutil = 100.0*log(cumfg)
set logpoil = 100.0*log(oilprice)
set logcpi = 100.0*log(pcusle)
*
@varlagselect(Lags=6,crit=hq)
# logcutil logcpi logpoil
*
system(model=varmodel)
variables logcutil logcpi logpoil
lags 1 to 3
det constant
end(system)
*
@mssysregression(model=varmodel,regimes=2,switch=ch)
*
* Position of oil in the model (dependent variable)
*
compute oilpos=3

```

```

*
* Estimate the model by least squares to help with the
* initialization of regimes.
*
estimate(model=MSSysRegModel, resids=varres)
compute gstart=%regstart(), gend=%regend()
*
* Guess the regimes to be 1 and 2 based upon values of
* standardized residuals for oil in the VAR.
*
gset MSRegime = $
    %if(abs(varres(oilpos))/sqrt(%sigma(oilpos,oilpos))<1.0,1,2)
*
@mssysreginitial(guess=MSRegime) gstart gend
*
* Convert the P to the full size matrix
*
compute p=%mspexpand(p)
*
* Prior for transitions. Weak Dirichlet priors with preference
* for staying in a given regime.
*
dec vect[vect] gprior(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
*
* Hierarchical prior for sigmas
* Uninformative prior on the common component.
*
* Priors for the relative Wisharts
*
dec vect nuprior(nregimes)
compute nuprior=%fill(nregimes,1,6.0)
*
dec vect[series] vresids(nvar)
dec vect[symm] uu(nregimes)
dec vect tcounts(nregimes)
*
* Flat prior for the regressions.
*
dec symm hprior
dec vect bprior
compute hprior=%zeros(nvar*nreg,nvar*nreg)
compute bprior=%zeros(nvar*nreg,1)
*
compute nburn=2000,ndraws=5000
nonlin(parmset=allparms) betasys sigmav p
*
* For relabeling
*
dec vect voil(nregimes)
*
* Minimum size of a regime (has to be at least as large as the
* number of coefficients in an equation, usually at least a

```

```

* bit more than that.)
*
compute minsize=15
*
* Bookkeeping arrays for the parameters
*
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %parmspeek(allparms)
set regime1 gstart gend = 0.0
*
* Bookkeeping information for impulse responses
*
declare vect[rect] %%responses
dim %%responses(ndraws)
compute steps=100
*
infobox(action=define, lower=-nburn, upper=ndraws, progress) $
  "Gibbs Sampling"
do draw=-nburn, ndraws
  *
  * Draw sigma's given beta's and regimes
  * Compute the regime-specific residuals
  *
  @MSSysRegResids(regime=MSRegime) vresids gstart gend
  *
  * Compute the sums of squared residuals for each regime.
  *
  ewise uu(i)=%zeros(nvar, nvar)
  ewise tcounts(i)=0.0
  do time=gstart, gend
    compute uu(MSRegime(time))=uu(MSRegime(time))+$
      %outerxx(%xt(vresids, time))
    compute tcounts(MSRegime(time))=tcounts(MSRegime(time))+1
  end do time
  compute uucommon=%zeros(nvar, nvar)
  do k=1, nregimes
    compute uucommon=uucommon+nuprior(k)*inv(sigmapv(k))
  end do k
  *
  * Draw the common sigma given the regime-specific ones.
  *
  compute sigma=%ranwishartf(%decomp(inv(uucommon)), %sum(nuprior))
  *
  * Draw the regime-specific sigmas given the common one.
  *
  do k=1, nregimes
    compute sigmapv(k)=%ranwisharti($
      %decomp(inv(uu(k)+nuprior(k)*sigma)), tcounts(k)+nuprior(k))
  end do k
  *
  * Draw beta's given sigma's and regimes
  *
  do i=1, nregimes
    cmom(smpl=(MSRegime==i), model=MSSysRegModel) gstart gend
  end do i
end do draw

```

```

        compute betasys(i)=%reshape(%ranmvkroncmom($
            %cmom,inv(sigmav(i)),hprior,bprior),nreg,nvar)
    end do i
    *
    * Relabel if necessary. They are ordered based upon the variance
    * of oil.
    *
    ewise voil(i)=sigmav(i)(oilpos,oilpos)
    compute swaps=%index(voil)
    if %MSSwapCheck(swaps) {
        disp "Draw" draw "Executing swap"
        @MSSysRegRelabel swaps
    }
    *
    * Draw the regimes
    *
    @MSSysRegFilter gstart gend
:redrawregimes
    @MSSample(counts=tcounts) gstart gend MSRegime
    if %minvalue(tcounts)<minsize {
        disp "Draw" draw "Redrawing regimes with regime of size" $
            %minvalue(tcounts)
        goto redrawregimes
    }
    *
    * Draw p's
    *
    @MSDrawP(prior=gprior) gstart gend p
    infobox(current=draw)
    if draw>0 {
        *
        * Do the bookkeeping
        *
        set regimel gstart gend = regimel+(MSRegime==1)
        *
        compute bgibbs(draw)=%parmspeek(allparms)
        *
        * Save IRF's for regime 1. Cholesky factor standardized to unit
        * shock is used.
        *
        @MSSysRegSetModel(regime=1)
        compute factor=%decomp(sigmav(1)), $
            factor=%ddivide(factor,%xdiag(factor))
        impulse(noprint,model=MSSysRegModel,results=impulses1,$
            steps=steps,factor=factor)
        *
        * Save IRF's for regime 2
        *
        @MSSysRegSetModel(regime=2)
        compute factor=%decomp(sigmav(2)), $
            factor=%ddivide(factor,%xdiag(factor))
        impulse(noprint,model=MSSysRegModel,results=impulses2,$
            steps=steps,factor=factor)
        *
    }

```

```

    * This will interleave the shocks from the two regimes.
    *
    dim %%responses(draw) (%rows(impulses1)*%cols(impulses1)*2,steps)
    ewise %%responses(draw) (i,j)=$
        ix=%vec(%xt(impulses1,j)~~%xt(impulses2,j)),ix(i)
    }
end do draw
infobox(action=remove)
@mcmcpstproc(ndraws=ndraws,mean=bmeans,stderrs=bstderrs) bgibbs
*
report(action=define)
report(atrow=1,atcol=1,fillby=cols) %parmlabels(allparms)
report(atrow=1,atcol=2,fillby=cols) bmeans
report(atrow=1,atcol=3,fillby=cols) bstderrs
report(action=format,picture="*.###")
report(action=show)
*
set regime1 gstart gend = regime1/ndraws
graph(header="MCMC Probability of Low Variance Regime")
# regime1
*
@MCGraphIRF(model=MSSysRegModel,onlyshocks=||2*oilpos-1,2*oilpos||,$
    shocklabels=||"Regime 1","Regime 2"||,$
    varlabels=||"Cap Util","CPI","Oil Price"||)

```

Markov Switching VAR's

The example in Chapter 9 is a univariate autoregression and that in Chapter 10 is a vector autoregression. So why a separate chapter? If you recall, there were some difficulties in interpreting the results from those models when both coefficients and variances were allowed to switch. While the univariate regression did have an intercept (and process mean) that were different (as intended) in the two regimes, it turned out that a better interpretation of the model was that it had regimes with different *variances* instead. And the VAR model was mainly a one-time split of the sample in 1986 largely based upon changes (again) in the variance process.

While it's possible to restrict the model to a fixed covariance matrix, there remains the possibility that a vector autoregression with full coefficient vector switching will produce a split which is difficult to interpret. Also, the sheer number of coefficients can become overwhelming. Instead, it's common to use a much more restricted specification, keeping the lag coefficients fixed across regimes, allowing only the intercept (or the process mean) and possibly covariance matrix to switch.

The `@MSSysRegression` procedure is general enough to handle VAR's with varying intercepts but fixed lag coefficients by using the `NFIX` option. You just have to arrange the regressors so the **CONSTANT** is last. Hamilton (1989) chooses a different restriction: the (univariate) model takes the form:

$$y_t = \mu_t + z_t \quad (11.1)$$

where the process mean μ_t is a Markov switching process, while z_t is a stationary (non-switching) AR process. While apparently similar to an AR model with a switching intercept like the example from Chapter 9, it turns out to be quite a bit more complicated. In the terminology from page 5, this is the *Additive* form, while the switching intercept is the *Innovational*. It's probably a good idea to try both in any application, since it's not clear *a priori* which will work better. For instance, in the case of Hamilton's original data set, it turns out that the (simpler) shifting intercept fits slightly better. Note that this is the model that is used in Section 8.1 as an example of the sometimes fickle nature of MS models, as the estimates on a longer data range do not provide the same easily interpreted behavior.

If we expand (11.1), we get

$$y_t - \mu_t = \phi_1(y_{t-1} - \mu_{t-1}) + \dots + \phi_p(y_{t-p} - \mu_{t-p}) + u_t \quad (11.2)$$

By contrast, the innovational (switching intercept) form is:

$$y_t = \alpha_t + \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} + u_t$$

The complication in (11.2) is that the density function for y_t depends upon the current μ_t and on p of its lags. Thus, if we have M choices for each μ_t , there are M^{p+1} branches in the likelihood, each with a distinct value. All three estimation methods will need to take this into account. Note that estimation time goes up at roughly that same M^{p+1} factor since the evaluation of the likelihoods is a fairly big piece of the calculation.

The discussions in the previous chapters will handle all cases except the switching means model, so that will be the focus of this chapter. You can use the specialized sets of MS-VAR procedures for switching intercept and coefficient models as well, as it will be simpler to use when you have a VAR—it's just that the description of what those are doing has been covered in the earlier chapters.

11.1 Estimation

Most of the hard work is done inside the procedures described in 11.1.1. Maximum likelihood and EM are both handled with programs very similar to what we've seen with other types of regressions. MCMC is, however, definitely harder.

We're using μ_t to represent the value of the switching mean process at t , and will use $\mu(s)$ to represent the value for this when $S_t = s$. So we need to estimate the ϕ , μ and variance(s) or covariance matrices.

Both EM and MCMC need to estimate the lag coefficients ϕ and the means μ separately, each given the other. To estimate the means, we need to rearrange (11.2) to

$$y_t - \phi_1 y_{t-1} - \dots - \phi_p y_{t-p} = \mu_t - \phi_1 \mu_{t-1} - \dots - \phi_p \mu_{t-p} + u_t \quad (11.3)$$

Note that the left side of this is the same for all combinations of regimes. The right side will be a linear combination of the $\mu(s)$ where the multipliers will depend upon the precise combination of regimes $\{S_t, \dots, S_{t-p}\}$. Since both $\mu(s)$ will, in general, appear on the right side, the μ have to be estimated jointly.¹

Internally, the RATS procedures number the combinations of expanded regimes as $1, \dots, M^{p+1}$, with the longest lags varying most quickly; that is, with $M = 2$, they will be in the order $1, \dots, 1, 1; 1, \dots, 1, 2; 1, \dots, 2, 1; 1, \dots, 2, 2$, etc.

11.1.1 MSVARSETUP procedures

The specialized procedure **@MSVARSetup** sets up a switching autoregression or VAR and pulls in the necessary support files. This takes the form

¹This isn't true with switching intercepts, where each can be estimated separately.


```
@MSVARSetup( options )
# list of dependent variables
```

The option `LAGS=#` of VAR lags (with a default of 1) sets the number of lags. This sets up an autoregression or VAR with each equation having a `CONSTANT` and the lags of the dependent variables.

The options which control the switching are `REGIMES=#` of regimes, with a default of 2, and `SWITCH=[M]/I/MH/IH/C/CH`, which determines what switches—an `M` means switching mean (fixed lags), an `I` means switching intercepts (fixed lags), and `C` means intercept form with all coefficients switching. An `H` suffix means that the variances (covariance matrices) switch as well.

Because it's designed (mainly) for *vector* autoregressions, the parameters that are used are matrices, not scalars. These are:

- `MU` is a `VECTOR` of `VECTORS`, where the outer `VECTOR` numbers the regimes and the inner numbers the variables; that is `MU(s)(i)` is the mean (or intercept, it's used for both) for dependent variable `i` in regime `s`.
- `PHI` is a `VECTOR` of `RECT`, where the `VECTOR` numbers the lags. `PHI(l)(i,j)` is the lag `l` coefficient on variable `j` in equation `i`. This is used with fixed lag coefficient models.
- `PHIV` is a `RECT` of `RECT`. This is similar to `PHI`, except the outer matrix now has the first subscript numbering regimes, so `PHIV(s,l)` is the matrix of lag `l` coefficients in regime `s`. This is used with the `SWITCH=C` and `SWITCH=CH` forms.
- `SIGMA` is a `SYMMETRIC` covariance matrix, used with the fixed covariance matrix models.
- `SIGMAV` is a `VECTOR` of `SYMMETRIC`, where `SIGMAV(s)` is the covariance matrix in regime `s`. This is used with any of the "H" choices for the `SWITCH` option.

In addition, there are the standard `P` and `THETA` forms for the transition probabilities.

Because the working set of regimes for a MS-VAR can take two different forms, depending upon whether or not we use the switching mean variant, there are a whole set of specialized procedures included. For instance, the filter and smoothing for the switching means model has to work with the expanded regime. However, in the end, we only need the (marginal) probabilities of the current regime. So the procedure `@MSVARSmoothed` does the calculation of smoothed probabilities as required by the form of the model, but then marginalizes if necessary to return only the probabilities for S_t .

11.2 The example

We'll work with the original Hamilton model, which has switching means with four lags and two regimes. That makes an expanded regime vector with 32 components, which makes it slower than a three or four variable VAR with the simpler switching intercepts.

11.2.1 Maximum Likelihood

As usual, the hard work is done by the procedures. Note that this is where the warnings earlier about the (slow) speed of ML becomes most apparent. The combination of several variables and relatively long lags creates a very large parameter set with a substantial amount of calculation to evaluate the likelihood. The set up for estimation is:

```
@msvarsetup(lags=4,switch=m)
# g
@msvarparmset(parmset=varparms)
nonlin(parmset=msparms) p
```

The **@MSVARPARMSET** procedure generates a **PARMSET** for all the “VAR” variables needed given the options for the model. In this case, it will be **MU** (for the switching means), **PHI** (for the non-switching AR coefficients) and **SIGMA** (for the non-switching variance).

The next step gets a standard set of guess values.

```
@msvarinitial
compute gstart=%regstart(),gend=%regend()
```

The guess values are formed by running a (vector) autoregression to initialize the lag coefficients, and setting the regime means based upon the sample means and standard errors of the dependent variables—for two regimes, they are -1 and +1 standard deviations from the mean. For intercept models, the corresponding intercepts are solved out from the means. Note that regime 1 is the low mean for *all* variables. If you have a mix of variables where high values in some might likely associate with low values of others, you might want to try a different set of guess values. The **GUESSREGIMES** option on **@MSVARINITIAL** is helpful if you have at least some idea of where the regimes might be. For instance, the following would guess the means based upon the coding of the **RECESSQ** variable in the data set (+1 for recession, -1 for expansion):

```
dec series[int] guessregimes
gset guessregimes = %if(recessq==-1,2,1)
@msvarinitial(guessregimes=guessregimes)
```

The actual estimation is done with:

Table 11.1: Hamilton Model-Maximum Likelihood

MAXIMIZE - Estimation by BFGS					
Convergence in 22 Iterations. Final criterion was 0.0000083 <= 0.0000100					
Quarterly Data From 1952:02 To 1984:04					
Usable Observations		131			
Function Value		-181.2634			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	MU(1)(1)	-0.3588	0.2593	-1.3838	0.1664
2.	MU(2)(1)	1.1635	0.0841	13.8379	0.0000
3.	PHI(1)(1,1)	0.0135	0.1240	0.1087	0.9134
4.	PHI(2)(1,1)	-0.0575	0.1384	-0.4158	0.6776
5.	PHI(3)(1,1)	-0.2470	0.1094	-2.2586	0.0239
6.	PHI(4)(1,1)	-0.2129	0.1126	-1.8911	0.0586
7.	SIGMA(1,1)	0.5914	0.1095	5.4002	0.0000
8.	P(1,1)	0.7547	0.1043	7.2337	0.0000
9.	P(1,2)	0.0959	0.0380	2.5217	0.0117

```

frml msvarf = log(%MSVARProb(t))
maximize (parmset=varparms+msparms, $
  start=(pstar=%MSVARInit()), $
  reject=%MSVARInitTransition()==0.0, $
  pmethod=simplex, pitters=5, method=bfgs, iters=300) msvarf gstart gend

```

`pstar=%MSVARInit()` takes care of all the initialization required for evaluating the likelihood. The option `reject=%MSVARInitTransition()==0.0` makes sure the values for `P` are in range. If you use the `THETA` parameterization in `MSPARMS`, you don't need this, but need a different `START` option:

```

maximize (parmset=varparms+msparms, $
  start=(p=%mslogisticp(theta), pstar=%MSVARInit()), $
  pmethod=simplex, pitters=5, method=bfgs, iters=300) msvarf gstart gend

```

The output is in Table 11.1. As described above, `MU(i)` is technically a vector, to allow for use with a VAR, and `PHI(1)` is technically a matrix with dimensions $n \times n$, thus the `(1)` and `(1,1)` subscripting respectively. The coefficients show the rather nice interpretation that the “contraction” regime (1) has roughly a four quarter expected length ($1/(1 - .754)$), while the “expansion” regime has a mean quarter to quarter growth of 1.16% and expected length a bit more than ten ($1/.0959$). The full example is **11.1**.

11.2.2 EM

The E step here needs to smooth the expanded regime vector $\{S_t, \dots, S_{t-p}\}$. A generalized M step is used, with the values for μ and ϕ estimated separately (μ given the previous ϕ , ϕ given the recalculated μ).² Even though each of these is a type of linear regression, neither can be done using any standard regression. The estimator for ϕ is a probability-weighted regression, but at each t , a probability-weighted sum is required over all of combinations for $\{S_t, \dots, S_{t-p}\}$, since each generates a different set of dependent and explanatory variables. For a one-lag univariate model, the explicit formula is

$$\frac{\sum_{t=1}^T \sum_{i=1}^M \sum_{j=1}^M \hat{p}_{ij,t} (y_{t-1} - \mu(j)) (y_t - \mu(i))}{\sum_{t=1}^T \sum_{i=1}^M \sum_{j=1}^M \hat{p}_{ij,t} (y_{t-1} - \mu(j))^2} \quad (11.4)$$

where $\hat{p}_{ij,t}$ is the smoothed probability at t for the combination of $S_t = i$ and $S_{t-1} = j$. If variances are also switching, both sums would also need to be divided by $\sigma^2(i)$. With more lags, this is generalized to a matrix calculation.

The μ are estimated using the rearranged form (11.3). Again, the estimation is a probability-weighted regression, where the sums are across both t and the combination of regimes. For the regime tuple s_0, s_1, \dots, s_p , the explanatory variable in the regression on $\mu(k)$ is

$$(s_0 == k) - \sum_{j=1}^p \phi_j (s_j == k)$$

where the $==$ means 1 if true and 0 if false.

The M step for the transitions is exactly the same as it is for the simpler MS regression models. We need the smoothed probabilities of regime pairs (current and one lag), but as long as we have at least one lag we've had to compute at *least* that to estimate the regression coefficients. With more than one lag, we can just compute the marginals of the expanded regime probabilities for just the pair of current and lag and use those.

The first few steps are the same as for ML. Then the working code is:

```
@msvareEMgeneralsetup
do emits=1, 50
    @msvarestep gstart gend
    disp "Iteration" emits "Log Likelihood" %log1
end do emits
```

²Because of the multiplicative interaction, a full M step would require iterating over that combination to convergence, where the second and later iterations would largely be a waste of time, since the "problem" gets reset when new weights are computed.

This is similar to what we've seen before, but with a different set of procedures. Because the E step needs to smooth over $p + 1$ tuples of regimes rather than just pairs, both the "setup" and "emstep" procedures need to be different. You'll probably notice that the convergence of EM is slower (in terms of progress per iteration) than it is with the simpler model. This is partly because we're doing a generalized M and thus not making as much progress at each step as we would if we were doing a full M. And also the likelihoods are more dependent upon the transition probabilities and not just the probabilities of the regimes themselves, so the maximization problem changes more when P gets updated.

Again, we recommend that you do some BHHH iterations to polish the estimates and get standard errors. Note, however, that with a VAR, particularly one with switching coefficients, you could run out of data points to do BHHH. BHHH uses the inverse of the cross product of the gradient vectors to estimate the covariance matrix. But if the number of parameters in the complete model exceeds the number of data points, the matrix to be inverted is necessarily singular.

11.2.3 MCMC (Gibbs Sampling)

There are several complications in doing MCMC with a switching means VAR. With a standard regression (univariate or systems) model, the sampler for the regression coefficients can be done using an "off-the-shelf" linear regression simulator—it's just applied to the subsample formed by each regime in turn. That no longer works with either the ϕ or with the μ because the likelihoods for each involve the regimes at the lags as well. Also, the μ should, ideally, be estimated jointly, since that's the form of (11.3). This being Gibbs sampling, you *could* do them sequentially, but that might tend to create a greater correlation in the chain.

Sampling Variances

Sampling the variances (or covariance matrix) is no different from the previous two chapters, once you have the residuals. Those can be computed using the procedure `@MSVARResids`. The `@MSVARSETUP` procedure file defines a special `VECT[SERIES]` for the residuals named `MSVARU`.

```
@MSVARResids(regime=MSRegime) MSVARU gstart gend
```

Note that the output from `@MSVARResids` is a `VECT[SERIES]`, so if you're doing a univariate model, you'll have to use `MSVARU(1)` as the series of residuals. Once you're done drawing the variance(s), you need to execute:

```
compute %MSVARInitVariances()
```

which makes sure all the variance information is in the proper locations for work in the remaining steps.

Sampling Means

Given the other parameters, the likelihood for the means is based upon (11.3). A cross-product matrix with the (variance-weighted) sample information for this can be computed using:

```
@MSVARMuCMOM gstart gend xxem
```

The output from this (here called **XXEM**) takes the proper form for use with the procedure **%RANMVPOSTCMOM** where the data precision is 1.0, since the variances have already been incorporated into the crossproduct matrix. The “stacked” vector of means (or intercepts, the same procedure will handle both) is sampled with:

```
compute mustack=%ranmvpostcmom(xxem,1.0,hmu,bmu)
```

In our example, we'll use a symmetrical prior for the means and label-switch based upon the sampled values. The mean (**BMU**) is a vector of zeros and the precision (**HMU**) is a very loose diagonal matrix with .25's (meaning standard deviation 2.0) on the diagonal.

MUSTACK is just a **VECTOR** with # of regimes \times # of variables entries, which is not the form used elsewhere. The procedure **@MSVARMuUnstack** takes it and fills in the **VECT[VECT]** that is used for **MU**.

```
@MSVARMuUnstack mustack mu
```

Sampling Lag Coefficients

The likelihood for doing inference on the lag coefficients uses the form (11.2), where the (regime-dependent) means are treated as given. Again, there's a separate procedure which summarizes the (variance-weighted) data information in a cross product matrix:

```
@MSVARPhiCMOM gstart gend xxem
```

Note that **XXEM** will have quite different dimensions as it did when sampling means—the number of lag coefficients could be quite large. In this example, we're using a mean (**BPHI**) vector of zeros with the precision (**HPHI**) as a diagonal matrix with values of 4.0 (standard deviations .5). As with the means, the output is a long **VECTOR** which needs to be re-packaged into the proper form; the procedure **MSVARPhiUnstack** does that.

```
compute phistack=%ranmvpostcmom(xxem,1.0,hphi,bphi)
@MSVARPhiUnstack phistack phi
```

Relabeling

In this application, we want to label the regimes based upon the mean, from low to high. The following extracts the regime-specific means into a `VECTOR`, and computes the indexing array (`SWAPS`) to order that from low to high.

```
ewise muv(i)=mu(i)(1)
compute swaps=%index(muv)
```

If the means are in the correct order, `SWAPS` will have elements 1, 2 in order. (And similarly for more than two regimes). The `%MSSwapCheck` function tests to see if we need to relabel—it returns 1 if we do. If we have to relabel, we report that (getting a lot of these warnings is a bad sign) and uses the `@MSVARRelabel` procedure to actually execute the swaps of the elements of all the components of the model which depend upon regimes.

```
if %MSSwapCheck(swaps) {
    disp "Draw" draw "Executing swap"
    @MSVARRelabel swaps
}
```

Sampling Regimes

As with EM, you need to forward filter using the *expanded* set of regimes, so the `@MSFilterStep` procedure used in the previous chapters won't work. Instead, there's a separate procedure `@MSVARFilter` which does the forward filter on whatever collection of regimes is required given the settings for the model. Similarly, the backwards sampling requires a specialized routine as well which samples the expanded regimes—this is `@MSVARSample`. When you sample the final period, you get $\{S_T, \dots, S_{T-p}\}$, the next will get $\{S_{T-1}, \dots, S_{T-p-1}\}$, backing up until the start of the sample finally gives $\{S_1, S_0, \dots, S_{-p+1}\}$, where the “pre-sample” regimes S_0 to S_{-p+1} are needed to compute the likelihood for the early values of t .

The following is the process followed here:

```
@MSVARFilter gstart gend
:redrawregimes
@MSVARSample(counts=tcounts) gstart gend MSRegime
if %minvalue(tcounts)<minsize {
    disp "Draw" draw "Redrawing regimes with regime of size" $
        %minvalue(tcounts)
    goto redrawregimes
}
```

If the regime sampling procedure leaves a regime with below `MINSIZE` members, that sampling is rejected and a new one drawn. In this application (with a relatively small data set and one regime that's expected to be on the small side), `MINSIZE` is 5. With a larger data set, you would probably want this to be higher.

Table 11.2: Hamilton Model-Gibbs Sampling

	Mean	Std. Dev.
MU(1)(1)	-0.252	0.390
MU(2)(1)	1.103	0.165
PHI(1)(1,1)	0.160	0.149
PHI(2)(1,1)	0.028	0.139
PHI(3)(1,1)	-0.163	0.118
PHI(4)(1,1)	-0.122	0.111
SIGMA(1,1)	0.726	0.164
P(1,1)	0.737	0.104
P(2,1)	0.263	0.104
P(1,2)	0.128	0.066
P(2,2)	0.872	0.066

Sampling Transition Probabilities

This is exactly the same as in the previous chapters.

Execution Time

The switching means model is again *many* times slower than the switching intercepts model. **11.3** does a “test” number of draws (500 with 200 burn-in). For a final “production” run, you would probably want to up those by a factor of at least 10.

Results

Table 11.2 shows the mean and standard deviation from a run with 5000 saved draws. (Note that this depends upon simulations, so won't be exactly the same when re-run). This is similar to the ML estimates, but the means aren't quite as widely separated and have higher standard errors. If we look at the graphs of the densities of the simulated means (Figure 11.1), we can see that the two densities skew towards each other. There's a “gray-zone” of values around .5% quarter-to-quarter growth which isn't clearly either expansion or recession. This wouldn't be as obvious from looking at ML alone.

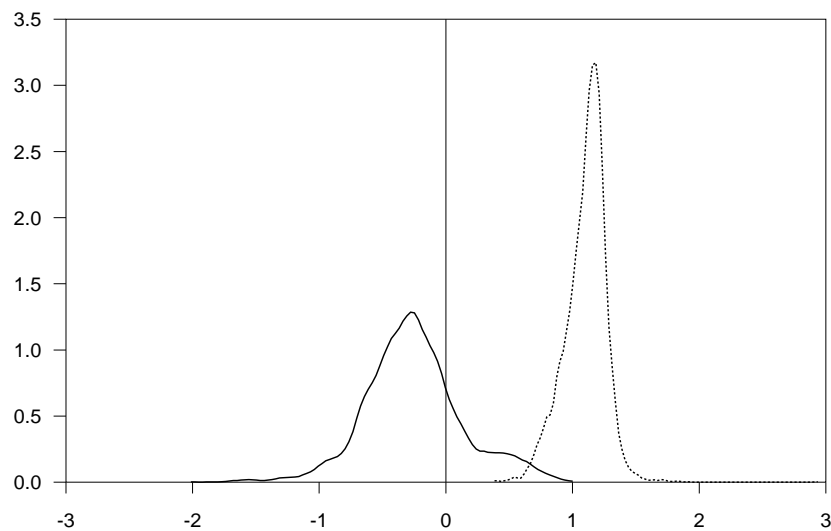


Figure 11.1: Densities for Means from Gibbs Sampling

Example 11.1 Hamilton Model: ML Estimation

This is an example of a maximum likelihood estimation of a (univariate) MS-AR with switching means (only). The technical details are in Section 11.2.1.

```
cal(q) 1951:1
open data gnpdata.prn
data(format=prn,org=columns) 1951:1 1984:4
*
set g = 100*log(gnp/gnp{1})
graph(header="GNP growth")
# g
*
* Set up a mean-switching model with just the one variable and four
* lags.
*
@msvarsetup(lags=4,switch=m)
# g
@msvarparmset(parmset=varparms)
nonlin(parmset=msparms) p
*
@msvarinitial
*
* Pick up the maximum range for the model.
*
compute gstart=%regstart(),gend=%regend()
*
* Estimate the model by maximum likelihood.
*
frml msvarf = log(%MSVARProb(t))
maximize(parmset=varparms+msparms,$
    start=(pstar=%MSVARInit()),$
    reject=%MSVARInitTransition()==0.0,$
```

```

    pmethod=simplex,piters=5,method=bfgs,itors=300) msvarf gstart gend
*
* Compute smoothed estimates of the regimes.
*
@msvarsmoothed gstart gend psmooth
set pcontract gstart gend = psmooth(t)(1)
*
* To create the shading marking the recessions, create a dummy series
* which is 1 when the recessq series is 1, and 0 otherwise. (recessq
* is 1 for NBER recessions and -1 for expansions).
*
set contract = recessq==1
*
spgraph(vfields=2)
graph(header="Quarterly Growth Rate of US GNP",shade=contract)
# g %regstart() %regend()
graph(style=polygon,shade=contract,$
    header="Probability of Economy Being in Contraction")
# pcontract %regstart() %regend()
spgraph(done)

```

Example 11.2 Hamilton Model: EM Estimation

This is an example of EM estimation of a MS-AR (just univariate) with switching means (only). The technical details are in Section 11.2.2.

```
cal(q) 1951:1
open data gnpdata.prn
data(format=prn,org=columns) 1951:1 1984:4
*
set g = 100*log(gnp/gnp{1})
graph(header="GNP growth")
# g
*
* Set up a mean-switching model with just the one variable and four
* lags.
*
@msvarsetup(lags=4,switch=m)
# g
@msvarinitial
compute gstart=%regstart(),gend=%regend()
*
@msvaremgeneralsetup
do emits=1,50
    @msvaremstep gstart gend
    disp "Iteration" emits "Log Likelihood" %log1
end do emits
*
* Polish estimates with 10 iterations of BHHH. Set up the parameter
* sets and the initialize the regime probability matrices (required
* for ML, but not EM, which uses its own).
*
@msvarparmset(parmset=varparms)
nonlin(parmset=msparms) p
*
* The "P" matrix needs to be truncated to remove the bottom row since
* that's the form used for ML estimation.
*
compute p=%xsubmat(p,1,nregimes-1,1,nregimes)
*
frml msvarf = log(%MSVARProb(t))
maximize(parmset=varparms+msparms,$
    start=(pstar=%MSVARInit()),$
    reject=%MSVARInitTransition()==0.0,$
    method=bhhh,itors=10) msvarf gstart gend
```

Example 11.3 Hamilton Model: MCMC Estimation

This is an example of MCMC analysis of a MS-AR (just univariate) with switching means (only). The technical details are in Section 11.2.3. Note that this is written to allow for switching variances as well (with a hierarchical prior), even though that part isn't used in this application.

```
cal(q) 1951:1
open data gnpdata.prn
data(format=prn,org=columns) 1951:1 1984:4
*
set g = 100*log(gnp/gnp{1})
*
* Set up a mean-switching model with just the one variable and
* four lags.
*
@msvarsetup(lags=4,switch=m)
# g
@msvarinitial
compute gstart=%regstart(),gend=%regend()
*
* This is a specific initializer to allow for backwards
* sampling in an MSVAR.
*
compute p=%mspexpand(p)
@msvarfiltersetup
*
* Prior for transitions. Weak Dirichlet priors with preference
* for staying in a given regime.
*
dec vect[vect] gprior(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
dec vect tcounts(nregimes)
*
* This is the minimum size allowed for a regime.
*
compute minsize=5
*
* Guess the initial regimes based upon the growth rate. (This
* needs to map to 1 and 2, hence the 1+..).
*
gset MSRegime = 1+fix(g>0.5)
*
* Hierarchical prior for sigmas
* Uninformative prior on the common component.
*
compute nucommon=0.0
compute s2common=0.0
*
* Priors for the relative variances (if needed)
*
dec vect nuprior(nregimes)
```

[illegible]

```

*
* Draw the common variance factor given the relative variances
* and regimes.
*
sstats gstart gend $
      MSVARU(1)^2*scommon/sigmav(MSRegime(t))(1,1)>>sumsq
compute scommon=(sumsq+nucommon*s2common)/$
      %ranchisqr(%nobs+nucommon)

*
* Draw the relative variances, given the common variances and
* the regimes.
*
do s=1,nregimes
  sstats(smpl=MSRegime==s) gstart gend $
    MSVARU(1)^2/scommon>>sumsq
  compute sigmav(s)(1,1)=scommon*(sumsq+nuprior(s))/$
    %ranchisqr(%nobs+nuprior(s))
end do s
}
compute %MSVARInitVariances()
*
* Draw mu given phi
*
@MSVARMuCMOM gstart gend xxem
compute mustack=%ranmvpostcmom(xxem,1.0,hmu,bmu)
@MSVARMuUnstack mustack mu
*
* Draw phi given mu
*
@MSVARPhiCMOM gstart gend xxem
compute phistack=%ranmvpostcmom(xxem,1.0,hphi,bphi)
@MSVARPhiUnstack phistack phi
*
* Relabel if necessary
*
ewise muv(i)=mu(i)(1)
compute swaps=%index(muv)
if %MSSwapCheck(swaps) {
  disp "Draw" draw "Executing swap"
  @MSVARRelabel swaps
}
@MSVARFilter gstart gend
:redrawregimes
@MSVARSample(counts=tcnts) gstart gend MSRegime
if %minvalue(tcnts)<minsize {
  disp "Draw" draw "Redrawing regimes with regime of size" $
    %minvalue(tcnts)
  goto redrawregimes
}
*
* Draw p's
*
@MSDrawP(prior=gprior) gstart gend p
infobox(current=draw)

```

```

    if draw>0 {
      *
      * Do the bookkeeping
      *
      set regime1 gstart gend = regime1+(MSRegime==1)
      compute bgibbs(draw)=%parmspeek(allparms)
    }
  end do draw
  infobox(action=remove)
  *
  @mcmcpostproc(ndraws=ndraws,mean=bmeans,stderrs=bstderrs) bgibbs
  *
  report(action=define)
  report(atrow=1,atcol=1,fillby=cols) %parmslabels(allparms)
  report(atrow=1,atcol=2,fillby=cols) bmeans
  report(atrow=1,atcol=3,fillby=cols) bstderrs
  report(action=format,picture="*.###")
  report(action=show)
  *
  @nbercycles(down=recess)
  set regime1 gstart gend = regime1/ndraws
  graph(header="MCMC Probability of Low Mean Regime",shading=recess)
  # regime1
  set mulowsample 1 ndraws = bgibbs(t)(1)
  set muhighsample 1 ndraws = bgibbs(t)(2)
  density(grid=automatic,maxgrid=100,smooth=1.5) mulowsample $
    1 ndraws xmulow fmulow
  density(grid=automatic,maxgrid=100,smooth=1.5) muhighsample $
    1 ndraws xmuhigh fmuhigh
  scatter(style=lines,header="MCMC Densities of Means") 2
  # xmulow fmulow
  # xmuhigh fmuhigh

```

Markov Switching State-Space Models

Every textbook uses slightly different notation to describe a state-space model. We'll stick here with the notation used in the RATS manual. A state-space model takes the form:

$$\mathbf{X}_t = \mathbf{A}_t \mathbf{X}_{t-1} + \mathbf{Z}_t + \mathbf{F}_t \mathbf{W}_t \quad (12.1)$$

$$\mathbf{Y}_t = \mu_t + \mathbf{C}'_t \mathbf{X}_t + \mathbf{V}_t \quad (12.2)$$

The \mathbf{X} 's are the unobservable *state variables*. The \mathbf{Y} 's are the observable data. The first equation is the *state* or *transition equation*, the second is the *observation* or *measurement equation*. The \mathbf{Z} 's, if present, are exogenous variables in the evolution of the states. The \mathbf{W} 's are shocks to the states; the \mathbf{F} matrix has the loadings from those shocks to states, which allows for there to be fewer shocks than states (which will generally be the case). The μ are any components in the description of the observable which depend upon exogenous variables (such as a mean in the form of a linear regression). The \mathbf{V} 's are measurement errors. The \mathbf{W} 's and \mathbf{V} 's are assumed to be mean zero, typically Normally distributed, independent across time and independent of each other at time t as well.¹

There are many ways to incorporate some type of switching process into a state-space model. For instance, the μ_t might be a switching mean or regression process, with the state-space model handling a (fixed) serially correlated error process. Or, perhaps, the variance of the state shocks \mathbf{W}_t or of the measurement error process \mathbf{V}_t might switch.

Regardless, with few exceptions, these are more complicated to handle than the regression models in the previous three chapters. This is due to the fact that (in most models) \mathbf{X}_t is at least partially unobservable. If you have an M branch process, \mathbf{X}_1 will have M branches. \mathbf{X}_2 will have M branches for each possible density for \mathbf{X}_1 , and thus will have M^2 possibilities. Continuing, we find that \mathbf{X}_T has M^T possibly densities, which is too large a set to handle unless T is quite small. Thus, exact maximum likelihood won't work, and neither will EM, both of which require an ability to list all the possible branches. We're left with two options: an approximate ML, which somehow collapses the list of branches to

¹Some descriptions of state-space models allow for \mathbf{W} 's and \mathbf{V} 's to be correlated, which complicates the formulas, and doesn't actually make the model more general because a \mathbf{V} can always be moved into \mathbf{W} by adding another (trivial) state variable.

a more manageable number, or MCMC, which avoids the problem by sampling regime combinations rather than attempting to enumerate them.

12.1 The Examples

We'll look at two examples. The first is from Lam (1990). This is similar to the Hamilton model from Chapter 11, but models log GDP directly rather than its growth rate:

$$y_t = \tau_t + x_t \quad (12.3)$$

$$\tau_t = \tau_{t-1} + \delta_{S_t} \quad (12.4)$$

where τ_t is the unobservable trend process, subject to a Markov switching drift process δ_{S_t} , and x_t is a low-order non-switching AR process. Because the stationary shocks in the Hamilton model are to the growth rates, there are no purely transient shocks to the GDP process itself. In Lam's model, there are both persistent *and* transient changes to the level of GDP, with the former done through a switching process for the growth rate.

The states will be τ_t and x_t along with any lags of x required for creating a proper model. If x follows an AR(2) process, the state equation will be:

$$\mathbf{X}_t \equiv \begin{bmatrix} x_t \\ x_{t-1} \\ \tau_t \end{bmatrix} = \begin{bmatrix} \phi_1 & \phi_2 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ x_{t-2} \\ \tau_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \delta_{S_t} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} w_t$$

with measurement equation:

$$y_t = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \mathbf{X}_t + 0$$

The unknown parameters are $\phi_1, \phi_2, \text{var}(w_t)$, and the parameters governing δ_{S_t} . If this has a two-state Markov chain, there will be the drift parameters δ_1 and δ_2 and the two parameters governing the transition probabilities. In this model, the switch is in the Z component of the state-space model. An alternative representation, which is what we'll actually use, is to first difference (12.3) to get:

$$y_t - y_{t-1} = \delta_{S_t} + x_t - x_{t-1}$$

The observable is now the first difference of y , and the switch is now in the μ component in the observation equation. This gives us the slightly simpler:

$$\mathbf{X}_t \equiv \begin{bmatrix} x_t \\ x_{t-1} \end{bmatrix} = \begin{bmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ x_{t-2} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} w_t$$

$$y_t - y_{t-1} = \delta_{S_t} + \begin{bmatrix} 1 & -1 \end{bmatrix} \mathbf{X}_t$$

For this model, exact maximum likelihood is *possible*, though very time-consuming. If we unwind the definition for the trend, we get:

$$\tau_t = \tau_0 + \sum_{s=1}^t \delta(S_s) = \tau_0 + \delta(1) \sum_{s=1}^t (S_s = 1) + \delta(2) \sum_{s=1}^t (S_s = 2) \quad (12.5)$$

Since it is a straight (unweighted) sum of the values of δ , it depends only upon the number of occurrences of each regime, and not the specific sequence. Thus, the likelihood at t has t branches, rather than M^t . Because the recent past matters for the AR part, it turns out that you need to keep track of $M^{q+1}t$ possible combinations.

The second example is a time-varying parameters regression with the error process having Markov switching variances. The regression is a money demand function on the growth rate with

$$\Delta M_t = \beta_{0t} + \beta_{1t}\Delta i_{t-1} + \beta_{2t}inf_{t-1} + \beta_{3t}surp_{t-1} + \beta_{4t}\Delta M_{t-1} + e_t$$

where M_t is log money, i_t is the interest rate, inf_t is the inflation rate, $surp_t$ is the detrended full-employment budget surplus. The β are assumed to follow independent random walks, so we have the state-space representation:

$$\mathbf{X}_t \equiv \begin{bmatrix} \beta_{0t} \\ \beta_{1t} \\ \beta_{2t} \\ \beta_{3t} \\ \beta_{4t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_{0t-1} \\ \beta_{1t-1} \\ \beta_{2t-1} \\ \beta_{3t-1} \\ \beta_{4t-1} \end{bmatrix} + \begin{bmatrix} w_{0t} \\ w_{1t} \\ w_{2t} \\ w_{3t} \\ w_{4t} \end{bmatrix}$$

$$y_t = [1, \Delta i_{t-1}, inf_{t-1}, surp_{t-1}, \Delta M_{t-1}] \mathbf{X}_t + e_t$$

What will switch here is the variance of e_t . The parameters to be estimated are the variances of the w 's, the different regime variances for e and the parameters governing the Markov chain.

12.2 The Kim Filter

The Kim (1994) filter is a specific approximation to the likelihood for state-space models. In a non-switching state-space model, the Kalman filter is used to evaluate the likelihood. This summarizes the information about \mathbf{X}_{t-1} in the subsample through time $t-1$ with its mean (denoted $\mathbf{X}_{t-1|t-1}$) and covariance matrix ($\Sigma_{t-1|t-1}$). The first step in the Kalman filter for period t is to *predict* the mean and covariance matrix of \mathbf{X}_t using (12.1), and, with (12.2), this is used to predict the density for the observable \mathbf{Y}_t and the joint density for \mathbf{Y}_t and \mathbf{X}_t .

The Kim filter uses the same type of calculation, but instead of summarizing the data information through $t-1$ with just the one mean and one covariance matrix, it has a mean and covariance matrix for each regime. Each sequence of regimes through $t-1$ is assumed to be adequately summarized by the one at $t-1$. In addition, there is a probability distribution across the regimes at $t-1$ (denoted \hat{p}_{t-1}). So entering the update calculation for time t , there are M densities at $t-1$. For each of those, there will now be M predictions for the density for \mathbf{Y}_t for a total of M^2 predictions for \mathbf{Y}_t . We will use the notation $f(i, j)$ for the predicted density (not logged) for \mathbf{Y}_t with $S_t = i$ and $S_{t-1} = j$.

The Markov Chain gives us the joint probability of i, j given the probabilities of the regimes at $t - 1$ as $p(i, j)\hat{p}_{t-1}(j)$. If there had been no approximations along the way, the likelihood of Y_t given data through $t - 1$ would be

$$\sum_{i=1}^M \sum_{j=1}^M f(i, j)p(i, j)\hat{p}_{t-1}(j) \quad (12.6)$$

and the log of this is the value of the log likelihood element at t for the Kim filter. Bayes rule gives us the updated probabilities of the i, j combinations as:

$$\hat{p}_t(i, j) = \frac{f(i, j)p(i, j)\hat{p}_{t-1}(j)}{\sum_{i=1}^M \sum_{j=1}^M f(i, j)p(i, j)\hat{p}_{t-1}(j)} \quad (12.7)$$

The standard Kalman filter calculations will give us an updated mean and covariance matrix for X_t for each combination of i, j , which we'll call $X_{t|t}(i, j)$ and $\Sigma_{t|t}(i, j)$. We now need to collapse those to a summary just for the time t regime i by aggregating out the time $t - 1$ regimes (the j). The mean is simple—it's just the probability-weighted average:

$$X_{t|t}(i) = \frac{\sum_{j=1}^M \hat{p}_t(i, j)X_{t|t}(i, j)}{\sum_{j=1}^M \hat{p}_t(i, j)} \quad (12.8)$$

The covariance matrix is the probability-weighted average of the covariance matrices, plus the bias-squared term for the difference between the regime i, j mean and the marginal mean for i :

$$\Sigma_{t|t}(i) = \frac{\sum_{j=1}^M \hat{p}_t(i, j) \left\{ \Sigma_{t|t}(i, j) + (X_{t|t}(i) - X_{t|t}(i, j)) (X_{t|t}(i) - X_{t|t}(i, j))' \right\}}{\sum_{j=1}^M \hat{p}_t(i, j)} \quad (12.9)$$

The part of this calculation which will be specific to a model and form of switching is the calculation of the $f(i, j)$, $X_{t|t}(i, j)$ and $\Sigma_{t|t}(i, j)$. These are all standard Kalman filter calculations, but need to be adjusted to allow for whatever switches, so the Kalman filtering steps have to be done manually.

The examples both define a (model-specific) function called `KimFilter` which does a single step in the Kim filter. Most of this is common to all such applications. First, the following need to be defined:

```
dec vect[vect] xstar(nregimes)
dec vect[symm] sstar(nregimes)
```

`XSTAR` and `SSTAR` are the collapsed means and variances of the state vector, indexed by the regime. They get replaced at the end of each step through

the filter. The VECTOR PSTAR is also used; that's defined as part of the standard Markov procedures and keeps the filtered probability distribution for the regimes, as it has before.

Within the function, XWORK and SWORK are the predicted means and covariance matrices $X_{t|t}(i, j)$ and $\Sigma_{t|t}(i, j)$. FWORK is used for $f(i, j)$. The model-specific calculations will be in the first double loop over I and J. This is for the Lam model, in the first-differenced form. The line in upper case is the one that has an adjustment for the switch.

```
function KimFilter time
type integer    time
*
local integer    i j
local real       yerr likely
local symm       vhat
local rect       gain
local rect       phat(nregimes,nregimes)
local rect       fwork(nregimes,nregimes)
local rect[vector] xwork(nregimes,nregimes)
local rect[symm]  swork(nregimes,nregimes)
*
do i=1,nregimes
  do j=1,nregimes
    *
    * Do the SSM predictive step
    *
    compute xwork(i,j)=a*xstar(j)
    compute swork(i,j)=a*sstar(j)*tr(a)+sw
    *
    * Do the prediction error for y under state i, and
    * compute the density function for the prediction error.
    *
    compute YERR=G(TIME)-(%DOT(C,XWORK(I,J))+DELTA(I))
    compute vhat=c*swork(i,j)*tr(c)+sv
    compute gain=swork(i,j)*tr(c)*inv(vhat)
    compute fwork(i,j)=exp(%logdensity(vhat,yerr))
    *
    * Do the SSM update step
    *
    compute xwork(i,j)=xwork(i,j)+gain*yerr
    compute swork(i,j)=swork(i,j)-gain*c*swork(i,j)
  end do j
end do i
*
* Compute the Hamilton filter likelihood
*
compute likely=0.0
do i=1,nregimes
  do j=1,nregimes
    compute phat(i,j)=p(i,j)*pstar(j)*fwork(i,j)
    compute likely=likely+phat(i,j)
  end do j
end do i
```

```

end do i
*
* Compute the updated probabilities of the regime combinations.
*
compute phat=phat/likely
compute pstar=%sumr(phat)
compute pt_t(time)=pstar
*
* Collapse the SSM matrices down to one per regime.
*
compute xstates(time)=%zeros(ndlm,1)
do i=1,nregimes
  compute xstar(i)=%zeros(ndlm,1)
  do j=1,nregimes
    compute xstar(i)=xstar(i)+xwork(i,j)*phat(i,j)/pstar(i)
  end do j
  compute sstar(i)=%zeros(ndlm,ndlm)
  do j=1,nregimes
    compute sstar(i)=sstar(i)+phat(i,j)/pstar(i)*$
      (swork(i,j)+%outerxx(xstar(i)-xwork(i,j)))
  end do j
  *
  * This is the overall best estimate of the filtered state.
  *
  compute xstates(time)=xstates(time)+xstar(i)*pstar(i)
end do i
compute KimFilter=likely
end

```

It's important to note that the Kim filter is *not* a relatively innocuous approximation—there are no results which show that it gives consistent estimates for the parameters of the actual model, and how far off it is in practice isn't really known. What would be expected is that it will bias the estimates in favor of parameters for which the approximation works reasonably well, which one would speculate would be models dominated by a persistent regime.

12.2.1 Lam Model by Kim Filter

The following is the special setup code for using the Kim filter for the Lam switching model. The number of states in the differenced form is just the number of AR lags in the noise term. To use the differenced form of the model, this needs to be at least two, since you need both x_t and x_{t-1} in the measurement equation. If you actually wanted only one lag, you could set this up with two states, and just leave the 1,2 element of the A matrix as zero. We'll call the size of the state vector NDLM.

```

compute q=2
dec vect phi(q)
compute ndlm=q

```

The **A** matrix will have everything below the first row fixed as the standard lag-shifting submatrix in this type of transition matrix. We'll set that up with the first row as zeros (for now). Since the first row depends upon parameters which we need to estimate, we'll need to patch that over as part of a function evaluation.

```
dec rect a(ndlm,ndlm)
ewise a(i,j)=(i==j+1)
```

We know the first two elements of **C** will be 1 and -1 and everything else will be zeros. This is written to handle any lags above two automatically:

```
dec rect c(ndlm,1)
compute c=||1.0,-1.0||~%zeros(1,ndlm-2)
```

The **F** matrix will always just be a unit vector for the first element with the proper size:

```
dec rect f(ndlm,1)
compute f=%unitv(ndlm,1)
dec symm sw(ndlm,ndlm)
compute sv=0.0
```

If this were a standard state-space model for use with the **DLM** instruction, the **SW** matrix would be just 1×1 since there's just the one shock—**DLM** automatically computes the rank-one covariance matrix $F\Sigma_w F'$ for the shock to the states. Since we have to do the filtering calculations ourselves, we'll just do the calculation of the full-size matrix one time at the start of each function evaluation. There's no error in the measurement equation, so **SV** is 0.

The next sets up the shifting part of the model, which are the regime-specific means for the change. For estimation purposes, we'll use guess values which go from large to small, since that was the ordering used in the original paper.

```
dec vect delta(nregimes)
```

The final piece of this is a vector of initial (pre-sample) values for the x component. This is one of several ways to handle the pre-sample states. If this were a model in which all the state matrices and variances were time-invariant, the most straightforward way to handle this would be to use the unconditional mean and covariance matrix (done with the **PRESAMPLE=ERGODIC** option on **DLM**). With switching models, by definition there are some input matrices which aren't time-invariant, so there isn't an obvious choice for the pre-sample distribution. Plus, we need a mean and covariance matrix *for each regime*. In the calculations for this example, the pre-sample state is added to the parameter set, with a single vector used for each regime, with a zero covariance matrix. The hope is that the estimation won't be very sensitive to the method of handling the pre-sample.

```
dec vect x0(ndlm)
compute x0=%zeros(ndlm,1)
```

The free parameters are divided into three parameter sets, one for the state-space parameters (the means, the variance in the evolution of x and the autoregressive coefficients), one for the transition (here modeled in logistic form) and finally the initial conditions:

```
nonlin(parmset=dlmparms) delta sigsq phi
nonlin(parmset=msparms) theta
nonlin(parmset=initparms) x0
```

There's quite a bit of initialization code required at the start of each function evaluation. Some of this is standard for state-space models, some is standard for Markov switching models. Both are packaged into a function called `DLMStart`.

The standard state-space initialization code stuffs the current settings for `PHI` into the top row of the `A` matrix, and expands the `SW` matrix from `F` and the variance parameter:

```
compute %psubmat(a,1,1,tr(phi))
compute sw=f*tr(f)*sigsq
```

The standard Markov switching initialization has generally been done as part of a `START` option on `MAXIMIZE`. This transforms the `THETA` into a transition matrix, computes the stationary distribution for the regimes and then expands the transition matrix to full size, which will make it easier to use.

```
compute p=%MSLogisticP(theta)
compute pstar=%mcergodic(p)
compute p=%mspexpand(p)
```

The one step that is new to this type of model is initializing the `XSTAR` and `SSTAR` matrices. With the handling of the initial conditions described above, this copies the `X0` out of the parameter set into each component of `XSTAR` and zeros out each component of `SSTAR`.

```
ewise xstar(i)=x0
ewise sstar(i)=%zeros(ndlm,ndlm)
```

The actual estimation is done with:

```
frml kimf = kf=KimFilter(t),log(kf)
@MSFilterInit
gset xstates = %zeros(ndlm,1)
maximize(parmset=msparms+dlmparms+initparms,start=DLMStart(),$
    reject=sigsq<=0.0,method=bfgs) kimf 1952:2 1984:4
```

This is similar to what we've seen before, as the details are “hidden” by the `KimFilter` function. The full code is Example 12.1.

12.2.2 Time-Varying Parameters Model by Kim Filter

Time-varying parameters models create a number of special problems even without any form of switching in the underlying process. The first is that the model for the states has k unit roots, which means that the likelihood really isn't properly defined until you've observed at least k data points. For a model without switching, the **DLM** instruction can handle this with either the option `PRESAMPLE=DIFFUSE` or `PRESAMPLE=ERGODIC`.² These use a specialized double calculation for the first k data points to keep separate track of the “infinite” (due to unit roots) and finite parts of the estimates. With each data point from 1 to k , the rank of the infinite matrix goes down one until it is zeroed out. This calculation is based upon a formal limit as the initial variances on the coefficients goes to ∞ .

While it would be possible to do the same thing in a switching context, it would require doing the parallel calculations for each of the M^2 branches required for the Kim filter, and extending the Kim reduction step to the partially infinite, partially finite matrices. The simpler alternative is to approximate this by using large finite variances (and zero means) for the pre-sample coefficients, then leaving out of the working log likelihood at least the first k data points. You just need to be somewhat careful about how you choose the “large” values—too large and you can have a loss of precision in the calculations, since you end up subtracting two large numbers (several times) with the end result being a small number.

It's also quite possible for the variance of the drift on any one coefficient (and sometimes on all of them) to be optimally zero; that is, a fixed coefficient model has a higher likelihood than any model that allows for drifting coefficients. In general, a drifting coefficients model will produce smaller residuals, but at the cost of a higher predictive variance, which gets penalized in the likelihood. It's thus necessary to parameterize the variance in a way that keeps it in the range $[0, \infty)$, such as by estimating in standard deviation form.

The parameters to be estimated in the TVP model are the variances on the five coefficient drifts, the variances in the two branches for the switching process and the parameters in the transition matrix. The variances are all put into standard deviation form and given guess values with:

```
dec vect sigmae(nregimes)
compute sigmae(1)=.25*sqrt(%seesq), sigmae(2)=1.5*sqrt(%seesq)
compute sigmav=.01*%stderrs
*
nonlin(parmset=dlmparms) sigmae sigmav
```

²They'll be equivalent here because all the roots are unit roots.

The `SIGMAE` are the regression error standard deviations—one for each regime. `SIGMAV` is a `VECTOR` with one value per regressors—they're initialized at .01 times the corresponding standard error from a fixed coefficient regression.

For the state-space model, `A` is just the identity, and the `C` matrix changes from data point to data point and is just the current set of regressors from the equation of interest. The model-specific part of the `KimFilter` function is:

```

COMPUTE C=TR(%EQNXVECTOR(MDEQ, TIME))
*
compute flhat(time)=0.0, f2hat(time)=0.0
do i=1,nregimes
  do j=1,nregimes
    *
    * Do the SSM predictive step. In this application A is
    * the identity, so the calculations simplify quite a bit.
    *
    compute xwork(i,j)=xstar(j)
    compute swork(i,j)=sstar(j)+sw

    *
    * Do the prediction error and variance for y under
    * regime i. The predictive variance is the only part of
    * this that depends upon the regime. Compute the
    * density function for the prediction error.
    *
    COMPUTE YERR=M1GR(TIME)-%DOT(C,XWORK(I,J))
    COMPUTE VHAT=C*SWORK(I,J)*TR(C)+SIGMAE(I)^2

    *
    * Do the decomposition of vhat into its components and
    * add probability-weighted values to the sums across
    * (i,j)
    *
    compute flhat(time)=flhat(time)+$
      %scalar(c*swork(i,j)*tr(c))*p(i,j)*pstar(j)
    compute f2hat(time)=f2hat(time)+sigmae(i)^2*p(i,j)*pstar(j)
    compute gain=swork(i,j)*tr(c)*inv(vhat)
    compute fwork(i,j)=exp(%logdensity(vhat,yerr))

    *
    * Do the SSM update step
    *
    compute xwork(i,j)=xwork(i,j)+gain*yerr
    compute swork(i,j)=swork(i,j)-gain*c*swork(i,j)
  end do j
end do i

```

Again, the lines in upper case are the ones which are special to this model. The first two (the `COMPUTE C` at the start) and the `COMPUTE YERR` in the middle) are just to get the `C` and `Y` for this data point. It's the `COMPUTE VHAT` which includes the switching component. In computing `XWORK` and `SWORK`, this also takes advantage of the fact that `A` is the identity to simplify the calculation.

This also adds an extra calculation of `F1HAT` and `F2HAT` which decompose the predictive variance (the `VHAT`) into the part due to the uncertainty in the coefficients (`F1HAT`) and that due to the (switching) regression variance (`F2HAT`). These are the probability-weighted averages across the M^2 combinations.

In the `DLMStart` function for this model, the only calculation needed for the state-space model is to square the standard deviations on the drift variances. (The regression standard deviations are squared to make variances as part of the calculation above).

```
compute sw=%diag(sigmav.^2)
```

The Kim filter initialization is to zero out `XSTAR` and make `SSTAR` a diagonal matrix with relatively large values (here 100). Whether 100 is a good choice isn't clear without at least some experimentation.

```
ewise xstar(i)=%zeros(ndlm,1)
ewise sstar(i)=100.0*%identity(ndlm)
```

By the way, there's no requirement that this be constant across variables—if you have a coefficient whose scale is quite different from the others, a diagonal matrix with differing values might be a better choice. For instance, multiplying the OLS variances by 10000 might be a choice which works in more cases than any fixed number.

The estimation code is similar to the previous model, except that the log likelihood is zeroed for the first ten data points. With five regressors, you need to leave out at least the first five—ten was the choice made by Kim and Nelson.

```
frml kimf = kf=KimFilter(t),%if(t<1962:1,0.0,log(kf))
maximize(parmset=dlmparms+mparams,start=DLMStart(),method=bfgs) $
kimf 1959:3 1989:2
```

12.3 Estimation with MCMC

As with other Markov Switching models, we add the history of regimes to parameter set, and repeat the steps of sampling the parameters given the regimes, and then regimes given the parameters. In general, however, we also have the unobservable states from the state-space model, which need to be sampled as well. So, the general process is to (in some order):

1. Sample the regimes given the states, and the switching and other parameters.
2. Sample the switching parameters given the regimes (nothing else should matter).
3. Sample the states given the regimes, switching and other parameters.
4. Sample the other parameters given states and regimes.

Number 2 is what we've seen in each section on Markov Switching. Numbers 3 and 4 are typical of Gibbs sampling for state-space models. What's new here is number 1. Whether the efficient FFBS algorithm can apply will depend a great deal on how the regimes and the states interact. We'll need Single-Move Sampling (page 154) for the Lam model, while we can use FFBS for the time-varying regression.

12.3.1 Lam Model by MCMC

The state-space setup used for estimation with the Kim filter can't be used in a standard way for MCMC. Ordinarily, the measurement equation in first differenced form:

$$\Delta y_t = \delta(S_t) + \Delta x_t$$

would be rearranged to

$$\Delta y_t - \delta(S_t) = [1, -1]X_t$$

for the purposes of sampling the states X given the regimes and parameters (number 3 on the list). And, in fact, that *can* be done to generate a series of x_t . The problem is that this equation is the only place where S_t appears, and it has no error term. Given the sampled state series, and the δ , there's only one possible set of S_t —the ones just used to create the x . Because X and S are linked by an identity, we can't sample the S treating the X as given.

For this example, we'll go back to the original specification of the model with y as the observable rather than its difference—the differenced form loses the connection to the level of y so it can't really give an accurate estimate of the trend series itself. With q as the number of lags in the AR, the size of the state vector is $q + 1$ with the extra one being the trend variable τ which will be put in the last position. The setup for the fixed parts of the state matrices is:

```
compute ndlm=q+1
*
dec rect a(ndlm,ndlm)
ewise a(i,j)=%if(i==ndlm,(i==j),(i==j+1))
*
dec rect c(ndlm,1)
compute c=%unitv(q,1)~~1.0
*
dec rect f(ndlm,1)
compute f=%unitv(ndlm,1)
```

We need a prior for the transition matrix, which will again be Dirichlet weakly favoring staying in each state:

```
dec vect[vect] gprior(nregimes)
dec vector tcounts(nregimes) pdraw(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
```

The initial values for the lag coefficients and the variance will come from an OLS regression of a preliminary estimate of the cycle on its lags.

```
linreg(define=areqn) x 1952:2 1984:4
# x{1 to q}
compute phi=%beta,sigsq=%seesq
```

The prior for the lag coefficients will be very loose, with a 0 mean and .5 standard deviation (precision is 4.0) on each:

```
dec vect bprior(%nreg)
dec symm hprior(%nreg,%nreg)
compute bprior=%zeros(%nreg,1)
compute hprior=%diag(%fill(%nreg,1,4.0))
```

We'll use an uninformative prior for the variance, since that will always be estimated using the full sample:

```
compute s2prior=1.0
compute nuprior=0.0
```

Finally, we'll use an uninformative prior for the δ .

The regimes will be sampled using Single-Move Sampling, using the template from page 155. We use the **DLM** instruction to compute the log likelihood. We first have to paste the current values of ϕ into the **A** matrix. In this model form, the switching comes in the **Z** component of the model, where the final component is the value of **DELTA** for the regime—since this changes with time, we need to put it in as the formula `delta(MSRegime(t))*%unitv(ndlm,ndlm)`

```
compute %psubmat(a,1,1,tr(phi))
dln(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,$
    z=delta(MSRegime(t))*%unitv(ndlm,ndlm)) gstart gend
compute logplast=%logl
compute pstar=%mcergodic(p)
```

```

do time=xstart,gend
  compute oldregime=MSRegime(time)
  do i=1,nregimes
    if oldregime==i
      compute logptest=logplast
    else {
      compute MSRegime(time)=i
      dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,$
        z=delta(MSRegime(t))*%unitv(ndlm,ndlm)) gstart gend
      compute logptest=%logl
    }

    compute pleft =%if(time==xstart,pstar(i),$
      p(i,MSRegime(time-1)))
    compute pright=%if(time==gend,1.0,$
      p(MSRegime(time+1),i))
    compute fps(i)=pleft*pright*exp(logptest-logplast)
    compute logp(i)=logptest
  end do i
  compute MSRegime(time)=%ranbranch(fps)
  compute logplast=logp(MSRegime(time))
end do time

```

Given the regimes, the transition is drawn using the standard techniques.

As we described above, the cycle is almost completely determined once we know the regimes and the δ (there's just a certain randomness coming from the pre-sample values). However, we'll sample as you would typically with a state-space model, using **DLM** with the option `TYPE=CSIMULATE` (conditional simulation). The estimated cycle is the first element of the state vector. This also produces a simulated value for the trend series in the final component of the state vector. Note that the **DLM** starts at `XSTART`, which is q elements before the official start of estimation. This allows generation of the pre-sample values.

```

compute %psubmat(a,1,1,tr(phi))
dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,type=csim,$
  z=delta(MSRegime(t))*%unitv(ndlm,ndlm)) xstart gend xstates
set x xstart gend = xstates(t)(1)

```

Given the generated x series, the ϕ (and variance) can be drawn using standard Bayesian procedures for a least squares regression. We'll reject non-stationary estimates, doing a redraw if we have an unstable root.

```

cmom(equation=areqn) gstart gend
:redraw
compute phi      =%ranmvpostcmom(%cmom,1.0/sigsq,hprior,bprior)
compute %eqnsetcoeffs(areqn,phi)
compute cxroots=%polycxroots(%eqnlagpoly(areqn,x))
if %cabs(cxroots(%rows(cxroots)))<=1.00 {
    disp "PHI draw rejected"
    goto redraw
}
compute sumsqr=%rsscmmom(%cmom,phi)
compute sigsq =(sumsqr+s2prior*nuprior)/%ranchisqr(%nobs+nuprior)

```

All that remains is to draw the δ . There are two possible approaches to this. First, we can unwind τ_t as in (12.5). Other than τ_0 (for which we just produced a simulated value), this is a linear function of the δ . The equation

$$y_t = \tau_0 + \delta_1 c_{1t} + \delta_2 c_{2t} + x_t$$

is in the form of a linear regression with serially correlated errors with a known form for the covariance matrix—the ϕ and σ^2 are assumed known. We can filter the data and sample as if it were a least squares regression. There is one potential problem with this for this particular model and data set: the second regime (low drift) is likely to be quite sparse so δ_2 won't be very well determined and we might need a prior that is more informative than we would like in order to keep the sampler working properly.

Instead, we're choosing to use (random walk) Metropolis within Gibbs. Our proposal density will be the current value plus a Normal increment. After a bit of experimenting (for random walk M-H, the goal is to come up with a moderate acceptance probability, neither too high, nor too low), we came up with (independent) Normal increments with standard deviation .10:

```

compute fdelta=||.10,0.0|0.0,.10||

```

The Metropolis code is:

```

compute %psubmat(a,1,1,tr(phi))
dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,$
    z=delta(MSRegime(t))*unitv(ndlm,ndlm)) gstart gend
compute logplast=%logl
*
compute [vector] deltatest=delta+%ranmvnormal(fdelta)
dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,$
    z=deltatest(MSRegime(t))*unitv(ndlm,ndlm)) gstart gend
compute logptest=%logl
compute alpha=exp(logptest-logplast)
if %ranflip(alpha)
    compute delta=deltatest,accept=accept+1

```

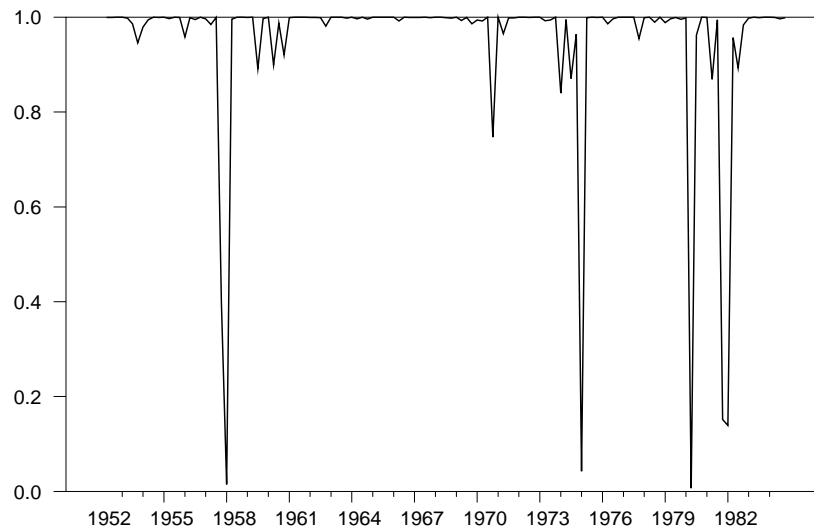


Figure 12.1: Filtered Probabilities from Kim filter estimates

The results are surprisingly different from those from the Kim filter. The filtered estimates of the probabilities of the high-mean regime from the Kim filter are in Figure 12.1 and those from the MCMC procedure are in Figure 12.2:

3

The Kim filter has picked up a mode which basically just identifies the outliers—quarters with sharply negative growth. MCMC comes up with a recession-expansion breakdown similar to what comes out of the Hamilton model. Because the data set is small enough, it's possible to do exact maximum likelihood (rather than the Kim approximation) and that finds that the two modes have very similar likelihoods, barely favoring the recession-expansion mode. However, because the outlier mode is so narrow, it's very hard to move to it using Gibbs sampling.

The Kim filter approximation also identifies the two modes, but much more clearly favors the outlier mode—since the data set is largely classified as one regime, the approximation will be more accurate than with the mode where there are many data points in each regime.

12.3.2 Time-varying parameters by MCMC

This is Example 12.4. The sampler for the regimes is simpler here than in the previous case. We can add the measurement errors and shocks to the regression coefficients to the parameter set and simulate them using **DLM** (given the previous settings for the regimes). Taking the measurement errors as given,

³Estimates of the regimes coming out of MCMC are smoothed, since they're produced using the full data set, but the smoothed estimates from the Kim filter are almost identical to the filtered ones.

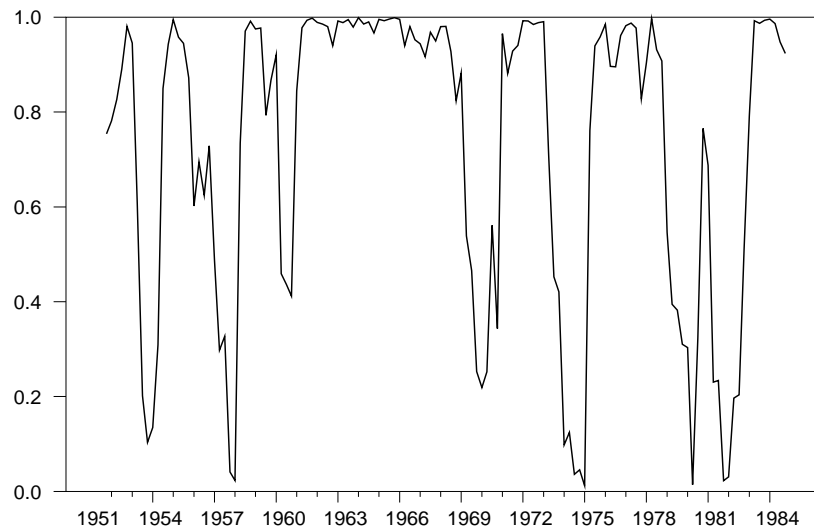


Figure 12.2: Smoothed Probabilities from MCMC Estimates

the regimes can be sampled using a simple FFBS algorithm exactly as in Example 8.3. While there is *some* correlation between the regimes and the measurement errors (almost no Gibbs sampler will avoid some correlation among blocks), this is nothing like the identity we had in the previous example, and isn't as tight a relationship as we would have if the regime-switching controlled the mean (rather than variance) of a process.⁴

The main problems come from it being a time-varying parameters regression. The difficulty coming out of the diffuse initial conditions won't be the problem here because the `PRESAMPLE=DIFFUSE` option on **DLM** can apply since there's only one branch that needs to be evaluated. The possibility of the variance in a drift being (optimally) zero remains. The variances will be drawn as inverse chi-squareds, which will never be true zero. However, if you use conditional simulation on a state-space model with a component variance being effectively zero, the element of the disturbances that that variance controls will be forced to be (near) zero as well, causing the next estimate of the variance to again be near zero. Thus, the Gibbs sampler will have an absorbing state at zero for each of the variances of the coefficient drifts, unless we use an informative prior; that ensures that if the variance goes to zero that it has a chance of being sampled non-zero in a future sweep.

As we did with the Kim filter estimates for this model, we'll start with a linear regression to get initial values. We'll start with the variances on the drifts as values too large to be reasonable.

```
compute sigmae(1)=.25*%seesq,sigmae(2)=2.5*%seesq
compute sigmav=%stderrs.^2
```

⁴The modal value of a Normal is still zero whether the variance is high or low.

The switching variance for the equation will be handled as before with a hierarchical prior, using a non-informative prior for the common variance scale:

```
compute nucommon=0.0
compute s2common=.5*%seesq
dec vect nuprior(nregimes)
dec vect s2prior(nregimes)
compute nuprior=%fill(nregimes,1,4.0)
compute s2prior=%fill(nregimes,1,1.0)
compute scommon =%seesq
```

The prior on the coefficient drift variances is (very) weakly informative, centered on a small multiple of the least squares variances.

```
dec vect nusw(ndlm)
dec vect s2sw(ndlm)
*
ewise nusw(i)=1.0
ewise s2sw(i)=(.01*%stderrs(i))^2
```

The Gibbs sampling loop starts by simulating the state-space model given the settings for the variances and the current values of the regimes. This will give us simulated coefficient drifts in `WHAT`, which is a `SERIES` of `VECTORS`, and equation errors in `VHAT`, similarly a `SERIES` of `VECTORS` (size one in this case, since there's only one observable). This also includes simulated values for the state vector, which will here be the (time-varying) regression coefficients.

```
dlim(y=m1gr,c=%eqnxvector(mdeq,t),sv=sigmae(MSRegime(t)), $
     sw=%diag(sigmav),presample=diffuse,type=csimulate,$
     what=what,vhat=vhat) gstart gend xstates vstates
```

We then treat the `WHAT` and `VHAT` as given in drawing the variances. This does the coefficient drift variances:

```
do i=1,ndlm
  sstats gstart+ncond gend what(t)(i)^2>>sumsqr
  compute sigmav(i)=(sumsqr+nusw(i)*s2sw(i))/ $
    %ranchisqr(%nobs+nusw(i))
end do i
```

and this does the (switching) equation variances using a hierarchical prior:

```

sstats gstart+ncond gend $
      vhat(t)(1)^2/sigmae(MSRegime(t))>>sumsqr
compute scomon=(scomon*sumsqr+nucommon*s2common)/$
      %ranchisqr(%nobs+nucommon)
do k=1,nregimes
  sstats(smpl=MSRegime(t)==k) gstart+ncond gend $
      vhat(t)(1)^2>>sumsqr
  compute sigmae(k)=(sumsqr+nuprior(k)*scomon)/$
      %ranchisqr(%nobs+nuprior(k))
end do k

```

The regimes are sampled using FFBS:

```

compute pstar=%mcergodic(p)
do time=gstart,gend
  compute pt_t1(time)=%mcstate(p,pstar)
  compute pstar=%msupdate(RegimeF(time),pt_t1(time),fpt)
  compute pt_t(time)=pstar
end do time
@%mssample p pt_t pt_t1 MSRegime

```

which requires the function `RegimeF` which returns the vector of likelihoods given the simulated `VHAT`:

```

function RegimeF time
type integer time
type vector RegimeF
local integer i
*
dim RegimeF(nregimes)
ewise RegimeF(i)=exp(%logdensity(sigmae(i),vhat(time)(1)))
end

```

Note that (at least with this data set), this requires a very large number of draws to get the numerical standard errors on the coefficient variances down to a reasonable level (compared to their means). The switching variance and the transition probabilities are quite a bit more stable.

Example 12.1 Lam GNP Model-Kim Filter

This estimates the Lam MS model for GNP using the Kim filter. This is Application 5.4.2 from pp 111-115 of Kim & Nelson (1999). In this workbook, the detailed discussion is in Section 12.2.1.

```

open data gdp4795.prn
calendar(q) 1947:1
data(format=free,org=columns) 1947:01 1995:03 gdp
*
set loggdp = 100.0*log(gdp)
set g = 100*log(gdp/gdp{1})
@nbercycles(down=contract)
*
*****
*
* This is common to any MS state space model analyzed using the
* Kim filter.
*
@MSSetup(regimes=2)
*
* These are the collapsed SSM mean and variance from the previous
* time period.
*
dec vect[vect] xstar(nregimes)
dec vect[symm] sstar(nregimes)
*
* This is the filtered best estimate for the state vector
*
dec series[vect] xstates
*****
*
* This is specific to this model
*
* Number of lags in the AR
*
compute q=2
dec vect phi(q)
*
* Fill in the fixed coefficients in the A, C and F matrices
*
compute ndlm=q
*
dec rect a(ndlm,ndlm)
ewise a(i,j)=(i==j+1)
*
dec rect c
compute c=||1.0,-1.0||~%zeros(1,ndlm-2)
*
dec rect f(ndlm,1)
compute f=%unitv(ndlm,1)
dec symm sw(ndlm,ndlm)
compute sv=0.0

```

```

*
dec vect delta(nregimes)
*
dec vect x0(ndlm)
compute x0=%zeros(ndlm,1)
*
nonlin(parmset=dlmparms) delta sigsq phi
nonlin(parmset=msparms) theta
nonlin(parmset=initparms) x0
*****
*
* This does a single step of the Kim (approximate) filter
*
function KimFilter time
type integer    time
*
local integer    i j
local real       yerr likely
local symm       vhat
local rect       gain
local rect       phat(nregimes,nregimes)
local rect       fwork(nregimes,nregimes)
local rect[vect] xwork(nregimes,nregimes)
local rect[symm] swork(nregimes,nregimes)
*
do i=1,nregimes
  do j=1,nregimes
    *
    * Do the SSM predictive step
    *
    compute xwork(i,j)=a*xstar(j)
    compute swork(i,j)=a*sstar(j)*tr(a)+sw
    *
    * Do the prediction error for y under state i, and
    * compute the density function for the prediction error.
    *
    compute yerr=g(time)-(%dot(c,xwork(i,j))+delta(i))
    compute vhat=c*swork(i,j)*tr(c)+sv
    compute gain=swork(i,j)*tr(c)*inv(vhat)
    compute fwork(i,j)=exp(%logdensity(vhat,yerr))
    *
    * Do the SSM update step
    *
    compute xwork(i,j)=xwork(i,j)+gain*yerr
    compute swork(i,j)=swork(i,j)-gain*c*swork(i,j)
  end do j
end do i
*
* Compute the Hamilton filter likelihood
*
compute likely=0.0
do i=1,nregimes
  do j=1,nregimes
    compute phat(i,j)=p(i,j)*pstar(j)*fwork(i,j)

```

```

        compute likely=likely+phat(i,j)
    end do j
end do i
*
* Compute the updated probabilities of the regime combinations.
*
compute phat=phat/likely
compute pstar=%sumr(phat)
compute pt_t(time)=pstar
*
* Collapse the SSM matrices down to one per regime.
*
compute xstates(time)=%zeros(ndlm,1)
do i=1,nregimes
    compute xstar(i)=%zeros(ndlm,1)
    do j=1,nregimes
        compute xstar(i)=xstar(i)+xwork(i,j)*phat(i,j)/pstar(i)
    end do j
    compute sstar(i)=%zeros(ndlm,ndlm)
    do j=1,nregimes
        compute sstar(i)=sstar(i)+phat(i,j)/pstar(i)*$
            (swork(i,j)+%outerxx(xstar(i)-xwork(i,j)))
    end do j
    *
    * This is the overall best estimate of the filtered state.
    *
    compute xstates(time)=xstates(time)+xstar(i)*pstar(i)
end do i
compute KimFilter=likely
end
*****
*
* This is the start up code for each function evaluation
*
function DLMStart
*
local integer i
*
* Fill in the top row in the A matrix
*
compute %psubmat(a,1,1,tr(phi))
*
* Compute the full matrix for the transition variance
*
compute sw=f*tr(f)*sigsq
*
* Transform the theta to transition probabilities.
*
compute p=%MSLogisticP(theta)
compute pstar=%mcergodic(p)
compute p=%mspexpand(p)
*
* Initialize the KF state and variance. This is set up for estimating
* the pre-sample states.

```

```

*
ewise xstar(i)=x0
ewise sstar(i)=%zeros(ndlm,ndlm)
end
*****
*
* Get guess values for the means from running an AR(2) on the
* growth rate. However, the guess values for the AR
* coefficients need to be input; the BOXJENK values aren't
* persistent enough because the mean model doesn't take
* switches into account.
*
boxjenk(ar=q,constant) g * 1984:4
compute delta(1)=%beta(1)+0.5,delta(2)=%beta(1)-1.5
compute phi=||1.2,-.3||
compute sigsq=%seesq
*
* Initialize theta from transitions common for Hamilton type
* models
*
compute theta=%msplogistic(||.9,.5||)
*
frml kimf = kf=KimFilter(t),log(kf)
*
* Do initializations
*
@MSFilterInit
gset xstates = %zeros(ndlm,1)
*
maximize(parmset=msparms+dlmparms+initparms,start=DLMstart(),$
    reject=sigsq<=0.0,method=bfgs) kimf 1952:2 1984:4
*
set cycle 1952:2 1984:4 = xstates(t)(1)
set trend 1952:2 1984:4 = loggdp-cycle
*
set phigh 1952:2 1984:4 = pt_t(t)(1)
graph(footer=$
    "Figure 5.2a Filtered Probabilities of High-Growth Regime")
# phigh
graph(footer=$
    "Figure 5.3 Real GNP and Markov-switching trend component") 2
# loggdp 1952:2 1984:4
# trend
graph(footer=$
    "Figure 5.4 Cyclical component from Lam's Model")
# cycle

```

Example 12.2 Time-Varying Parameters-Kim Filter

This estimates a time-varying parameters regression with MS regression variances using the Kim filter. It is from Application 5.5.1 from pp 115-121 of Kim & Nelson (1999), and the detailed discussion is in Section 12.2.2.

```
open data tvpmrkf.prn
cal(q) 1959:3
data(format=prn,nolabels,org=columns) 1959:03 1989:02 $
    mlgr dintlag inflag surplag mllag qtr
*
* Least squares estimates of the money demand function
*
linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag
*
equation(lastreg) mdeq
compute ndlm=%nreg
dec vect sigmav(ndlm)
dec symm sw(ndlm,ndlm)
*
@MSSSetup(regimes=2)
*****
*
* This is common to any MS state space model analyzed using
* the Kim filter.
*
* These are the collapsed SSM mean and variance from the
* previous time period.
*
dec vect[vect] xstar(nregimes)
dec vect[symm] sstar(nregimes)
*
* These are the work SSM mean and variance for the current time
* period.
*
dec rect[vect] xwork(nregimes,nregimes)
dec rect[symm] swork(nregimes,nregimes)
*
dec rect      fwork(nregimes,nregimes)
*
dec vect      pstar(nregimes)
dec rect      p(nregimes,nregimes)
*
dec series[vect] xstates
dec series[vect] pt_t
*
nonlin(parmset=msparms) theta
compute theta=%zeros(nregimes-1,nregimes)
*
*****
*
* What switches in this model is the variance in the
```

```

* measurement equation.
*
dec vect sigmae(nregimes)
*
* Get guess values based upon the results of a linear
* regression. All standard deviations are scales of the
* corresponding variances in the least squares regression.
* (The drift ones being a much smaller multiple).
*
compute sigmae(1)=.25*sqrt(%seesq),sigmae(2)=1.5*sqrt(%seesq)
compute sigmav=.01*%stderrs
*
nonlin(parmset=dlmparms) sigmae sigmav
*****
*
* These are for keeping track of the decomposition of the
* predictive variance for money growth.
*
set flhat = 0.0
set f2hat = 0.0
*****
*
* This does a single step of the Kim (approximate) filter
*
function KimFilter time
type integer time
*
local integer i j
local real      yerr likely
local symm      vhat
local rect      gain
local rect      phat(nregimes,nregimes)
local rect      fwork(nregimes,nregimes)
*
local rect      c
*
* Pull out the C matrix, which is time-varying (regressors at
* *time*)
*
compute c=tr(%eqnxvector(mdeq,time))
*
compute flhat(time)=0.0,f2hat(time)=0.0
do i=1,nregimes
  do j=1,nregimes
    *
    * Do the SSM predictive step. In this application A is
    * the identity, so the calculations simplify quite a bit.
    *
    compute xwork(i,j)=xstar(j)
    compute swork(i,j)=sstar(j)+sw
    *
    * Do the prediction error and variance for y under
    * regime i. The predictive variance is the only part of
    * this that depends upon the regime. Compute the

```



```

    * density function for the prediction error.
    *
    compute yerr=mlgr(time)-%dot(c,xwork(i,j))
    compute vhat=c*swork(i,j)*tr(c)+sigmae(i)^2
    *
    * Do the decomposition of vhat into its components and
    * add probability-weighted values to the sums across
    * (i,j)
    *
    compute flhat(time)=flhat(time)+$
        %scalar(c*swork(i,j)*tr(c))*p(i,j)*pstar(j)
    compute f2hat(time)=f2hat(time)+sigmae(i)^2*p(i,j)*pstar(j)
    compute gain=swork(i,j)*tr(c)*inv(vhat)
    compute fwork(i,j)=exp(%logdensity(vhat,yerr))
    *
    * Do the SSM update step
    *
    compute xwork(i,j)=xwork(i,j)+gain*yerr
    compute swork(i,j)=swork(i,j)-gain*c*swork(i,j)
end do j
end do i

*
* Everything from here to the end of the function is the same
* for all such models.
*
* Compute the Hamilton filter likelihood
*
compute likely=0.0
do i=1,nregimes
    do j=1,nregimes
        compute phat(i,j)=p(i,j)*pstar(j)*fwork(i,j)
        compute likely=likely+phat(i,j)
    end do j
end do i

*
* Compute the updated probabilities of the regime combinations.
*
compute phat=phat/likely
compute pstar=%sumr(phat)
compute pt_t(time)=pstar
*
* Collapse the SSM matrices down to one per state
*
compute xstates(time)=%zeros(ndlm,1)
do i=1,nregimes
    compute xstar(i)=%zeros(ndlm,1)
    do j=1,nregimes
        compute xstar(i)=xstar(i)+xwork(i,j)*phat(i,j)/pstar(i)
    end do j
end do i

*
* This is the overall best estimate of the filtered state.
*
compute xstates(time)=xstates(time)+xstar(i)*pstar(i)
compute sstar(i)=%zeros(ndlm,ndlm)

```

```

    do j=1,nregimes
        compute sstar(i)=sstar(i)+phat(i,j)/pstar(i)*$
            (swork(i,j)+%outerxx(xstar(i)-xwork(i,j)))
    end do j
end do i
compute KimFilter=likely
end
*****
*
* This is the start up code for each function evaluation
*
function DLMStart
*
local integer i
*
* Compute the full matrix for the transition variance
*
compute sw=%diag(sigmav.^2)
*
* Transform the theta to transition probabilities.
*
compute p=%MSLogisticP(theta)
compute pstar=%mcergodic(p)
compute p=%mspexpand(p)
*
* Initialize the KF state and variance. This uses a large
* finite value for the variance, then conditions on the first
* 10 observations.
*
ewise xstar(i)=%zeros(ndlm,1)
ewise sstar(i)=100.0*%identity(ndlm)
end
*****
*
* Skip the first 10 data points in evaluating the likelihood.
* The textbook seems to have p00 and p11 mislabeled in Table
* 5.2.
*
frml kimf = kf=KimFilter(t),%if(t<1962:1,0.0,log(kf))
@MSFilterInit
gset xstates = %zeros(ndlm,1)
maximize(parmset=dlmparms+msparms,start=DLMStart(),method=bfgs) $
    kimf 1959:3 1989:2
*
set totalvar = flhat+f2hat
graph(style=bar,key=below,$
    klabels=||"Total Variance","TVP","MRKF"||,$
    footer=$
        "Figure 5.5 Decomposition of monetary growth uncertainty") 3
# totalvar 1962:1 *
# flhat 1962:1 *
# f2hat 1962:1 *
*
set plow = pt_t(t)(1)

```

```

graph(header="Filtered Probability of Low-Variance Regime")
# plow
*
set binterest 1962:1 * = xstates(t) (2)
set binflation 1962:1 * = xstates(t) (3)
set bsurplus 1962:1 * = xstates(t) (4)
set blagged 1962:1 * = xstates(t) (5)
graph(header="Estimated Coefficient on Lagged Change in R")
# binterest
graph(header="Estimated Coefficient on Lagged Inflation")
# binflation
graph(header="Estimated Coefficient on Lagged Surplus")
# binflation
graph(header="Estimated Coefficient on Lagged Growth")
# binflation

```

Example 12.3 Lam GNP Model-MCMC

This estimates the Lam GNP model using MCMC. The details are in Section 12.3.1.

```

open data gdp4795.prn
calendar(q) 1947:1
data(format=free,org=columns) 1947:01 1995:03 gdp
*
set y = 100.0*log(gdp)
set g = y-y{1}
*
* Estimate a trend (just for initialization purposes) from the
* HP filter.
*
filter(type=hp) y / x
set x = y-x
stats(fractiles) g
*
@MSSSetup(regimes=2)
*
dec vect delta(nregimes)
compute delta(1)=%fract75
compute delta(2)=%fract10
*
* Number of lags in the stationary AR
*
compute q=2
linreg(define=areqn) x 1952:2 1984:4
# x{1 to q}
compute gstart=%regstart(),gend=%regend()
compute xstart=gstart-q
*
* Fill in the fixed coefficients in the A, C and F matrices.
* Unlike the example with the Kim filter, we'll be using the
* DLM instruction, so we can use a 1x1 SW matrix combined with

```

```

* the F option.
*
compute ndlm=q+1
*
dec rect a(ndlm,ndlm)
ewise a(i,j)=%if(i==ndlm,(i==j),(i==j+1))
*
dec rect c(ndlm,1)
compute c=%unitv(q,1)~~1.0
*
dec rect f(ndlm,1)
compute f=%unitv(ndlm,1)
*
* Get initial values for the sampler from a linear regression.
*
compute phi=%beta,sigsq=%seesq
*
* Prior for the lag coefficients. (0 mean, .5 s.d. for each,
* independent)
*
dec vect bprior(%nreg)
dec symm hprior(%nreg,%nreg)
compute bprior=%zeros(%nreg,1)
compute hprior=%diag(%fill(%nreg,1,4.0))
*
* Prior for the variance
*
compute s2prior=1.0
compute nuprior=0.0
*
* Prior for transitions
*
dec vect[vect] gprior(nregimes)
dec vector tcounts(nregimes) pdraw(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
*
* For draws for the delta's
*
compute fdelta=||.10,0.0|0.0,.10||
*
* Draw counts.
*
compute nburn=5000,ndraws=5000
*
* Initial values for p.
*
dim p(nregimes,nregimes)
ewise p(i,j)=gprior(j)(i)/%sum(gprior(j))
*
* For re-labeling
*
dec vect temp
dec rect ptemp

```

```

*
* For single-move sampling
*
dec vect fps logp
dim fps(nregimes) logp(nregimes)
*
* Initialize regimes based upon 1 for above average and 2 for
* below.
*
gset MSRegime = 1+fix(g<.2*delta(1)+.8*delta(2))
*
dec vect[series] count(nregimes) fcount(nregimes)
*
* For bookkeeping
*
set regime1      = 0.0
set trendsum     = 0.0
set trendsumsq   = 0.0
*
nonlin(parmset=mcmcparms) delta phi sigsq p
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %parmspeek(mcmcparms)
*
compute accept=0
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
    "Gibbs sampling"

do draw=-nburn,ndraws
    *
    * Single-move sampling for the regimes.
    *
    * Evaluate the log likelihood at the current settings.
    *
    compute %psubmat(a,1,1,tr(phi))
    dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,$
        z=delta(MSRegime(t))*%unitv(ndlm,ndlm)) gstart gend
    compute logplast=%logl
    compute pstar=%mcergodic(p)
    do time=xstart,gend
        compute oldregime=MSRegime(time)
        do i=1,nregimes
            if oldregime==i
                compute logptest=logplast
            else {
                compute MSRegime(time)=i
                dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,$
                    z=delta(MSRegime(t))*%unitv(ndlm,ndlm)) gstart gend
                compute logptest=%logl
            }
        }
    compute pleft =%if(time==xstart,pstar(i),$
        p(i,MSRegime(time-1)))
    compute pright=%if(time==gend ,1.0,$
        p(MSRegime(time+1),i))
    compute fps(i)=pleft*pright*exp(logptest-logplast)

```

```

        compute logp(i)=logptest
    end do i
    compute MSRegime(time)=%ranbranch(fps)
    compute logplast=logp(MSRegime(time))
end do time
*
* Draw p's
*
@MSDrawP(prior=gprior) gstart gend p
*
* Draw the state-space states given the current regimes and other
* parameters. First, fill in the top row in the A matrix using the
* current values of phi.
*
compute %psubmat(a,1,1,tr(phi))
dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,type=csim,$
    z=delta(MSRegime(t))*%unitv(ndlm,ndlm)) xstart gend xstates
set x xstart gend = xstates(t)(1)
*
* Draw the lag coefficients and variance using a linear
* regression of x on its lags. Reject clearly explosive roots.
*
cmom(equation=areqn) gstart gend
:redraw
compute phi      =%ranmvpostcmom(%cmom,1.0/sigsq,hprior,bprior)
compute %eqnsetcoeffs(areqn,phi)
compute cxroots=%polycxroots(%eqnlagpoly(areqn,x))
if %cabs(cxroots(%size(cxroots)))<=1.00 {
    disp "PHI draw rejected"
    goto redraw
}
compute sumsqr=%rsscmom(%cmom,phi)
compute sigsq =(sumsqr+s2prior*nuprior)/%ranchisqr(%nobs+nuprior)
*
* Draw the delta's by random walk Metropolis-Hastings
*
compute %psubmat(a,1,1,tr(phi))
dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,$
    z=delta(MSRegime(t))*%unitv(ndlm,ndlm)) gstart gend
compute logplast=%logl
*
compute [vector] deltatest=delta+%ranmvnormal(fdelta)
dlm(a=a,f=f,c=c,sw=sigsq,y=y,presample=ergodic,$
    z=deltatest(MSRegime(t))*%unitv(ndlm,ndlm)) gstart gend
compute logptest=%logl
compute alpha=exp(logptest-logplast)
if %ranflip(alpha)
    compute delta=deltatest,accept=accept+1
*
* Check to see if we need to switch labels
*
compute swaps=%index(-1.0*delta)
if %MSSwapCheck(swaps) {
    disp "Need label switching"

```

```

        compute temp=delta
        ewise delta(i)=temp(swaps(i))
        gset MSRegime = swaps(MSRegime(t))
        compute ptemp=p
        ewise p(i,j)=ptemp(swaps(i),swaps(j))
    }
    *
    infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")
    *
    * If we're past the burn-in, do the bookkeeping
    *
    if draw>0 {
        *
        * Combine delta, sigsq and p into a single vector and save the
        * draw.
        *
        compute bgibbs(draw)=%parmspeek(mcmcparms)
        *
        * Update the probability of being in regime 1
        *
        set regimel = regimel+(MSRegime(t)==1)
        *
        * Update the estimate of the trend
        *
        set tau xstart gend = xstates(t)(ndlm)
        set trendsum xstart gend = trendsum+tau
        set trendsumsq xstart gend = trendsumsq+tau^2
    }
end do draw
infobox(action=remove)
*
set trendsum = trendsum/ndraws
set trendsumsq = trendsumsq/ndraws-trendsum^2
set trendlower = trendsum-2.0*sqrt(trendsumsq)
set trendupper = trendsum+2.0*sqrt(trendsumsq)
*
graph(footer="Estimate of Trend with 2 s.d. bounds") 4
# y
# trendsum xstart gend
# trendlower xstart gend 3
# trendupper xstart gend 3
*
set regimel = regimel/ndraws
graph(footer="Probability of being in high growth regime")
# regimel xstart gend
@mcmcpostproc(ndraws=ndraws,mean=bmeans,stderrs=bstderrs) bgibbs
report(action=define)
report(atrow=1,atcol=2) "Mean" "S.D"
report(atrow=2,atcol=1) "delta(1)" bmeans(1) bstderrs(1)
report(atrow=3,atcol=1) "delta(2)" bmeans(2) bstderrs(2)
report(atrow=4,atcol=1) "phi(1)" bmeans(3) bstderrs(3)
report(atrow=5,atcol=1) "phi(2)" bmeans(4) bstderrs(4)
report(atrow=6,atcol=1) "sigmasq" bmeans(5) bstderrs(5)
report(atrow=7,atcol=1) "p" bmeans(6) bstderrs(6)

```

```

report(atrow=8,atcol=1) "q" bmeans(9) bstderrs(9)
report(action=format,picture="###.###")
report(action=show)
*
compute plabels=%parmlabels(mcmcparms)
do i=1,%rows(plabels)
    set xstat 1 ndraws = bgibbs(t)(i)
    density(smoothing=1.5) xstat 1 ndraws xx fx
    scatter(style=line,footer=plabels(i))
    # xx fx
end do i

```

Example 12.4 Time-Varying Parameters-MCMC

This estimates a time-varying parameters regression with a Markov switching equation variance, using MCMC. This is discussed in detail in Section 12.3.2.

```

open data tvpmrkf.prn
cal(q) 1959:3
data(format=prn,nolabels,org=columns) 1959:03 1989:02 $
    mlgr dintlag inflag surplag mllag qtr
*
* Least squares estimates of the money demand function
*
linreg mlgr 1962:1 *
# constant dintlag inflag surplag mllag
*
compute gstart=%regstart(),gend=%regend()
*
equation(lastreg) mdeq
compute ndlm=%nreg
dec vect sigmav(ndlm)
dec symm sw(ndlm,ndlm)
*
* Number of observations to skip at the front end due to unit
* roots.
*
compute ncond=10
*
@MSSetup(regimes=2)
*
* Prior for transitions. Weak Dirichlet priors with preference
* for staying in a given regime.
*
dec vect[vect] gprior(nregimes)
dec vect tcounts(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
*
dec rect          p(nregimes,nregimes)
*
* Initial values for p.

```



```

*
ewise p(i,j)=gprior(j) (i)/%sum(gprior(j))
*
dec series[vect] xstates
*
*****
*
* What switches in this model is the variance in the
* measurement equation.
*
dec vect sigmae(nregimes)
*
* Get guess values based upon the results of a linear
* regression. All variances are scales of the corresponding
* variances in the least squares regression. Start with what
* seems to be far too much time-variation.
*
compute sigmae(1)=.25*%seesq, sigmae(2)=2.5*%seesq
compute sigmav=%stderrs.^2
*
*****
gset xstates = %zeros(ndlm,1)
*****
*
* Hierarchical prior for sigmas
* Uninformative prior on the common component.
*
compute nucommon=0.0
compute s2common=.5*%seesq
*
* Priors for the relative variances (if needed)
*
dec vect nuprior(nregimes)
dec vect s2prior(nregimes)
compute nuprior=%fill(nregimes,1,4.0)
compute s2prior=%fill(nregimes,1,1.0)
compute scomon =%seesq
*
* Prior for the coefficient drift variances
*
dec vect nusw(ndlm)
dec vect s2sw(ndlm)
*
ewise nusw(i)=1.0
ewise s2sw(i)=(.01*%stderrs(i))^2
*
compute nburn=10000, ndraws=100000
*
* For relabeling
*
dec vect tempsigma(nregimes)
dec rect temp(nregimes,nregimes)
*
dec series[vect] vhat

```

```

*****
function RegimeF time
type integer time
type vector RegimeF
local integer i
*
dim RegimeF(nregimes)
ewise RegimeF(i)=exp(%logdensity(sigmae(i),vhat(time)(1)))
end
*****
gset MSRegime = fix(%if(%resids^2>%seesq,2,1))
*
dec series[vect] betahist
gset betahist gstart gend = %zeros(ndlm,1)
*
nonlin(parmset=mcmcparms) sigmav sigmae p
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %parmspeek(mcmcparms)
*
compute gstart=%regstart(),gend=%regend()
set regime1 gstart gend = 0.0
*
infobox(action=define,lower=-nburn,upper=ndraws,progress) $
"Gibbs Sampling"
do draw=-nburn,ndraws
*
* Draw coefficients and shocks jointly given the regimes.
*
dlm(y=mlgr,c=%eqnxvector(mdeq,t),sv=sigmae(MSRegime(t)), $
    sw=%diag(sigmav),presample=diffuse,type=csimulate,$
    what=what,vhat=vhat) gstart gend xstates vstates
*
* Draw sigmav's given the what's.
*
do i=1,ndlm
    sstats gstart+ncond gend what(t)(i)^2>>sumsq
    compute sigmav(i)=(sumsq+nusw(i)*s2sw(i))/ $
        %ranchisqr(%nobs+nusw(i))
end do i
*
* Draw the common component for sigmae
*
sstats gstart+ncond gend $
    vhat(t)(1)^2/sigmae(MSRegime(t))>>sumsq
compute scomon=(scomon*sumsq+nucommon*s2common)/ $
    %ranchisqr(%nobs+nucommon)
*
* Draw the specific components for the sigmae
*
do k=1,nregimes
    sstats(smpl=MSRegime(t)==k) gstart+ncond gend $
        vhat(t)(1)^2>>sumsq
    compute sigmae(k)=(sumsq+nuprior(k)*scomon)/ $
        %ranchisqr(%nobs+nuprior(k))
end do k
end

```

```

end do k
*
* Relabel if necessary
*
ewise tempsigma(i)=sigmae(i)
compute swaps=%index(tempsigma)
if %MSSwapCheck(swaps) {
    disp "Draw" draw "Executing swap"
    *
    * Relabel the variances
    *
    ewise sigmae(i)=tempsigma(swaps(i))
    *
    * Relabel the transitions
    *
    compute tempp=p
    ewise p(i,j)=tempp(swaps(i),swaps(j))
}
*
* Draw MSRegime using the vhat's and variances. At this point,
* it's a simple variance-switch which can be done using FFBS.
*
@MSFilterInit
do time=gstart,gend
    @MSFilterStep time RegimeF(time)
end do time
*
* Backwards sample
*
@MSSample gstart gend MSRegime
*
* Draw p's
*
@MSDrawP(prior=gprior) gstart gend p
infobox(current=draw)
if draw>0 {
    *
    * Do the bookkeeping
    *
    set regimel gstart gend = regimel+(MSRegime==1)
    compute bgibbs(draw)=%parmspeek(mcmcparms)
    gset betahist gstart gend = betahist+xstates
}
end do draw
infobox(action=remove)
*
@mcmcpostproc(ndraws=ndraws,mean=bmeans,stderrs=bstderrs) bgibbs
*
report(action=define)
report(atrow=1,atcol=1,fillby=cols) %parmslabels(mcmcparms)
report(atrow=1,atcol=2,fillby=cols) bmeans
report(atrow=1,atcol=3,fillby=cols) bstderrs
report(action=format,picture="*.#####")
report(action=show)

```

```
*
set regime1 = regime1/ndraws
graph(footer="Probability of low-variance regime")
# regime1
*
gset betahist gstart gend = betahist/ndraws
*
set binterest gstart gend = betahist(t) (2)
set binflation gstart gend = betahist(t) (3)
set bsurplus gstart gend = betahist(t) (4)
set blagged gstart gend = betahist(t) (5)
graph(header="Estimated Coefficient on Lagged Change in R")
# binterest
graph(header="Estimated Coefficient on Lagged Surplus")
# binflation
graph(header="Estimated Coefficient on Lagged Inflation")
# binflation
graph(header="Estimated Coefficient on Lagged M1 Growth")
# blagged
```

Markov Switching ARCH and GARCH

Variations on GARCH models dominate the empirical literature in finance. However, while they seem to be capable of explaining the volatility in the observed data, it's been long known that their out-of-sample forecasting ability is suspect. In Hamilton & Susmel (1994), a standard GARCH model forecast volatility less well than the sample variance.¹ Hamilton and Susmel, and independently, Cai (1994), proposed a Markov switching ARCH model as an alternative. In these models, instead of the lagged variance term providing the strong connection for volatility from one period to the next, a Markov switching model governs switches between several variance regimes. These are similar to models analyzed in Chapter 8, with some additional ARCH terms for shorter-term dependence.

Given that ARCH models were very quickly superseded by the more flexible GARCH models, a question arises why the addition of Markov switching was first applied to ARCH rather than GARCH. There's a very good, practical, reason for this. The ARCH specification has a finite length memory for the regimes—in Hamilton and Susmel's model, the likelihood depends upon regimes back the number of ARCH lags, while in Cai's formulation, it depends only upon the current regime. However, in a GARCH model, the lagged variance term will depend upon the entire prior history of the regimes. Thus, the exact likelihood function can't be computed, particularly in large finance data sets. Instead, some form of approximation is required to reduce the length of dependence. And, for the same reason, MCMC estimation is rendered almost infeasible because it requires Single Move sampling (page 154) which takes too long given the typical size of the data sets. In addition, the main reason that GARCH supplanted ARCH—the lack of memory in the ARCH process—doesn't apply once you add Markov switching, as the MS process itself provides the persistence.

In this chapter, we'll first look at the switching ARCH, then move on to examine switching GARCH.

¹It should be noted, however, that using the squared residuals as a proxy for (unobservable) volatility in evaluating out-of-sample forecasting (as was done in HS) is known to give weak and possibly misleading results. In particular, the MAE is strongly biased in favor of a fixed variance forecast. See Patton (2011).

13.1 Markov Switching ARCH models

Neither Hamilton and Susmel (hereafter HS) nor Cai chose to allow the entire ARCH process to switch with regime. HS, in particular, made clear that they thought the fully switching model would be overparameterized. Instead, both papers switch one parameter to make the overall variance change among regimes.

The Cai model can be seen in the RATS **SWARCH.RPF** example. Because the switch is in the constant in the ARCH process, the likelihood depends upon only the current regime, making estimation particularly simple. We'll focus here on the HS model.

HS analyze weekly value-weighted SP500 returns. Their mean model is a one-lag autoregression, which is assumed to be fixed among regimes:

$$y_t = \alpha + \beta y_{t-1} + u_t$$

$$Eu_t^2 \equiv h_t = a_0 + \sum_{i=1}^q a_i u_{t-i}^2$$

Cai's model replaces a_0 with $a_0(S_t)$ with S_t Markov switching. With HS, what switches is a variance inflation factor, which gives the formula:

$$Eu_t^2 = g(S_t) \left\{ a_0 + \sum_{i=1}^q \frac{a_i u_{t-i}^2}{g(S_{t-i})} \right\}$$

Compared with Cai's model, this won't respond quite as dramatically to large shocks since this scales down the persistence of the lagged squared residuals in the high-variance regimes. Whether one or the other tends to work better is an open question. HS makes the likelihood dependent upon current and q lags of the regimes, so it's a bit harder to set up and takes longer to estimate.

In addition, HS include an asymmetry term:

$$Eu_t^2 = g(S_t) \left\{ a_0 + \xi \frac{u_{t-1}^2}{g(S_{t-1})} (u_{t-1} < 0) + \sum_{i=1}^q \frac{a_i u_{t-i}^2}{g(S_{t-i})} \right\} \quad (13.1)$$

and use conditionally Student- t errors in most of their models. In (13.1), there is a need for a normalization, since the right side is a product with fully free coefficients in both factors. HS choose to fix one of the g at 1. In our coding for this, we'll instead normalize with $a_0 = 1$, which is simpler to implement since it allows the g to be put into the parameter set as a complete VECTOR.

HS estimate quite a few models. What we'll show is a three regime-two lag asymmetric model with t errors. The initial setup is done with:

```
compute q=2
@MSSSetup(lags=q, regimes=3)
```

This sets up the expanded regime system required for handling the lagged regimes. This defines the variable `NEXPAND` as the number of expanded regimes, which will here be $3^{2+1} = 27$.

The parameter set has three parts:

1. the mean equation parameters,
2. the ARCH model parameters, which include the ARCH lags (the VECTOR named `A`), the variance inflations (the VECTOR named `GV`), plus, for this model, the asymmetry parameter (`XI`) and degrees of freedom (`NU`),
3. the Markov switching parameters, which here will be the logistic indexes `THETA`.

```
dec vect gv(nregimes)
*
dec vect a(q)
dec real xi
dec real nu
*
nonlin(parmset=archparms) a gv xi nu
nonlin(parmset=msparms) theta
```

This does a linear regression to define the mean equation and provide guess values for many of the parameters.

```
linreg(define=meaneq) vw
# constant vw{1}
compute gstart=%regstart()+q,gend=%regend()
compute b=%beta
*
nonlin(parmset=meanparms) b
```

For the likelihood calculation, we need the value for each expanded regime, so the loop inside the `ARCHRegimeTF` function runs over the range from 1 to `NEXPAND`. The `%MSLagState` function is defined when you do the `@MSSETUP` with a non-zero value for the `LAGS` option. `%MSLagState(SX, LAG)` maps the expanded regime number `SX` and lag number `LAG` to the base regime represented by `SX` at that lag (with 0 for `LAG` meaning the current regime).

```
function ARCHRegimeTF time e
type vector    ARCHRegimeTF
type real      e
type integer    time
*
local integer  i k
local real     h
*
dim ARCHRegimeTF(nexpand)
```

```

do i=1,nexpand
  compute h=1.0
  do k=1,q
    compute h=h+a(k)*uu(time-k)/gv(%MSLagRegime(i,k))
  end do k
  compute h=h+xi*$
  %if(u(time-1)<0.0,uu(time-1)/gv(%MSLagRegime(i,1)),0.0)
  compute ARCHRegimeTF(i)=%if(h>0,$
    exp(%logtdensity(h*gv(%MSLagRegime(i,0)),e,nu)),0.0)
end do i
end

```

We'll demonstrate both maximum likelihood and MCMC estimation of the model. EM has no real computational advantage over ML for a highly nonlinear model like this, and has the disadvantage of slow convergence.

13.1.1 Estimation by ML

We could use either a **GARCH** instruction or a **LINREG** to get guess values for the parameters. Since it's simpler to get the information out, we'll use **LINREG**. Because we need lagged residuals for the ARCH function, we'll shift the start of estimation for the SWARCH model up q periods at the front end. As with any ARCH or GARCH model done using **MAXIMIZE**, we need to keep series of squared residuals and also (to handle the asymmetry) the residuals themselves so we can compute the needed lags in the variance equation.

```

clear uu u
*
set uu = %seesq
set u = %resids
frml uf = %eqnrvalue(meaneq,t,b)
*
* For the non-switching parts of the ARCH parameters
*
compute a=%const(0.05),xi=0.0,nu=10.0

```

We initialize the **GV** array by using a fairly wide scale of the least squares residual variance. Note that the actual model variance after a large outlier can be quite a bit higher than the largest value of **GV** because of the ARCH terms.

```

compute gv=%seesq*||0.2,1.0,5.0||

```

Finally, we need guess values for the transition. Since we're using the logistic indexes, we input the **P** matrix first, since it's more convenient, and invert it to get **THETA**.


```
compute p=|.8, .1, .05|.15, .8, .15|
compute theta=%msplogistic(p)
```

The log likelihood is generated recursively by the following formula:

```
frml logl = u(t)=uf(t), uu(t)=u(t)^2, f=ARCHRegimeTF(t, u(t)), $
      fpt=%MSProb(t, f), log(fpt)
```

This evaluates the current residual using the U_F formula, saves the residual and its square, evaluates the likelihoods across the expanded set of regimes, then uses the `%MSProb` function to update the regime probabilities and compute the final likelihood. The log of the return from `%MSProb` is the desired log likelihood.

Estimation is carried out with the standard:

```
@MSFilterInit
maximize(parmset=meanparms+archparms+msparms, $
      start=(p=%mslogisticp(theta), pstar=%MSInit()), $
      method=bfgs, iters=400, pmethod=simplex, peters=5) logl gstart gend
```

Note that this takes a fair bit of time to estimate. The analogous Cai form would take about 1/9 as long, since this has a 27 branch likelihood while Cai's would just have 3.

The smoothed probabilities of the regimes are computed and graphed (Figure 13.1) with:

```
@MSSmoothed gstart gend psmooth
set p1 = psmooth(t) (1)
set p2 = psmooth(t) (2)
set p3 = psmooth(t) (3)
graph(max=1.0, style=stacked, footer=$
      "Probabilities of Regimes in Three Regime Student-t model") 3
# p1
# p2
# p3
```

For the complete program, see Example 13.1. Tables 13.1 and 13.2 compare the results for the MS-ARCH model with a corresponding GARCH. As one might expect, the use of the switching variance regimes rather dramatically reduces the need for the GARCH recursion—the **A** parameters are barely bigger than zero in the switching model, while in the GARCH the variance recursion parameters are close to the unstable region. The degrees of freedom on the t isn't quite as extreme in the switching case—since it has that high variance regime, an outlier is always going to be seen as a greater possibility than it will be in the GARCH, where the variance won't increase until *after* the outlier. However, with over 1000 data points, the gain for adding 8 additional parameters might

Table 13.1: MS-ARCH Model

MAXIMIZE - Estimation by BFGS					
Convergence in 62 Iterations. Final criterion was 0.0000000 <= 0.0000100					
Weekly Data From 1962:07:24 To 1987:12:29					
Usable Observations		1328			
Function Value		-2804.7006			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	B(1)	0.3496	0.0463	7.5473	0.0000
2.	B(2)	0.2500	0.0274	9.1080	0.0000
3.	A(1)	0.0277	0.0353	0.7837	0.4332
4.	A(2)	0.1168	0.0422	2.7691	0.0056
5.	GV(1)	0.5684	0.1146	4.9576	0.0000
6.	GV(2)	2.4696	0.2884	8.5622	0.0000
7.	GV(3)	7.4693	1.2095	6.1753	0.0000
8.	XI	0.4239	0.0874	4.8517	0.0000
9.	NU	7.2393	1.3455	5.3805	0.0000
10.	THETA(1,1)	14.1288	12.7001	1.1125	0.2659
11.	THETA(2,1)	9.2619	12.7260	0.7278	0.4667
12.	THETA(1,2)	-12.8107	20.3173	-0.6305	0.5283
13.	THETA(2,2)	4.7386	0.7167	6.6115	0.0000
14.	THETA(1,3)	-5.9411	1.0700	-5.5525	0.0000
15.	THETA(2,3)	-4.2044	0.7676	-5.4774	0.0000

Table 13.2: GARCH model on Hamilton-Susmel data

GARCH Model - Estimation by BFGS					
Convergence in 23 Iterations. Final criterion was 0.0000010 <= 0.0000100					
Dependent Variable Y					
Weekly Data From 1962:07:24 To 1987:12:29					
Usable Observations		1328			
Log Likelihood		-2824.0513			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	Constant	0.3105	0.0497	6.2420	0.0000
2.	Y{1}	0.2736	0.0287	9.5454	0.0000
3.	C	0.2819	0.1261	2.2357	0.0254
4.	A	0.0726	0.0298	2.4381	0.0148
5.	B	0.7757	0.0578	13.4132	0.0000
6.	D	0.2269	0.0654	3.4683	0.0005
7.	Shape	5.6077	0.8139	6.8899	0.0000

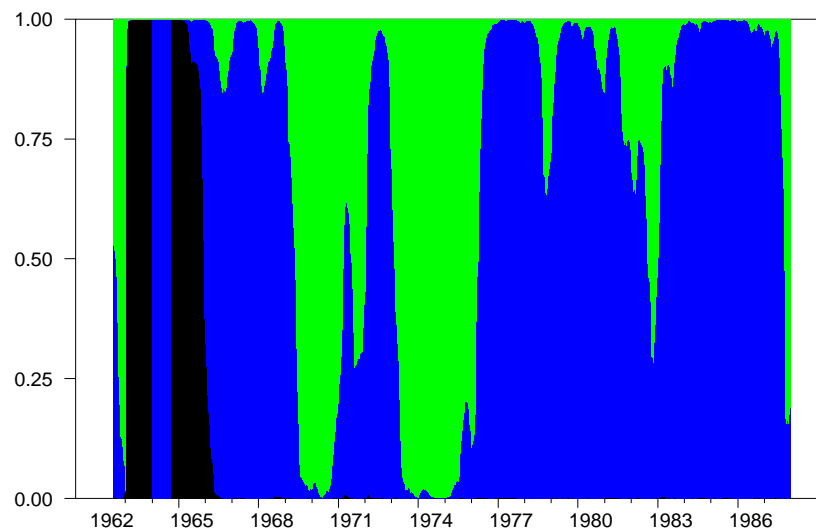


Figure 13.1: Smoothed Probabilities from Hamilton-Susmel Model

not be seen as worth it—the AIC is better with the switching model, while the BIC is better with the GARCH.

13.1.2 Estimation by MCMC

We'll do a somewhat simpler model with just two regimes (rather than three) and with conditionally Gaussian rather than t errors. The function to return the `VECTOR` of likelihoods across expanded regimes is almost identical for Gaussian errors to the one with t errors, requiring a change only to the final line that computes the likelihood given the error and variance:

```

function ARCHRegimeGaussF time e
type vector    ARCHRegimeGaussF
type real      e
type integer    time
*
local integer  i k
local real     hi
*
dim ARCHRegimeGaussF(nexpand)

do i=1,nexpand
  compute hi=1.0
  do k=1,q
    compute hi=hi+a(k)*uu(time-k)/gv(%MSLagRegime(i,k))
  end do k
  compute hi=hi+xi*$
  %if(u(time-1)<0.0,uu(time-1)/gv(%MSLagRegime(i,1)),0.0)
  compute ARCHRegimeGaussF(i)=%if(hi>0,$
    exp(%logdensity(hi*gv(%MSLagRegime(i,0)),e)),0.0)
end do i
end

```

There will remain two ARCH lags. Because of the way the HS model works, the FFBS procedure will need to work with augmented regimes using current and two lags. `@MSSample` adjusts the output regimes, sampling the full set of augmented regimes, but copying out only the current regime into `MSRegime`.

```

@MSFilterInit
do time=gstart,gend
  compute u(time)=uf(time),uu(time)=u(time)^2
  compute ft=ARCHRegimeGaussF(time,u(time))
  @MSFilterStep time ft
end do time
@MSSample gstart gend MSRegime

```

The transition probabilities are drawn with the standard:

```
@MSDrawP(prior=gprior) gstart gend p
```

We need to sample the mean parameters, the ARCH lag parameters and the variance inflation factors. It probably is best to do them as those three groups rather than breaking them down even further.² You might think that you could do the mean parameters as a relatively simple weighted least squares sampler taking the variance process as given. Unfortunately, that isn't a proper

²Tsay (2010) has an example (TSAYP663.RPF) which does Gibbs sampling on a switching GARCH model. He uses single-move sampling and does each parameter separately, using the very inefficient process of "griddy Gibbs."

procedure— h is a deterministic function of the residuals, and so can't be taken as given for estimating the parameters controlling the residuals. Instead, both the mean parameters and the ARCH parameters require Metropolis within Gibbs (Appendix D). For this, we need to be able to compute the full sample log likelihood given a test set of parameters at the currently sampled values of the regimes. This is similar to **ARCHRegimeGaussF** except it returns the log, and doesn't have the loop over the expanded set of regimes, instead doing the calculation using the current sampled values of **MSRegime**:

```
function MSARCHLogl time
type integer time
*
local integer k
local real    hi
*
compute u(time)=uf(time)
compute uu(time)=u(time)^2
compute hi=1.0

do k=1,q
    compute hi=hi+a(k)*uu(time-k)/gv(MSRegime(time-k))
end do k
compute hi=hi+xi*$
    %if(u(time-1)<0.0,uu(time-1)/gv(MSRegime(time-1)),0.0)
compute MSARCHLogl=%if(hi>0,$
    %logdensity(hi*gv(MSRegime(time)),u(time)),%na)
end
```

To get a distribution for doing Random Walk M-H, we run the maximum likelihood ARCH model. We take the ML estimates as the initial values for the Gibbs sampler, and use the Cholesky factor of the proper segment of the ML covariance matrix as the factor for the covariance matrix for the random Normal increment. This is the initialization code for drawing the mean parameters:

```
garch(q=2,equation=meaneq,asymmetric) gstart gend
*
compute msb=%xsubvec(%beta,1,%nregmean)
compute far=%decomp(%xsubmat(%xx,1,%nregmean,1,%nregmean))
compute aaccept=0
```

and this is the corresponding code for drawing mean parameters inside the loop:

```

sstats gstart gend msarchlogl(t)>>logplast
compute blast=msb
compute msb =blast+%ranmvnormal(far)
sstats gstart gend msarchlogl(t)>>logptest
compute alpha=exp(logptest-logplast)
if %ranflip(alpha) {
    compute logplast=logptest
    compute aaccept=aaccept+1
}
else
    compute msb=blast

```

First off, this computes the sample log likelihood given the current values for the regimes and all the parameters. It then saves the current values for the mean parameters (into `BLAST`) and draws a test set by adding a Normal increment to the current values. The test set has to go into `MSB` since those are the parameters used by `MSARCHLog1`. The sample log likelihood is then recomputed at the test set. The Metropolis condition is evaluated and if the draw is acceptable, we keep it (and increment the count of accepted draws); if not, we copy the previous set back into `MSB`. Note that the log likelihood at whatever is the set of parameters we take is in `LOGPLAST`; that way, we don't need to recompute it when we do the procedure for drawing the ARCH parameters.

The setup code for drawing the ARCH parameters (immediately after the `GARCH` instruction) is:

```

compute a=%xsubvec(%beta,%nregmean+2,%nregmean+1+q), $
        xi=%beta(%nregmean+2+q)
compute farch=%decomp(%xsubmat(%xx,%nregmean+2,%nregmean+2+q,$
                                %nregmean+2,%nregmean+2+q))
compute [vector] msg=a~~xi
compute gaccept=0

```

`A` is the `VECTOR` of ARCH lag parameters and `XI` is the asymmetry coefficient. We'll draw them as a unit, using the `VECTOR MSG` to group them together when we do the draws. We'll copy them back into `A` and `XI` when we're done with a draw.

This is the code for drawing the ARCH parameters. This is also set to reject any test set with negative lag parameters. Whether that's a good idea with an ARCH model isn't clear—if a fair number of draws for a parameter end up negative, it might be taken as a sign that the model is overparameterized or just generally inadequate, information which is lost if the negative draws are rejected out of hand. Other than that and the need to copy information out into the working `A` and `XI`, this is basically identical to the procedure for the mean parameters.

```

compute glast=msg
:redrawg
compute msg=glast+%ranmvnormal(farch)
compute a=%xsubvec(msg,1,q),xi=msg(q+1)
if %minvalue(a)<0.0
    goto redrawg
sstats gstart gend msarchlogl(t)>>logptest
compute alpha=exp(logptest-logplast)
if %ranflip(alpha) {
    compute logplast=logptest
    compute gaccept=gaccept+1
}
else
    compute msg=glast
compute a=%xsubvec(msg,1,q),xi=msg(q+1)

```

The methods used for drawing the mean and ARCH parameters are quite commonly used in non-linear models, and are generally very effective. Sometimes it's necessary to tweak the increment distribution by scaling up the covariance matrix, or by switching to a fatter-tailed t rather than the Normal, but usually not with such a small set of free parameters in each grouping.

The variance inflation factors are a bit trickier. They can't be treated like the variances in Examples 8.3 and 9.3 because they're more firmly tied into the rest of the model. And, because they are (of necessity) positive, a Normal or t increment random walk isn't fully appropriate.³ Instead of a "random walk", we'll use mean one random multipliers, drawn as random chi-squares normalized to unit means. This requires a relatively simple correction factor for the log ratio of the probability of moving to the probability of reversing the move. Unlike the other parameters, it isn't clear *a priori* what the appropriate spread is for the increments. χ^2_ν/ν has mean 1 and variance $2/\nu$. A value like $\nu = 10$ is almost certainly too small, as almost half of all moves will be on the order of 25% up or down, which is likely to be too sizeable to be accepted on a regular basis. After some experimenting (looking at the acceptance percentage), we ended up with $\nu = 50$, though almost anything in the 20-50 range will give fairly similar performance. The set up keys off the constant in the variance equation from the **GARCH** instruction, scaling up and down from that:

```

compute gv(1)=%beta(%nregmean+1)*0.5
compute gv(2)=%beta(%nregmean+1)*2.0
dec vect nuv(nregimes) fiddle(nregimes)
ewise nuv(i)=50.0
compute vaccept=0

```

The code for drawing is:

³Although a Normal increment in the vector of log variances would be possible.

```

compute glast=gv
ewise fiddle(i)=%ranchisqr(nuv(i))/nuv(i)
compute logqratio=-2.0*%sum(%log(fiddle))
compute gv=glast.*fiddle
sstats gstart gend msarchlogl(t)>>logptest
compute alpha=exp(logptest-logplast-logqratio)
if %ranflip(alpha) {
    compute logplast=logptest
    compute vaccept=vaccept+1
}
else
    compute gv=glast

```

This completes the sampler (other than the re-labeling code, which switches regimes based upon the order of the GV). We need to check how our various M-H draws worked, which is done outside the draw loop:

```

disp "Acceptance Percentages"
disp "Mean Parameters" ###.## 100.0*aaccept/ndraws
disp "ARCH Parameters" ###.## 100.0*gaccept/ndraws
disp "Variance Scales" ###.## 100.0*vaccept/ndraws

```

You should always start with a very modest number of draws to check whether your rates are reasonable. Make adjustments if they aren't, running a "production" number of draws only when you're satisfied that you have the sampler set up properly. Note that even though this is able to use Multi-Move sampling, because of the size of the data set (1328 usable observations), it takes quite a while even to do just the 200 burn-in plus 500 keeper draws that we've used in the full example (Example 13.2). The acceptance probabilities should be something like (these are based on random numbers):

Acceptance Percentages	
Mean Parameters	73.00
ARCH Parameters	36.00
Variance Scales	22.60

which are all reasonable for random walk MCMC—the 73% might be a bit higher than one would like (too high an acceptance rate generally means the steps are smaller than they should be for adequate coverage), though there is probably no good reason to be concerned about the shape of the mean parameters, which are unlikely to be affected all that much by the complicated variance calculations. The means and standard deviations of the parameters generated by MCMC are shown in Table 13.3. Both regimes are highly persistent—perhaps the most interesting number in the table is the large value of $MSG(3)$, which is the ξ asymmetry coefficient.

Table 13.3: MS-ARCH MCMC parameter estimates and standard deviations

GV(1)	1.3467	0.1217
GV(2)	8.0180	0.8046
MSB(1)	0.3250	0.0612
MSB(2)	0.2760	0.0336
MSG(1)	0.0317	0.0268
MSG(2)	0.0620	0.0248
MSG(3)	0.4628	0.0872
P(1,1)	0.9515	0.0118
P(2,1)	0.0485	0.0118
P(1,2)	0.0718	0.0215
P(2,2)	0.9282	0.0215

13.2 Markov Switching GARCH

An exact analysis of an MS-GARCH model is nearly impossible (except in very small data sets) because of the dependence of the lagged variance on the entire prior history of the regimes up until that time. Instead, the likelihood can only be approximated by using some method to summarize that history in a finite number of lags. There are two principal “filter” methods that have been proposed: Gray (1996) collapses it right away, so there is just a single lagged variance, while Dueker (1997) collapses it after a one period lag, so each regime at $t - 1$ has its own variance. Of the two, Dueker’s is actually simpler to use in practice and more general. For instance, Gray’s filter requires extra calculation if the residual is also regime-dependent (if the mean equation switches) because the lagged variance calculation needs to take into account the differing means, and the lagged squared residual terms also must be collapsed. Because Dueker’s filter keeps one lag of the regime, those terms can be handled the way they are in the simpler MS-ARCH. Note that, although a least one paper has claimed that Dueker and Gray were effectively the same, that is most definitely *not* true.

A basic regime-switching GARCH model can be written

$$\begin{aligned}
 y_t &= \mu_t(S_t) + \varepsilon_t(S_t) \\
 \text{var}(\varepsilon_t(S_t)|t-1) &= h_t(S_t, S_{t-1}, \dots) \\
 h_t(S_t, S_{t-1}, \dots) &= c(S_t) + a(S_t)\varepsilon_{t-1}^2(S_{t-1}) + b(S_t)h_{t-1}(S_{t-1}, S_{t-2}, \dots)
 \end{aligned} \tag{13.2}$$

One thing to note is that the “infinite” memory doesn’t depend upon the GARCH coefficients themselves being regime-dependent. Even if the a , b and c are fixed, if the mean model is MS, the regime-dependent ε_{t-1}^2 feeds through to affect all future values of h , producing the same situation as with regime-dependent GARCH coefficients.

A likelihood based upon (13.2) directly will be infeasible for any data set with a standard size for GARCH modelling, since with 2 regimes and T data points, there are 2^T branches to it. However, even if the h value theoretically depends

upon all previous data points, for practical purposes, the distant past has relatively little effect, as the weights of the h decline exponentially and the probabilities of most of the regime sequences are quite small. Thus, we can expect that it will be possible to at least approximate the likelihood by somehow collapsing the information so an approximate h depends upon a shortened regime history.

The literature on MS-GARCH has largely chosen Gray's filter over Dueker's, without any particularly strong *statistical* justification. Aside from the difference in the filter calculation,

1. following Hamilton and Susmel, Dueker used a more tightly parameterized model, allowing only a limited number of "switching" parameters, while Gray allowed all parameters to switch.
2. Dueker allowed for conditionally t errors while Gray assumed conditionally Normal errors.
3. Dueker used daily S&P 500 returns, while Gray worked with weekly US T-bill yields.

Neither (1) nor (2) is required for either general method—Dueker's filter can be applied with more broadly switching models and Gray's can be used with t errors or a tighter model. Perhaps an important reason why Dueker's method has been largely ignored is that his application was (it is probably safe to say) considerably less successful—he found no obvious breakdown into regimes in the stock market returns, while the T-bill rate over the period covered by the Gray paper (1970-1994) had some fairly dramatic (and readily identifiable) changes. Here, we'll do a comparison of the two applied to the same data set (Gray's).

Gray's filter eliminates the regime-dependence immediately, by using a GARCH recursion of

$$h_t(S_t) = c(S_t) + a(S_t)\tilde{\varepsilon}_{t-1}^2 + b(S_t)\tilde{h}_{t-1}$$

where $\tilde{\varepsilon}_{t-1}^2$ and \tilde{h}_{t-1} don't depend upon the regimes.⁴ To compute $\tilde{\varepsilon}_t$, Gray goes back to what would be the regime-free definition of that (basically, what it is for a simple GARCH model), which is

$$\tilde{\varepsilon}_t = y_t - E(y_t | t-1)$$

With Markov switching, the conditional expectation is the probability-weighted sum of the regime-specific values:

$$E(y_t | t-1) = \sum_{\{S_t\}} P(S_t | t-1) \mu_t(S_t) \quad (13.3)$$

⁴The use of the \sim is *our* notation.

Similarly, the \tilde{h} is computed as the conditional variance of y :

$$\begin{aligned} E((y_t - E(y_t|t-1))^2|t-1) &= E(y_t^2|t-1) - E(y_t|t-1)^2 \\ &= E\{((\mu_t(S_t) + \varepsilon_t(S_t))^2|S_t)|t-1\} - E(y_t|t-1)^2 \\ &= \sum_{\{S_t\}} P(S_t|t-1) (\mu_t^2(S_t) + h_t(S_t)) - E(y_t|t-1)^2 \end{aligned} \quad (13.4)$$

By contrast, Dueker uses both the current and one lagged regime, so

$$\begin{aligned} h_t(S_t, S_{t-1}) &= c(S_t) + a(S_t)\varepsilon_{t-1}^2(S_{t-1}) + b(S_t)\tilde{h}(S_{t-1}) \\ \tilde{h}(S_t) &= \sum_{\{S_{t-1}\}} p(S_{t-1}|S_t, t) h(S_t, S_{t-1}) \end{aligned} \quad (13.5)$$

that is, the collapsed (now regime-dependent) variance is the probability-weighted average for h using the *filtered* (rather than predicted, as in Gray) conditional probabilities of the lagged regime. An alternative way to write the definition of \tilde{h} is as the implicit condition (as a function of S_t):

$$\sum_{\{S_{t-1}\}} p(S_t, S_{t-1}|t) (h(S_t, S_{t-1}) - \tilde{h}(S_t)) = 0$$

To motivate this method of collapsing the variance,⁵ let $g(S_t, \varepsilon_t(S_t), h_{t-1}(S_{t-1}, \dots))$ represent the (true) likelihood at t conditional on the full history of the regimes. Suppose we assume

$$h_{t-1}(S_{t-1}, S_{t-2}, \dots) \approx \tilde{h}_{t-1}(S_{t-1})$$

that is, we have a single value that can approximate the variances for all the combinations with the same setting for S_{t-1} . Integrating across regimes, we have a likelihood of

$$\begin{aligned} \sum_{\{S_t, S_{t-1}, \dots\}} p(S_t, S_{t-1}, \dots|t-1) g(S_t, \varepsilon_t(S_t), h_{t-1}(S_{t-1}, \dots)) &= \\ \sum_{\{S_t, S_{t-1}, \dots\}} p(S_t|S_{t-1}) p(S_{t-1}, \dots|t-1) g(S_t, \varepsilon_t(S_t), h_{t-1}(S_{t-1}, \dots)) \end{aligned}$$

which can be rearranged to

$$\sum_{\{S_t, S_{t-1}\}} p(S_t|S_{t-1}) \sum_{\{S_{t-2}, \dots\}} p(S_{t-1}, S_{t-2}, \dots|t-1) g(S_t, \varepsilon_t(S_t), h_{t-1}(S_{t-1}, \dots)) \quad (13.6)$$

For any specific S_t, S_{t-1} , by linearizing around $\tilde{h}_{t-1}(S_{t-1})$, the sum over S_{t-2}, \dots can be approximated by the sum of

$$\sum_{S_{t-2}, \dots} p(S_{t-1}, S_{t-2}, \dots|t-1) g(S_t, \varepsilon_t(S_t), \tilde{h}_{t-1}(S_{t-1})) \quad (13.7)$$

⁵Dueker describes it as marginalizing out the lagged regimes in a probability-weighted average of the h values, but never really explains why the probability-weighted averages of the h 's matters for the calculation.

and

$$\sum_{S_{t-2}, \dots} p(S_{t-1}, S_{t-2}, \dots | t-1) \times g' \left(S_t, \varepsilon_t(S_t), \tilde{h}_{t-1}(S_{t-1}) \right) \left(h_{t-1}(S_{t-1}, S_{t-2}, \dots) - \tilde{h}_{t-1}(S_{t-1}) \right) \quad (13.8)$$

where the derivative is with respect to $\tilde{h}_{t-1}(S_{t-1})$. Since $g' \left(S_t, \varepsilon_t(S_t), \tilde{h}_{t-1}(S_{t-1}) \right)$ is a constant inside the sum in (13.8), if we define $\tilde{h}_{t-1}(S_{t-1})$ to solve

$$\sum_{S_{t-2}, \dots} p(S_{t-1}, S_{t-2}, \dots | t-1) \left(h_{t-1}(S_{t-1}, S_{t-2}, \dots) - \tilde{h}_{t-1}(S_{t-1}) \right) = 0$$

which is (in effect) a re-dated version of the definition for \tilde{h} from (13.5), (13.8) becomes zero. In (13.7), the g depends only upon S_{t-1} , so that simplifies to just

$$p(S_{t-1} | t-1) g \left(S_t, \varepsilon_t(S_t), \tilde{h}_{t-1}(S_{t-1}) \right)$$

Inserting that back into (13.6), gives

$$\sum_{S_t, S_{t-1}} p(S_t | S_{t-1}) p(S_{t-1} | t-1) g \left(S_t, \varepsilon_t(S_t), \tilde{h}_{t-1}(S_{t-1}) \right)$$

which is the Dueker filtered likelihood.

If we compare the two methods, we can see two important reasons why the Gray filter should be suspect. First, it ignores the lagged regime in handling the ε_{t-1}^2 term, even though that depends only upon the one lag and so can be handled exactly. Second, the use of the predicted (rather than filtered) probabilities in (13.3) is unnecessary in doing the GARCH recursion. While Dueker's filter can reasonably be interpreted as an approximation to the actual likelihood of a true MS-GARCH model, Gray's would better be described as a Markov Switching approximate-GARCH model.

13.2.1 The Example

We'll apply the two filtering methods to the model in Gray's paper. (As noted above, Dueker's paper never really found any identifiable regimes in his data set). The model is

$$r_t - r_{t-1} = \alpha_0(S_t) + \alpha_1(S_t)r_{t-1} + u_t$$

where r is weekly data on the one month T-bill rates (Figure 13.2). Gray fits a number of different models to this, including fixed regime least squares, fixed regime GARCH, and a MS variance regime model (such as in Section 8.3.1). You can check the Gray replication programs for those. We'll concentrate on a two-regime Markov Switching GARCH model with conditionally Gaussian errors.

Example 13.3 uses Dueker's filter and 13.4 uses Gray's.

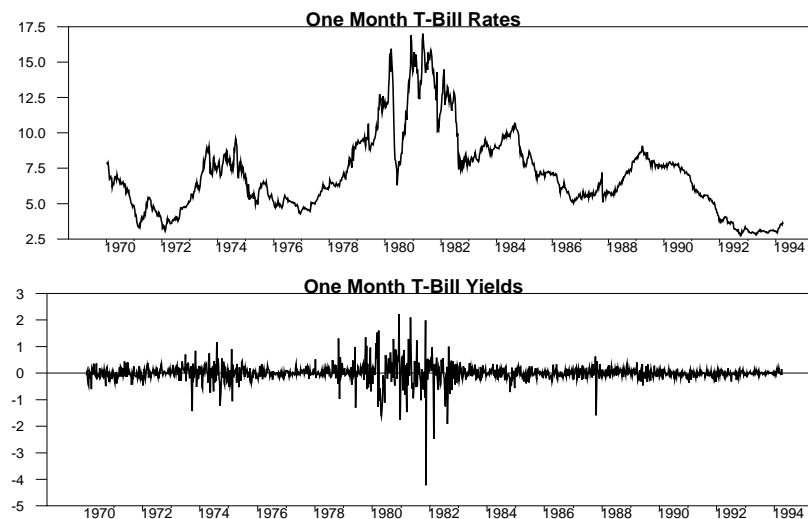


Figure 13.2: T-Bill Data from Gray

13.2.2 Dueker Filter

Because the Dueker filter is simpler to implement, we have a set of procedures which does most of the work. First, we estimate the model by least squares and define an `EQUATION` (called `MEANEQ`).

```
linreg(robust,define=meaneq) drate
# constant rate{1}
```

The example also does a fixed regime GARCH estimate for comparison:

```
garch(p=1,q=1,equation=meaneq,hseries=hgarch)
```

Similar to the various support procedures for Markov Switching regressions, etc. there's a `@DFSetup` procedure which does the setup. In the case here, where we will be allowing the mean equation to switch and allowing full switching on the GARCH variance model, we will use:

```
@DFSetup(equation=meaneq,regimes=2,mean=switching,garch=fullswitch)
```

The choices for the `MEAN` option are `FIXED` (default) and `SWITCHING` and the choices for `GARCH` are `FULL` (default), `HS` (for switching Hamilton-Susmel variance inflation factors, other parameters fixed) and `CAO` (for Cao switching variance intercepts, other parameters fixed).

`@DFInit` does initialization of the parameters (and is needed for other setup). The standard “labeling” is variances running from low to high, which is the opposite of the ordering in Gray’s paper.

```
@DFInit
compute gstart=%regstart(),gend=%regend()
```

The `@DFMeanParmset` and `@DFGARCHParmset` procedures put together the proper parameter sets of the mean model and the variance model given the options in use. `FIXPPARMS` is setup with the parameters for the transition probabilities.

```
@DFMeanParmset (parmset=meanparms)
@DFGARCHParmset (parmset=garchparms)
nonlin (parmset=fixpparms) p
```

The hard work of the Dueker filter is done by the `DFLogL` function which does the calculation for one entry:

```
function DFLogL time
type integer                time
*
local real                  e
local vector                fvec evec hvec ph
*
dim fvec(nexpand) evec(nregimes)
*
* Compute the "H" matrices for each expanded regime
*
compute hvec=msgarchfnc(time)
*
* Compute the residuals for each current regime
*
compute evec=GARCHRegimeResids(time)

*
* Compute the likelihood for each expanded regime.
*
do i=1,nexpand
    compute e=evec(%MSLagRegime(i,0))
    compute fvec(i)=msgarchll(msgarchvarf(hvec,i),e,0.0)
end do i
```

```

*
* Do a filter step to update the probabilities. The log
* likelihood element is returned in the LOGF option.
*
@MSFilterStep(logf=DFLogL) time fvec
*
* Take probability weighted averages of the h's for each regime.
*
compute ph=pt_t(time).*hvec
compute pdiv  =%sumc(%reshape(pt_t(time),nregimes,MSGFilterSize))
compute phstar=%sumc(%reshape(ph,nregimes,MSGFilterSize))./pdiv
*
* Push them out into the HDF vector
*
compute %pt(MSGHF ,time,phstar)
compute %pt(MSGUUF,time,evec.^2)
compute %pt(MSGUUSF,time,evec.*%minus(evec))
end

```

The HVEC is a VECTOR of the “H” matrices for all combinations of $\{S_t, S_{t-1}\}$. The EVEC is the VECTOR of residuals for each S_t . The FVEC converts those into the likelihoods for the $\{S_t, S_{t-1}\}$ combinations and the **MSFilterStep** will compute the filtered probabilities of $\{S_t, S_{t-1}\}$ and produces the log likelihood element based upon that. Once that is, done, the HVEC is collapsed to PHSTAR by marginalizing out the lagged regimes and that is put into the element of the `VECT[SERIES] MSGHF`, which will supply the values needed in the next time period. The squared and sign constrained squares of the residuals are pushed out into their own `VECT[SERIES]`, again for use in the recursion for the next period.

With a guess of fairly persistent regimes, this does the estimation:

```

compute p=||.90,.10||
*
frml LogDFFilter = DFLogL(t)
*
maximize(start=DFStart(),parmset=meanparms+garchparms+fixpparms,$
  pmethod=simplex,piters=5,method=bfgs) logDFFilter gstart gend

```

which provides a rather reasonable looking set of results (Table 13.4) and Figure 13.3), which shows that the switching model largely picks up as the “high” variance regimes the anti-inflation periods of 1973-1974 and 1979-1981. It also fits quite a bit better than the corresponding fixed GARCH model (which has a log likelihood of -129.75). Here (unlike Example 13.1), the BIC favors of the switching model over the fixed GARCH.

However, it turns out that the (approximate) log likelihood is multi-modal, in a big way. Guess values which are “anti-persistent”, combined with a broad

Table 13.4: MS-GARCH with persistent mode

MAXIMIZE - Estimation by BFGS					
Convergence in 33 Iterations. Final criterion was 0.0000070 <= 0.0000100					
Weekly Data From 1970:01:14 To 1994:04:13					
Usable Observations		1266			
Function Value		161.7108			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	MSGBETAS(1)(1)	0.0012	0.0134	0.0868	0.9308
2.	MSGBETAS(1)(2)	0.0000	0.0023	0.0066	0.9948
3.	MSGBETAS(2)(1)	0.3032	0.1250	2.4262	0.0153
4.	MSGBETAS(2)(2)	-0.0261	0.0114	-2.2835	0.0224
5.	GARCHS(1)(1)	0.0020	0.0006	3.1332	0.0017
6.	GARCHS(1)(2)	0.1744	0.0361	4.8310	0.0000
7.	GARCHS(1)(3)	0.7432	0.0445	16.6903	0.0000
8.	GARCHS(2)(1)	0.1672	0.0468	3.5702	0.0004
9.	GARCHS(2)(2)	0.6490	0.1979	3.2795	0.0010
10.	GARCHS(2)(3)	0.1240	0.0910	1.3628	0.1729
11.	P(1,1)	0.9954	0.0024	418.9162	0.0000
12.	P(1,2)	0.0187	0.0099	1.8903	0.0587

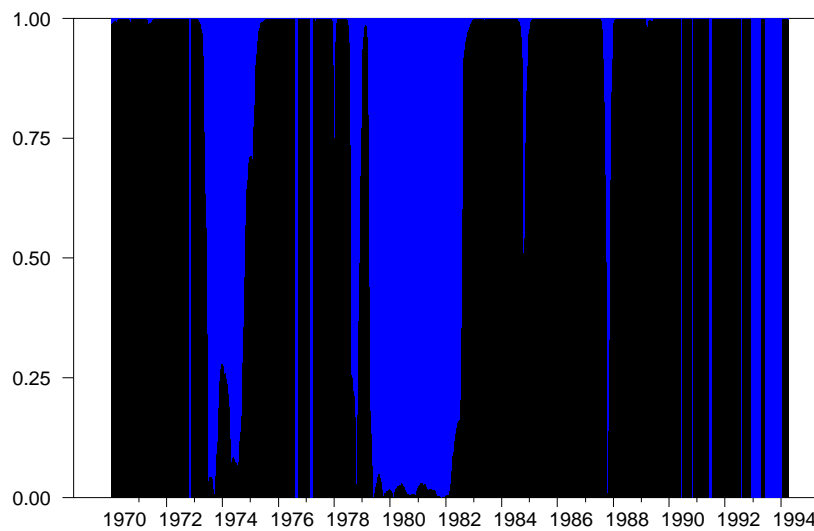
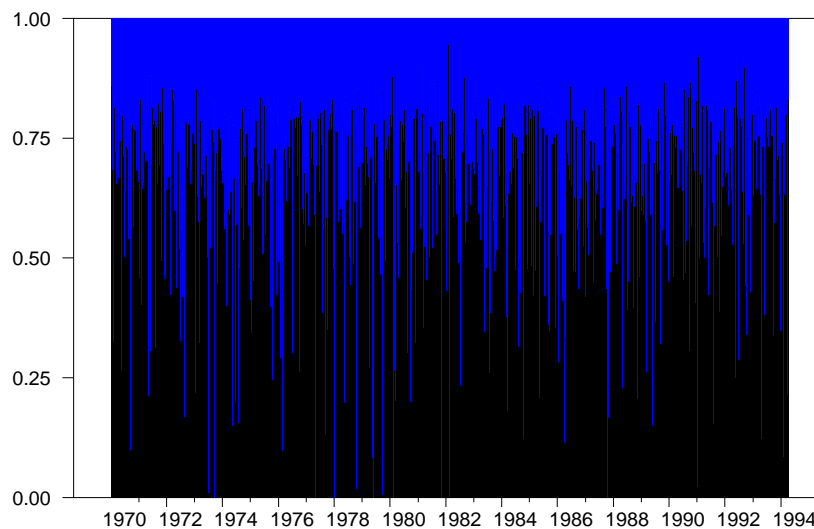
**Figure 13.3:** MS-GARCH smoothed probabilities with persistent mode

Table 13.5: MS-GARCH with anti-persistent mode

MAXIMIZE - Estimation by BFGS					
Convergence in 60 Iterations. Final criterion was 0.0000044 <= 0.0000100					
Weekly Data From 1970:01:14 To 1994:04:13					
Usable Observations		1266			
Function Value		206.0630			
	Variable	Coeff	Std Error	T-Stat	Signif
1.	MSGBTAS(1)(1)	-0.0111	0.0115	-0.9581	0.3380
2.	MSGBTAS(1)(2)	0.0007	0.0022	0.3426	0.7319
3.	MSGBTAS(2)(1)	0.0303	0.0393	0.7712	0.4406
4.	MSGBTAS(2)(2)	-0.0036	0.0060	-0.6083	0.5430
5.	GARCHS(1)(1)	0.0002	0.0003	0.7022	0.4825
6.	GARCHS(1)(2)	-0.0200	0.0114	-1.7586	0.0786
7.	GARCHS(1)(3)	0.4850	0.0388	12.5050	0.0000
8.	GARCHS(2)(1)	0.0004	0.0008	0.4253	0.6706
9.	GARCHS(2)(2)	-0.2034	0.0327	-6.2218	0.0000
10.	GARCHS(2)(3)	2.5833	0.2058	12.5521	0.0000
11.	P(1,1)	0.5046	0.0603	8.3614	0.0000
12.	P(1,2)	0.8131	0.0426	19.0910	0.0000

preliminary search with PMETHOD=GA (Genetic Annealing) produces Table 13.5 and Figure 13.4.

```
@DFInit
compute p=||.30,.70||
maximize(start=DFStart(),parmset=meanparms+garchparms+fixpparms,$
  pmethod=ga,piters=5,method=bfgs) logDFFilter gstart gend
```

**Figure 13.4:** MS-GARCH smoothed probabilities with anti-persistent mode

If the log likelihood were similar to the first set of estimates, we could probably reject this in favor of the better-behaved one based upon prior beliefs about the

persistence of the regimes. However, the log likelihood difference is very large (bigger than the gap between the switching and non-switching GARCH models) and this set of estimates is almost impossible to interpret—the fit ends up being a close to a mixture (Chapter 7) of two separately rather ill-behaved regimes. Regime 2 has a GARCH recursion coefficient of 2.58, so it would (if it were in place for several periods) produce an exploding variance. Of course, with $p_{2,2}$ being about .19, it *won't* be in place for long.

What can one make of this? First, note that MS models *can* pick up neglected non-linearities. Gray's paper includes a GRS model which has the variance “constant” in the GARCH replaced by $\sigma^2 r_{t-1}$; that is, the variance in the changes goes up with the interest rate. If that were correct, that would be a neglected non-linearity and, because r is (rather strongly) serially correlated, a MS model that didn't include it could seem to find “regimes” based upon the value of r_{t-1} . This is not good, because the breakdown into regimes would be spurious. That seems unlikely to be the problem here, since the model isn't even identifying any regimes *at all*. It may also be that (as was warned in Hamilton and Susmel), this has too many moving parts to create a reasonable model.

Another possibility is that the approximation is failing, and failing rather badly. The assumption underlying the collapsing of the regimes is that the single lagged regime can reasonably approximate all regime histories that lead to it. More to the point, that it can reasonably approximate those regime histories that have a non-trivial probability. For a process with two very persistent regimes, such as in Table 13.4, this is probably not an unreasonable assumption, since the probability will be heavily weighted towards just one set of regime histories for quite a few periods back. With the transition probabilities in Table 13.5, that's no longer the case at all, and the wild GARCH recursion in regime 2 may help explain some outliers while not ever being “penalized” for the explosive effect on the future variance predictions that it would have if we would calculate through fully. A method of examining this more carefully is a particle filter. This uses a cloud of “particles” to represent the state of an otherwise intractable dynamic model. It's not sufficiently accurate to be used for estimation of the free parameters of a model, but *can* be used given a particular set of parameters to check on the approximation to the variance. At time $t - 1$, the “state” of a particle would consist of its regime, its variance and its residual (squared). Rolling that forward to time t , it would (randomly) transition to the next regime, the variance computed and a residual drawn. Note that the period t variance and the distribution of the residual can be computed *exactly* given the new regime and last period's variance and residual. Figure 13.5 shows the average variance across particles vs the Dueker approximation for each regime during a two year period of fairly extreme variance. In Regime 2, the values are effectively identical; in Regime 1, there's a slight approximation error at the end of 1981, but otherwise the filter seems to work properly. Contrast that with Figure 13.6, which is the same for the anti-persistent mode. As you can see, the particle filter has a number of sizable peaks while the filtered

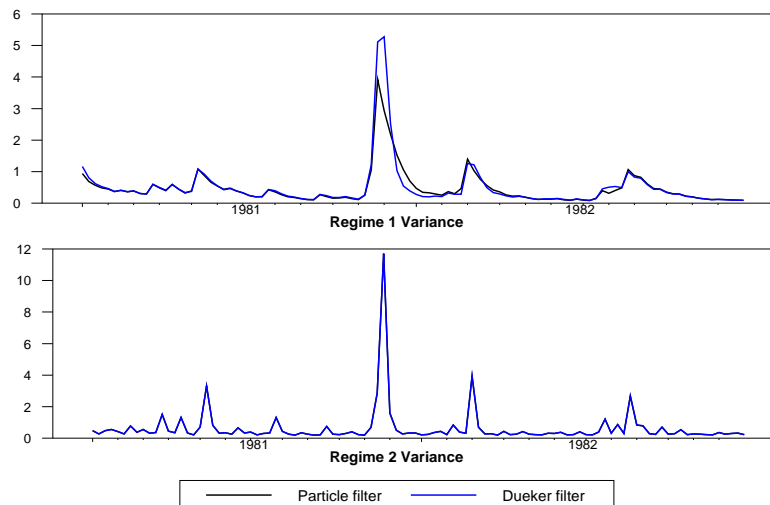


Figure 13.5: Comparison of Approximation, Persistent Mode

estimates are rather flat. Since the particle filter is a reflection of how the variance actually evolves according to the (unapproximated) model, the likelihood of the Dueker approximation will likely be quite a bit better than it should be if those peaks aren't predicting outliers.

13.2.3 Gray Filter

As we described above, Gray uses a different form of collapsing, so at time t , it reduces both the variance and residual to a single regime-free value. Collapsing the residual, in particular, seems to be a poor idea, as the residual *is* observable given the regime. It also means that there's really no obvious extension to deal with asymmetry.

Most of the calculations are the same as with the Dueker filter, except that the Gray filter doesn't need to keep track of the lagged regime. There's a special option (`GRAY`) on the `@DFSetup` that switches the internal calculations to only use the current regime.

```
@DFSetup(equation=meaneq, regimes=2, mean=switching, $
          garch=fullswitch, gray)
```

The function for returning the VECTOR of regime-specific variances (for a fully-switching process) is shown next. Note that the only things that depends upon the regime in this are the coefficients. (The current residuals also are regime-dependent, but they are computed elsewhere).

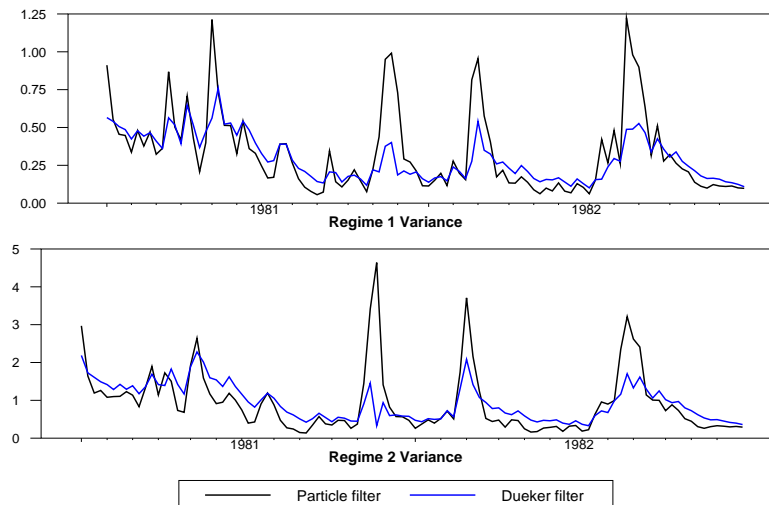


Figure 13.6: Comparison of Approximation, Persistent Mode

```
function GrayRegimeFull time
type vector    GrayRegimeFull
type integer   time
*
local integer i regime0
*
* When we get here, the HDF for the lag has already been collapsed
* so it's independent of the lagged regime.
*

dim GrayRegimeFull(nregimes)
do i=1,nregimes
  compute regime0=%MSLagRegime(i,0)
  compute GrayRegimeFull(i)=garchs(regime0)(1)+$
    garchs(regime0)(3)*MSGHF(1)(time-1)+$
    garchs(regime0)(2)*MSGUUF(1)(time-1)
  if MSGAsymmetric
    compute GrayRegimeFull(i)+=garchs(regime0)(4)*MSGUUSF(1)(time-1)
end do i
end
```

The filter function is

```

function GrayLogL time
type real      GrayLogL
type integer   time
*
local vector   hvec pprep
local vector   fvec(nregimes) evec(nregimes)
local real     e mu mux
local integer  i
*
* Compute regime-dependent variances
*
compute hvec=msgarchfnc(time)

*
* Compute the residuals for each current regime
*
compute evec=GARCHRegimeResids(time)

*
* Compute the likelihood of the regimes
*
do i=1,nregimes
  compute e=evec(i)
  compute fvec(i)=msgarchll(msgarchvarf(hvec,i),e,0.0)
end do i

*
* Do a filter step to update the probabilities. The log
* likelihood element is returned in the LOGF option.
*
@MSFilterStep(logf=GrayLogL) time fvec

*
* Fetch the predicted probability
*
compute ppred=pt_t1(time)

*
* Compute the values of uu (squared residual) and h (variance) to
* be used for the period following.
*
compute MSGHF(1)(time)=0.0
compute mux=0.0

```

```

do i=1,nregimes
  if MSGMeanSwitch
    compute mu=%eqnvalue(meaneq,time,MSGBetas(i))
  else
    compute mu=%eqnvalue(meaneq,time,MSGBeta)
  compute MSGHF(1)(time)+=ppred(i)*(mu^2+hvec(i))
  compute mux+=ppred(i)*mu
end do i

*
compute e=( [series] %eqndepvar(meaneq) ) (time) -mux
compute MSGUUF(1)(time)=e^2
compute MSGHF(1)(time)-=mux^2
*
* It's not clear what this should be
*
compute MSGUUSF(1)(time)=e*%minus(e)
end

```

The bottom part of this is to compute the adjustment to the variance to deal with the time-varying residuals. (If the mean equation is fixed, then the MU adjustments will cancel out).

The following reproduces the results from the paper (with the regimes reversed, Table 13.6). This is quite similar to Table 13.5 except the log likelihood is 10 points lower. Now, there is no test based upon this, since these are just two different approximations rather than being different models. However, it's probably telling that the more accurate approximation produces the higher likelihood.

```

frml LogGrayFilter = GrayLogL(t)
compute msgarchfnc=GrayRegimeFull
*
compute p=||.90,.10||
maximize(start=DFStart(),parmset=meanparms+garchparms+fixpparms,$
  pmethod=simplex,piters=10,method=bfgs,itors=100) $
  logGrayFilter gstart gend

```

What's perhaps even more interesting is if you try to estimate the anti-persistent mode using Gray's filter:

```

@DFInit
compute p=||.30,.70||
compute msgarchfnc=GrayRegimeFull
maximize(start=DFStart(),parmset=meanparms+garchparms+fixpparms,$
  pmethod=ga,piters=5,method=bfgs,itors=100,trace) $
  logGrayFilter gstart gend

```

Table 13.6: MS-GARCH estimates using Gray filter, Persistent model

MAXIMIZE - Estimation by BFGS

Convergence in 28 Iterations. Final criterion was 0.0000092 <= 0.0000100

Weekly Data From 1970:01:14 To 1994:04:13

Usable Observations 1266

Function Value 151.2219

	Variable	Coeff	Std Error	T-Stat	Signif
1.	MSGBTAS(1)(1)	-0.0011	0.0069	-0.1627	0.8707
2.	MSGBTAS(1)(2)	0.0006	0.0011	0.5463	0.5848
3.	MSGBTAS(2)(1)	0.1397	0.1010	1.3837	0.1665
4.	MSGBTAS(2)(2)	-0.0140	0.0099	-1.4128	0.1577
5.	GARCHS(1)(1)	0.0098	0.0018	5.6072	0.0000
6.	GARCHS(1)(2)	0.1648	0.0444	3.7133	0.0002
7.	GARCHS(1)(3)	0.2678	0.0662	4.0474	0.0001
8.	GARCHS(2)(1)	0.1883	0.0513	3.6676	0.0002
9.	GARCHS(2)(2)	0.4547	0.1345	3.3812	0.0007
10.	GARCHS(2)(3)	0.1992	0.1197	1.6639	0.0961
11.	P(1,1)	0.9894	0.0039	250.6137	0.0000
12.	P(1,2)	0.0275	0.0131	2.1002	0.0357

it's hard to get convergence, but it produces an even higher log likelihood than the Dueker filter. The explanation there is almost certainly what it was before—approximation error when the regimes aren't persistent.

Example 13.1 MS ARCH Model-Maximum Likelihood

From Hamilton & Susmel (1994), this does the estimation of the switching ARCH-L(3,2) with conditionally t disturbances. The details are in Section 13.1.1.

```

open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* This is the number of ARCH lags.
*
compute q=2
*
@MSSetup(lags=q,regimes=3)
*
* GV will be the relative variances in the regimes.
*
dec vect gv(nregimes)
*
* This will be the vector of ARCH parameters and the asymmetry
* parameter. The constant in the ARCH equation is fixed at 1 as the
* normalization.
*
dec vect a(q)
dec real xi
dec real nu
*
* We have three parts of the parameter set: the mean equation
* parameters (here, an intercept and an AR lag), the ARCH model
* parameters, which consists of the ARCH lags, the variance factors,
* and the asymmetry term and degrees of freedom, and the Markov
* switching parameters.
*
nonlin(parmset=archparms) a gv xi nu
nonlin(parmset=msparms) theta
*
* Run a linear regression to define the mean equation and provide guess
* values.
*
linreg(define=meaneq) vw
# constant vw{1}
compute gstart=%regstart()+q,gend=%regend()
compute b=%beta
*
nonlin(parmset=meanparms) b
*
* uu and u are used for the series of squared residuals and the
* series of residuals needed to compute the ARCH variances.
*
clear uu u
*
set uu = %seesq

```



```

set u    = %resids
frml uf = %eqnrvalue(meaneq,t,b)
*
*****
*
* ARCHRegimeTF returns a vector of likelihoods for the various
* expanded regimes at <<time>> given residual <<e>> using
* conditionally Student t errors.
*
function ARCHRegimeTF time e
type vector    ARCHRegimeTF
type real      e
type integer    time
*
local integer  i k
local real      h
*
dim ARCHRegimeTF(nexpand)
do i=1,nexpand
  compute h=1.0
  do k=1,q
    compute h=h+a(k)*uu(time-k)/gv(%MSLagRegime(i,k))
  end do k
  compute h=h+xi*$
  %if(u(time-1)<0.0,uu(time-1)/gv(%MSLagRegime(i,1)),0.0)
  compute ARCHRegimeTF(i)=%if(h>0,$
    exp(%logtdensity(h*gv(%MSLagRegime(i,0)),e,nu)),0.0)
end do i
end
*****
*
* For the non-switching parts of the ARCH parameters
*
compute a=%const(0.05),xi=0.0,nu=10.0
*
* As is typically the case with Markov switching models, there is no
* global identification of the regimes. By defining a fairly wide
* spread for gv, we'll hope that we'll stay in the zone of the
* likelihood where regime 1 is low variance and regime 2 is high.
*
compute gv=%seesq*||0.2,1.0,5.0||
*
* These are our guess values for the P matrix. We have to invert
* that to get the guess values for the logistic indexes.
*
compute p=||.80,.10,.05|.15,.80,.15||
compute theta=%msplogistic(p)
*
* We need to keep series of the residual (u) and squared residual
* (uu). Because the mean function is the same across regimes, we can
* just compute the residual and send it to ARCHRegimeTF, which
* computes the likelihoods for the different ARCH variances.
*
frml logl = u(t)=uf(t),uu(t)=u(t)^2,f=ARCHRegimeTF(t,u(t)), $

```

```

    fpt=%MSProb(t,f),log(fpt)
*
@MSFilterInit
maximize(parmset=meanparms+archparms+msparms,$
    start=(p=%mslogisticp(theta),pstar=%MSInit()),$
    method=bfgs, iters=400, pmethod=simplex, peters=5) logl gstart gend
*
@MSSmoothed gstart gend psmooth
set p1 = psmooth(t)(1)
set p2 = psmooth(t)(2)
set p3 = psmooth(t)(3)
graph(max=1.0, style=stacked, footer=$
    "Probabilities of Regimes in Three Regime Student-t model") 3
# p1
# p2
# p3
*
garch(equation=meaneq, distrib=t, asymmetric) gstart gend

```

Example 13.2 MS ARCH Model-MCMC

Estimation of a Markov switching ARCH model with two regimes, two lags, asymmetry and conditionally Gaussian errors. For technical information, see Section 13.1.2.

```

open data crspw.txt
calendar(w) 1962:7:3
data(format=prn,nolabels,org=columns) 1962:07:03 1987:12:29 vw
*
* These set the number of regimes, and the number of ARCH lags
*
compute q=2
@MSSSetup(regimes=2,lags=q)
*
* GV will be the relative variances in the regimes.
*
dec vect gv(nregimes)
*
* This will be the vector of ARCH parameters and the asymmetry
* parameter. The constant in the ARCH equation is fixed at 1 as the
* normalization.
*
dec vect a(q)
dec real xi
*
* uu and u are used for the series of squared residuals and the
* series of residuals needed to compute the ARCH variances.
*
clear uu u
*
* Set up the formula for the mean equation. For convenience, we'll

```

```

* use the VECTOR MSB to handle the parameters for this.
*
linreg(define=meaneq) vw
# constant vw{1}
compute gstart=%regstart()+q,gend=%regend()
compute msb=%beta
*
set uu = %seesq
set u = %resids
frml uf = %eqnrvalue(meaneq,t,msb)
*****
*
* ARCHRegimeGaussF returns a vector of likelihoods for the various
* expanded regimes at <<time>> given residual <<e>> using
* conditionally Gaussian errors.
*
function ARCHRegimeGaussF time e
type vector    ARCHRegimeGaussF
type real      e
type integer    time
*
local integer  i k
local real     hi
*
dim ARCHRegimeGaussF(nexpand)
do i=1,nexpand
  compute hi=1.0
  do k=1,q
    compute hi=hi+a(k)*uu(time-k)/gv(%MSLagRegime(i,k))
  end do k
  compute hi=hi+xi*$
    %if(u(time-1)<0.0,uu(time-1)/gv(%MSLagRegime(i,1)),0.0)
  compute ARCHRegimeGaussF(i)=%if(hi>0,$
    exp(%logdensity(hi*gv(%MSLagRegime(i,0)),e)),0.0)
end do i
end
*****
*
* This is similar to ARCHRegimeGaussF but evaluates the LOG
* likelihood and evaluates it only at the currently sampled values
* of MSRegime.
*
function MSARCHLogl time
type integer time
*
local integer k
local real    hi
*
compute u(time)=uf(time)
compute uu(time)=u(time)^2
compute hi=1.0
do k=1,q
  compute hi=hi+a(k)*uu(time-k)/gv(MSRegime(time-k))
end do k

```

```

compute hi=hi+xi*$
  %if(u(time-1)<0.0,uu(time-1)/gv(MSRegime(time-1)),0.0)
compute MSARCHLogl=%if(hi>0,$
  %logdensity(hi*gv(MSRegime(time)),u(time)),%na)
end
*****
*
* Do an ARCH model to get initial guess values and matrices for
* random walk M-H. Run this over the same range as we will use for
* the switching model.
*
garch(q=2,equation=meaneq,asymmetric) gstart gend
*
compute msb=%xsubvec(%beta,1,%nregmean)
compute far=%decomp(%xsubmat(%xx,1,%nregmean,1,%nregmean))
compute aaccept=0
*
* The GARCH instruction with 2 lags and asymmetry includes asymmetry
* terms for each lag. We'll ignore the second in extracting the guess
* values and the covariance matrix.
*
compute a=%xsubvec(%beta,%nregmean+2,%nregmean+1+q),$
  xi=%beta(%nregmean+2+q)
compute farch=%decomp(%xsubmat(%xx,%nregmean+2,%nregmean+2+q,$
  %nregmean+2,%nregmean+2+q))

compute [vector] msg=a~~xi
compute gaccept=0
*
* Start with guess values for GV that scale the original constant
* down and up.
*
compute gv(1)=%beta(%nregmean+1)*0.5
compute gv(2)=%beta(%nregmean+1)*2.0
dec vect nuv(nregimes) fiddle(nregimes)
ewise nuv(i)=50.0
compute vaccept=0
*
* Do the standard setup for MS filtering
*
@MSFilterSetup
*
* Prior for transitions.
*
dec vect[vect] gprior(nregimes)
dec vect tcounts(nregimes)
compute gprior(1)=||8.0,2.0||
compute gprior(2)=||2.0,8.0||
*
dec rect p(nregimes,nregimes)
ewise p(i,j)=gprior(j)(i)/%sum(gprior(j))
*
* Randomize the initial regimes
*
gset MSRegime gstart gend = %ranbranch(%fill(nregimes,1,1.0))

```

```

*
* This is a relatively small number of draws. If you have patience,
* you would want these to be more like 2000 and 5000.
*
compute nburn=200,ndraws=500
*
* For relabeling
*
compute tempvg=msg
compute tempvb=msb
compute tempp =p
*
* Used for bookkeeping
*
nonlin(parmset=allparms) gv msb msg p
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %parmspeek(allparms)
set regime1 = 0.0

infobox(action=define,lower=-nburn,upper=ndraws,progress) $
  "Gibbs Sampling"
do draw=-nburn,ndraws
  compute swaps=%index(gv)
  if %MSSwapCheck(swaps) {
    disp "Draw" draw "Executing swap"
    *
    * Relabel the scale factors
    *
    compute testvg=gv
    ewise gv(i)=tempvg(swaps(i))
    *
    * Relabel the transitions
    *
    compute tempp=p
    ewise p(i,j)=tempp(swaps(i),swaps(j))
  }
  compute a=%xsubvec(msg,1,q),xi=msg(q+1)
  *
  * Draw regimes by multi-move sampling
  *
  @MSFilterInit
  do time=gstart,gend
    compute u(time)=uf(time),uu(time)=u(time)^2
    compute ft=ARCHRegimeGaussF(time,u(time))
    @MSFilterStep time ft
  end do time
  @MSSample gstart gend MSRegime
  *
  * Draw p's
  *
  @MSDrawP(prior=gprior) gstart gend p
  *
  * Draw AR by Metropolis-Hastings
  *

```

```

sstats gstart gend msarchlogl(t)>>logplast
compute blast=msb
compute msb =blast+%ranmvnormal(far)
sstats gstart gend msarchlogl(t)>>logptest
compute alpha=exp(logptest-logplast)
if %ranflip(alpha) {
    compute logplast=logptest
    compute aaccept=aaccept+1
}
else
    compute msb=blast
*
* Draw the base GARCH parameters by M-H. Reject any with any
* negative coefficients on the ARCH lags.
*
compute glast=msg
:redrawg
compute msg=glast+%ranmvnormal(farch)
compute a=%xsubvec(msg,1,q),xi=msg(q+1)
if %minvalue(a)<0.0
    goto redrawg
sstats gstart gend msarchlogl(t)>>logptest
compute alpha=exp(logptest-logplast)
if %ranflip(alpha) {
    compute logplast=logptest
    compute gaccept=gaccept+1
}
else
    compute msg=glast
compute a=%xsubvec(msg,1,q),xi=msg(q+1)
*
* Draw the GARCH scale parameters. These can't be estimated like
* variances in regression models because they don't just scale
* the current variance, but also appear in lags in the ARCH
* formulation.
*
compute glast=gv
ewise fiddle(i)=%ranchisqr(nuv(i))/nuv(i)
compute logqratio=-2.0*%sum(%log(fiddle))
compute gv=glast.*fiddle
sstats gstart gend msarchlogl(t)>>logptest
compute alpha=exp(logptest-logplast-logqratio)
if %ranflip(alpha) {
    compute logplast=logptest
    compute vaccept=vaccept+1
}
else
    compute gv=glast
*
infobox(current=draw)
if draw>0 {
    set regime1 gstart gend = regime1+(MSRegime==1)
    compute bgibbs(draw)=%parmspeek(allparms)
}

```

```

end do draw
infobox(action=remove)
*
disp "Acceptance Percentages"
disp "Mean Parameters" ###.## 100.0*aaccept/ndraws
disp "ARCH Parameters" ###.## 100.0*gaccept/ndraws
disp "Variance Scales" ###.## 100.0*vaccept/ndraws
*
@mcmcpstproc(ndraws=ndraws,mean=bmeans,stderrs=bstderrs) bgibbs
*
report(action=define)
report(atrow=1,atcol=1,fillby=cols) %parmlabels(allparms)
report(atrow=1,atcol=2,fillby=cols) bmeans
report(atrow=1,atcol=3,fillby=cols) bstderrs
report(action=format,picture="*.#####")
report(action=show)
*
set regime1 = regime1/ndraws
graph(footer="Probability of low-variance regime")
# regime1

```

Example 13.3 MS GARCH Model-Approximate ML, Dueker filter

Example of estimation of a MS-GARCH model using the Dueker filter. This is discussed in Section 13.2.2.

```

open data weekly.xls
calendar(w) 1970:1:7
all 1994:04:13
data(format=xls,org=columns) 1970:01:07 1994:04:13 rate
diff rate / drate
spgraph(vfields=2,window="Figure 3")
  graph(header="One Month T-Bill Rates")
  # rate
  graph(header="One Month T-Bill Yields")
  # drate
spgraph(done)
*
* Estimate the mean equation by OLS
*
linreg(robust,define=meaneq) drate
# constant rate{1}
*
* Same by GARCH
*
garch(p=1,q=1,equation=meaneq,hseries=hgarch)
*
@DFSetup(equation=meaneq,regimes=2,mean=switching,garch=fullswitch)
@DFInit
compute gstart=%regstart(),gend=%regend()

```

```

*
@DFMeanParmset (parmset=meanparms)
@DFGARCHParmset (parmset=garchparms)
nonlin (parmset=fixpparms) p
compute p=|.90|.10|
*
frml LogDFFilter = DFLogL(t)
*
maximize (start=DFStart(),parmset=meanparms+garchparms+fixpparms,$
  pmethod=simplex,piters=5,method=bfgs) logDFFilter gstart gend
*
@MSSmoothed gstart gend psmooth
set p1 = psmooth(t) (1)
set p2 = psmooth(t) (2)
graph(max=1.0,footer="Probabilities of Regimes",$
  style=stacked) 2
# p1
# p2
*
* The anti-persistent mode
*
@DFInit
compute p=|.30|.70|
maximize (start=DFStart(),parmset=meanparms+garchparms+fixpparms,$
  pmethod=ga,piters=5,method=bfgs) logDFFilter gstart gend
@MSSmoothed gstart gend psmooth
set p1 = psmooth(t) (1)
set p2 = psmooth(t) (2)
graph(max=1.0,footer="Probabilities of Regimes, Anti-Persistence Mode",$
  style=stacked) 2
# p1
# p2

```

Example 13.4 MS GARCH Model-Approximate ML, Gray filter

Example of estimation of a MS-GARCH model using the Gray filter. Many of the details are shared with Example 13.3, and discussed in Section 13.2.2. Details specific to the Gray model are in Section 13.2.3.

```

open data weekly.xls
calendar(w) 1970:1:7
all 1994:04:13
data(format=xls,org=columns) 1970:01:07 1994:04:13 rate
diff rate / drate
spgraph(vfields=2,window="Figure 3")
  graph(header="One Month T-Bill Rates")
  # rate
  graph(header="One Month T-Bill Yields")
  # drate
spgraph(done)
*

```



```

* We can use @DFSetup with the added option of <<gray>> for most of
* the work.
*
linreg(robust,define=meaneq) drate
# constant rate{1}
*
* Same by GARCH
*
garch(p=1,q=1,equation=meaneq,hseries=hgarch)
*
@DFSetup(equation=meaneq,regimes=2,mean=switching,$
        garch=fullswitch,gray)
@DFInit
compute gstart=%regstart(),gend=%regend()
*
@DFMeanParmset(parmset=meanparms)
@DFGARCHParmset(parmset=garchparms)
nonlin(parmset=fixpparms) p
*****
*
* GrayRegimeFull returns the VECTOR of the H values across regimes
* for a full switching GARCH recursion
*
function GrayRegimeFull time
type vector    GrayRegimeFull
type integer    time
*
local integer i regime0
*
* When we get here, the HDF for the lag has already been collapsed
* so it's independent of the lagged regime.
*
dim GrayRegimeFull(nregimes)
do i=1,nregimes
    compute regime0=%MSLagRegime(i,0)
    compute GrayRegimeFull(i)=garchs(regime0)(1)+$
        garchs(regime0)(3)*MSGHF(1)(time-1)+$
        garchs(regime0)(2)*MSGUUF(1)(time-1)
    if MSGAsymmetric
        compute GrayRegimeFull(i)+=garchs(regime0)(4)*MSGUUSF(1)(time-1)
end do i
end
*
compute msgarchfnc=GrayRegimeFull
*
*****
*
* GrayLogL does a single step of the Gray filter
*
function GrayLogL time
type real      GrayLogL
type integer    time
*
local vector    hvec pprep

```

```

local vector    fvec(nregimes) evec(nregimes)
local real      e mu mux
local integer   i
*
* Compute regime-dependent variances
*
compute hvec=msgarchfnc(time)
*
* Compute the residuals for each current regime
*
compute evec=GARCHRegimeResids(time)
*
* Compute the likelihood of the regimes
*
do i=1,nregimes
  compute e=evec(i)
  compute fvec(i)=msgarchll(msgarchvarf(hvec,i),e,0.0)
end do i
*
* Do a filter step to update the probabilities. The log
* likelihood element is returned in the LOGF option.
*
@MSFilterStep(logf=GrayLogL) time fvec
*
* Fetch the predicted probability
*
compute ppred=pt_t1(time)
*
* Compute the values of uu (squared residual) and h (variance) to
* be used for the period following.
*
compute MSGHF(1)(time)=0.0
compute mux=0.0
do i=1,nregimes
  if MSGMeanSwitch
    compute mu=%eqnvalue(meaneq,time,MSGBetas(i))
  else
    compute mu=%eqnvalue(meaneq,time,MSGBeta)
  compute MSGHF(1)(time)+=ppred(i)*(mu^2+hvec(i))
  compute mux+=ppred(i)*mu
end do i
*
compute e=([series] %eqndepvar(meaneq))(time)-mux
compute MSGUUF(1)(time)=e^2
compute MSGHF(1)(time)-=mux^2
*
* It's not clear what this should be
*
compute MSGUUSF(1)(time)=e*%minus(e)
end
*****
*
frml LogGrayFilter = GrayLogL(t)
compute msgarchfnc=GrayRegimeFull

```

```
*
compute p=||.90,.10||
maximize(start=DFStart(),parmset=meanparms+garchparms+fixpparms,$
  pmethod=simplex,piters=10,method=bfgs,itors=100) $
  logGrayFilter gstart gend
*
@DFInit
compute p=||.30,.70||
compute msgarchfnc=GrayRegimeFull
maximize(start=DFStart(),parmset=meanparms+garchparms+fixpparms,$
  pmethod=ga,piters=5,method=bfgs,itors=100,trace) $
  logGrayFilter gstart gend
```

A General Result on Smoothing

Proposition. Let y and x be random variables defined on a common probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Let \mathcal{I} and \mathcal{J} be sub-sigma fields (information sets) of \mathcal{F} with $\mathcal{I} \subseteq \mathcal{J}$. If

$$f(x|y, \mathcal{I}) = f(x|y, \mathcal{J}) \quad (\text{A.1})$$

then

$$\frac{f(x|\mathcal{J})}{f(x|\mathcal{I})} = \int f(y|x, \mathcal{I}) \frac{f(y|\mathcal{J})}{f(y|\mathcal{I})} dy \quad (\text{A.2})$$

Proof

By Bayes' rule

$$f(x|y, \mathcal{I}) = \frac{f(y|x, \mathcal{I})f(x|\mathcal{I})}{f(y|\mathcal{I})}$$

By (A.1), $f(x|y, \mathcal{I}) = f(x|y, \mathcal{J})$, so we have

$$f(x|y, \mathcal{J}) = \frac{f(y|x, \mathcal{I})f(x|\mathcal{I})}{f(y|\mathcal{I})} \quad (\text{A.3})$$

By standard results,

$$f(x|\mathcal{J}) = \int f(x, y|\mathcal{J}) dy = \int f(x|y, \mathcal{J})f(y|\mathcal{J}) dy \quad (\text{A.4})$$

Substituting (A.3) into (A.4) yields

$$f(x|\mathcal{J}) = \int \frac{f(y|x, \mathcal{I})f(x|\mathcal{I})}{f(y|\mathcal{I})} f(y|\mathcal{J}) dy$$

which can be rearranged to yield (A.2).

The EM Algorithm

The EM algorithm (Dempster et al. (1977)) is a general method for handling situations where you have a combination of observed data and unobserved or latent data, and where the likelihood would have a convenient form *if* the unobserved data were known. It's an iterative algorithm, which repeats the pair of an E (or Expectations) step and an M (or Maximization) step.

Let y represent the full record of the observed data and x the full record of the unobserved data. Let Θ represent the model parameters being estimated. We're assuming that the log likelihood as a function of both x and y is $\log f(y, x|\Theta)$ which is itself fairly easy to handle. Unfortunately, since we only see y , maximum likelihood needs to maximize over Θ the log marginal density:

$$\log f(y|\Theta) = \log \int f(y, x|\Theta) dx \quad (\text{B.1})$$

which can be quite difficult in the (non-trivial) case where x and y aren't independent. The result in the DLR paper is that repeating the following process will (under regularity conditions) eventually produce the maximizer of (B.1):

E Step: Let Θ_0 be the parameter values at the end of the previous iteration. Determine the functional form for

$$Q(\Theta|\Theta_0) = E_{x|y, \Theta_0} \log f(y, x|\Theta) \quad (\text{B.2})$$

M Step: Maximize (B.2) with respect to Θ .

The E step is, in many cases, simpler than the calculation in (B.1) because it integrates the log density, which often is a much simpler functional form than the density itself. Where the augmented data set (y, x) is jointly Normal, a small simplifying assumption in the E step allows you to use the even simpler

$$\tilde{Q}(\Theta|\Theta_0) = \log f(y, E(x|y, \Theta_0)|\Theta) \quad (\text{B.3})$$

that is, you just do a standard log likelihood maximization treating x as observed at its expected value given the observable y and the previous parameter values.

While EM can be very useful, it does have some drawbacks. The most important is that it tends to be very slow reaching final convergence. While (B.2) can often be maximized with a single matrix calculation (if, for instance, it simplifies

to least squares on a particular set of data), that is only one iteration in the process. The next time through, the function changes since x has changed. It also (for much the same reason) gives you point estimates but not standard errors.

The best situation for EM is where the model is basically a large, linear model controlled (in a non-linear way) by the unobserved x . Even if (B.1) is tractable, estimating it by variational methods like BFGS can be very slow. Because EM takes care of the bulk of the parameters by some type of least squares calculation, the iterations are much faster, so taking more iterations isn't as costly. And in such situations, EM tends to make much quicker progress towards the optimum, again, because it handles most of the parameters constructively. Where possible, the ideal is to use EM to get close to the optimum, then switch to full maximum likelihood, which has better convergence properties, and gives estimates for the covariance matrix.

Except where (B.3) can be applied, an efficient implementation of EM usually requires special purpose calculations to do the M step, which need to be enclosed in a DO loop over the EM iteration process. RATS offers an alternative (the `ESTEP` option on **MAXIMIZE**) which is much simpler to put into place, though nowhere near as efficient computationally.¹ Again, it might be a good idea to start with this and switch to an optimized calculation if it turns out that the `ESTEP` approach is too slow. The `ESTEP` option executes code (generally calling a **FUNCTION**) at the start of each iteration to do any calculations required for the E Step. The formula for **MAXIMIZE** is then the one in (B.2). Note that this is different from the `START` option, which provides code which is executed at the start of each function evaluation. If you use **MAXIMIZE** with `ESTEP`, use `METHOD=BHHH`, not (the default) `METHOD=BFGS`. Because the function being optimized changes from iteration to iteration (as a result of the E Step), the BFGS update steps aren't valid.

Generalized EM

It turns out that it isn't necessary to fully maximize (B.2) during the M step to get the overall result that EM will eventually reach a mode of the overall optimization problem. In *Generalized EM*, the M step takes one (or more) iterations on a process which increases the value of (B.2). This is particularly helpful in multivariate estimation, since actually maximizing (B.2) would then almost always involve an iterated SUR estimate. With iterated SUR, most of the improvement in the function value comes on the first iteration—since the function itself will change with the next E step, extra calculations to achieve convergence to a temporary maximum aren't especially useful.

¹This is less efficient computationally because it estimates the entire parameter vector using a hill-climbing method, rather than directly optimizing each subset.

Hierarchical Priors

If $x_t \sim N(0, \sigma^2)$ *i.i.d.*, then the likelihood for T data points is

$$(\sigma^2)^{-T/2} \exp \left(-\frac{1}{2\sigma^2} \sum_{t=1}^T x_t^2 \right)$$

The natural conjugate prior for σ^2 takes the form:

$$(\sigma^2)^{-\nu/2-1} \exp \left(-\frac{\nu}{2\sigma^2} s^2 \right)$$

where ν (degrees of freedom) and s^2 (scale) are the two parameters governing this. This is a (scaled) inverse chi-squared density for σ^2 , as the precision $h \equiv \sigma^{-2}$ has a chi-squared distribution. Combining the likelihood and the prior gives the posterior:

$$(\sigma^2)^{-(T+\nu)/2-1} \exp \left(-\frac{1}{2\sigma^2} \left(\nu s^2 + \sum_{t=1}^T x_t^2 \right) \right)$$

which is an inverse chi-squared with $T + \nu$ degrees of freedom and scale

$$\frac{\left(\nu s^2 + \sum_{t=1}^T x_t^2 \right)}{(T + \nu)} \tag{C.1}$$

The scale parameter is roughly equal to the mean¹ of the distribution for σ^2 , and is a weighted average of the prior scale (s^2) and the maximum likelihood estimate

$$\hat{\sigma}^2 = \frac{\sum x_t^2}{T}$$

To draw from this distribution with RATS, you compute the numerator of (C.1) and the degrees of freedom and use `numerator/%ranchisqr(degrees)`

The most common choice for the prior is the non-informative $\nu = 0$ (which makes s^2 irrelevant). An obvious reason is that an informative prior requires s^2 , for which there rarely is an obvious value. This usually works out fine since, unless T is very small, the data have good information on σ^2 . However, in models with switching variances, this is much less likely to work well. The above would be applied to each subsample, some of which might end up being

¹Technically, it's the reciprocal of the mean of the reciprocal.

fairly small (at least in some Gibbs sweeps). When T is a number under 10, there is a non-trivial chance that the sampled value will be quite different (by a factor of two or more) from $\hat{\sigma}^2$.

An alternative which can be used in a situation like this is a *hierarchical* prior. The variances in the subsample are written as the product of a common scale factor (defined across the full sample), and a relative scale. We'll write this as $r_i^2 \sigma^2$ for subsample i . The common scale factor is drawn using the full sample of data, and so can be done with a non-informative prior, while the relative scale can use an informative prior with a scale of 1.0; that is, all the variances are thought to be scales of an overall variance. With a sample split into two regimes based upon variance, the likelihood is:

$$(r_1^2 \sigma^2)^{-T_1/2} \exp \left(-\frac{1}{2r_1^2 \sigma^2} \sum_{S=1} x_t^2 \right) (r_2^2 \sigma^2)^{-T_2/2} \exp \left(-\frac{1}{2r_2^2 \sigma^2} \sum_{S=2} x_t^2 \right)$$

and the hierarchical prior (assuming a non-informative one for the common variance) is

$$\sigma^{-2} (r_1^2)^{-\nu_1/2-1} \exp \left(-\frac{\nu_1}{2r_1^2} \right) (r_2^2)^{-\nu_2/2-1} \exp \left(-\frac{\nu_2}{2r_2^2} \right)$$

From the product of the prior and likelihood, we can isolate the r_1^2 factors as

$$(r_1^2)^{-(T_1+\nu_1)/2-1} \exp \left(-\frac{1}{2r_1^2} \left(\nu_1 + \frac{1}{\sigma^2} \sum_{S=1} x_t^2 \right) \right) \quad (\text{C.2})$$

The “mean” of σ_1^2 generated from this is

$$\frac{\nu_1 \sigma^2 + \sum_{S=1} x_t^2}{\nu_1 + T_1}$$

With even a fairly small ν_1 (say 3 or 4), this will “shrink” the variance estimate modestly towards the overall variance scale, and help prevent problems with small subsamples.

If we now isolate σ^2 , we get

$$(\sigma^2)^{-(T_1+T_2)/2-1} \exp \left(-\frac{1}{2\sigma^2} \left[\frac{1}{r_1^2} \sum_{S=1} x_t^2 + \frac{1}{r_2^2} \sum_{S=2} x_t^2 \right] \right) \quad (\text{C.3})$$

The process of drawing the variances from a hierarchical prior is to do the following (in either order):

1. Draw the overall scale using the sums of squares weighted by each regime's relative variance (the bracketed term in the exponent of (C.3)), with degrees of freedom equal to the total sample size.

2. Draw the relative variances using the sum of squares in each subsample weighted by the overall variance plus the prior degrees of freedom (the term in the exponent of (C.2)) with degrees of freedom equal to the subsample size plus the degrees of freedom.

Hierarchical Priors for Multivariate Regressions

The same type of problem can occur in a multivariate regression setting with switching covariance matrices, except it's even more pronounced. Without help from a prior, you need at least n data points in a regime just to get a full-rank estimate of an $n \times n$ covariance matrix and the behavior of covariance matrices estimated with barely a sufficient amount of data can be quite bad.

The same basic idea from above applies to drawing covariance matrices, which now will follow an inverse Wishart (Appendix F.9). If x_t is the $n \times 1$ vector of residuals at time t , Σ is the common covariance matrix and ν is the shrinkage degrees of freedom then the mean for the Wishart is

$$\sum_{t=1}^T x_t x_t' + \nu \Sigma$$

with $T + \nu$ degrees of freedom. Drawing the covariance matrix from this is done with

```
compute sigmav(k)=%ranwisharti(%decomp(inv(uu(k)+nuprior(k)*sigma)), $
    tcounts(k)+nuprior(k))
```

where `uu(k)` is the sum of the outer product of residuals for regime k , `tcounts(k)` is number of observations in that regime, and `nuprior(k)` is the prior degrees of freedom (which will almost always be the same across regimes).

The draw for the common Σ isn't quite as simple since we're now dealing with matrices rather than scalars, and, matrices, in general, don't allow wholesale rearrangements of calculations. The simplest way to handle this is to draw the common sigma conditional on the regime-specific ones by inverting the prior from an inverse Wishart for the regime-specific covariance matrix to a Wishart for the common one. The mean for this is the inverse of the sums of the inverses. The draw can be made with:

```
compute uucommon=%zeros(nvar,nvar)
do k=1,nstates
    compute uucommon=uucommon+nuprior(k)*inv(sigmav(k))
end do k
compute sigma=%ranwishartf(%decomp(inv(uucommon)), %sum(nuprior))
```

Gibbs Sampling and Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) techniques allow for generation of draws from distributions which are too complicated for direct analysis. The simplest form of this is Gibbs sampling. This simulates draws from the density by means of a correlated Markov Chain. Under the proper circumstances, estimates of sample statistics generated from this converge to their true means under the actual posterior density.

Gibbs Sampling

The idea behind Gibbs sampling is that we partition our set of parameters into two or more groups: for illustration, we'll assume we have two, called Θ_1 and Θ_2 . We want to generate draws from a joint density $f(\Theta_1, \Theta_2)$, but don't have any simple way to do that. We can always write the joint density as $f(\Theta_1|\Theta_2)f(\Theta_2)$ and as $f(\Theta_2|\Theta_1)f(\Theta_1)$. In a very wide range of cases, these conditional densities *are* tractable. It's the unconditional densities of the *other* block that are the problem.

The intuition behind Gibbs sampling is that if we draw Θ_1 given Θ_2 , and then Θ_2 given Θ_1 , the pair should be closer to their unconditional distribution than before. Each combination of draws is known as a *sweep*. Repeat sweeps often enough and it should converge to give draws from the joint distribution. This turns out to be true in most situations. Just discard enough at the beginning (called the *burn-in* draws) so that the chain has had time to converge to the unconditional distribution. Once you *have* a draw from $f(\Theta_1, \Theta_2)$, you (of necessity) have a draw from the marginals $f(\Theta_1)$ and $f(\Theta_2)$, so now each sweep will give you a new draw from the desired distribution. They're just not *independent* draws. With enough "forgetfulness", however, the sample averages of the draws will converge to the true mean of the joint distribution.

Gibbs sampling works best when parameters which are (strongly) correlated with each other are in the same block, otherwise it will be hard to move both since the sampling procedure for each is tied to the other.

Metropolis-Hastings/Metropolis within Gibbs

Metropolis within Gibbs is a more advanced form of MCMC. Gibbs sampling requires that we be able to generate the draws from the conditional densities. However, there are only a handful of distributions for which we can do that—things like Normals, gammas, Dirichlets. In many cases, the desired density

is the likelihood for a complicated non-linear model which doesn't have such a form.

Suppose, we want to sample the random variable θ from $f(\theta)$ which takes a form for which direct sampling is difficult.¹ Instead, we sample from a more convenient density $q(\theta)$. Let $\theta^{(i-1)}$ be the value from the previous sweep. Compute

$$\alpha = \frac{f(\theta)}{f(\theta^{(i-1)})} \times \frac{q(\theta^{(i-1)})}{q(\theta)} \quad (\text{D.1})$$

With probability α , we accept the new draw and make $\theta^{(i)} = \theta$, otherwise we stay with our previous value and make $\theta^{(i)} = \theta^{(i-1)}$. Note that it's possible to have $\alpha > 1$, in which case we just accept the new draw.

The first ratio in (D.1) makes perfect sense. We want, as much as possible, to have draws where the posterior density is high, and not where it's low. The second counterweights (notice that it's the ratio in the opposite order) for the probability of drawing a given value. Another way of looking at the ratio is

$$\alpha = \frac{f(\theta)}{q(\theta)} \bigg/ \frac{f(\theta^{(i-1)})}{q(\theta^{(i-1)})}$$

f/q is a measure of the relative desirability of a draw. The ones that really give us a strong "move" signal are where the target density (f) is high and the proposal density (q) is low—we may not see those again, so when we get a chance we should move. Conversely, if f is low and q is high, we might as well stay put—we may revisit that one at a later time.

What this describes is *Independence Chain Metropolis*, where the proposal density doesn't depend upon the last draw. Where a set of parameters is expected to have a single mode, a good proposal density often can be constructed from maximum likelihood estimates, using a Normal or multivariate t centered at the ML estimates, with the covariance matrix some scale multiple (often just 1.0) of the estimate coming out of the maximization procedure.

If the actual density has a shape which *isn't* a good match for a Normal or t , this is unlikely to work well. If there are points where f is high and q is relatively low, it might take a very large number of draws before we find them, and once we get there we'll likely stay for a while. A more general procedure has the proposal density depending upon the last value: $q(\theta|\theta^{(i-1)})$. The acceptance criterion is now based upon:

$$\alpha = \frac{f(\theta)}{q(\theta|\theta^{(i-1)})} \bigg/ \frac{f(\theta^{(i-1)})}{q(\theta^{(i-1)}|\theta)} \quad (\text{D.2})$$

Note that the counterweighting is now based upon the ratio of the probability of moving from $\theta^{(i-1)}$ to θ to the probability of moving back. The calculation

¹What we're describing here is Metropolis-Hastings or M-H for short. In all our applications, the densities will be conditional on the other blocks of parameters, which is formally known as Metropolis within Gibbs.

simplifies greatly if the proposal is a mean zero Normal or t added to $\theta^{(i-1)}$. Because of symmetry, $q(\theta|\theta^{(i-1)}) = q(\theta^{(i-1)}|\theta)$, so the q cancels, leaving just:

$$\alpha = f(\theta) / f(\theta^{(i-1)})$$

This is known as *Random Walk Metropolis*, which is probably the most common choice for Metropolis within Gibbs. Unlike Independence Chain, when you move to an isolated area where f is high, you can always move back by, in effect, retracing your route.

Avoiding Overflows

The f that we've been using in this is either the sample likelihood or the posterior (sample likelihood times prior). With many data sets, the sample *log* likelihood is on the order of 100's or 1000's (positive or negative). If you try to convert these large log likelihoods by taking the exp, you will likely overflow (for large positive) or underflow (for large negative), in either case, losing the actual value. Instead, you need to calculate the ratios by adding and subtracting in log form, and taking the exp only at the end.

Diagnostics

There are two types of diagnostics: those on the behavior of the sampling methods for subsets of the parameters, and those on the behavior of the overall chain. When you use Independence Chain Metropolis, you would generally like the acceptance rate to be fairly high. In fact, if $q = f$ (which means that you're actually doing a simple Gibbs draw), everything cancels and the acceptance is $\alpha = 1$. Numbers in the range of 20-30% are generally fine, but acceptance rates well below 10% are often an indication that you have a bad choice for q —your proposal density isn't matching well with f . When you use Random Walk Metropolis, an acceptance probability near 100% *isn't* good. You will almost never get rates like that unless you're taking very small steps and thus not moving around the parameter space sufficiently. Numbers in the 20-40% range are usually considered to be desirable. You can often tweak either the variance or the degrees of freedom in the increment to move that up or down. Clearly, in either case, you need to count the number of times you move and compare with the total number of draws.

If you're concerned about the overall behavior of the chain, you can run it several times and see if you get similar results. If you get decidedly different results, it's possible that the chain hasn't run long enough to converge, requiring a greater number of burn-in draws. The `@MCMCPOSTPROC` procedure also computes a CD measure which does a statistical comparison of the first part of the accepted draws with the last part. If the burn-in is sufficient, these should be asymptotically standard Normal statistics (there's one per parameter). If you get values that have absolute values far out in the tails for a Normal (such as 4 and above), that's a strong indication that you either have a general problem with the chain, or you just haven't done a long enough burn-in.

Pathologies

Most properly designed chains will *eventually* converge, although it might take many sweeps to accomplish this. There are, however, some situations in which it won't work no matter how long it runs. If the Markov Chain has an *absorbing state* (or set of states), once the chain moves into this, it can't get out. This is particularly common with switching models, where once the probability of a regime is low enough you get no data points which actually are considered to fall into it, therefore the probability is pushed even more towards zero.

Time-Varying Transition Probabilities

As an alternative to models with fixed transition probabilities, Filardo (1994) and Diebold et al. (1994) proposed allowing them to change from period to period based upon the values of exogenous variables.

If there are M regimes, we need to generate a transition matrix with $M(M - 1)$ free elements at each time period. The typical way to do this is to have that many linear “index” functions $Z_t \theta_{ij}$ which are combined non-linearly to form probabilities.¹ The most obvious way to do this is with a logistic, so

$$P(S_t = i | S_{t-1} = j) = \frac{\exp(Z_t \theta_{ij})}{\sum_k \exp(Z_t \theta_{kj})} \quad (\text{E.1})$$

where θ_{Mj} is normalized as the zero vector. For $M = 2$, it’s possible to use a standard Normal, rather than logistic, but the logistic generalizes easily to larger M and is simpler to use.

E.1 EM Algorithm

The only adjustment required to the M step is for the estimation of the parameters of the index functions. Unlike the case with fixed transition probabilities, it isn’t possible to maximize these with a single calculation. Instead, it makes more sense to do generalized EM and just make a single iteration of the maximization algorithm. We need one step in the maximizer for:

$$\sum_{t=1}^T (\log f(S_t | S_{t-1}, \Theta)) \hat{p}_{ij,t} \quad (\text{E.2})$$

where the $\hat{p}_{ij,t}$ are the smoothed probabilities of $S_t = i$ and $S_{t-1} = j$ at t . Because it appears in the denominator of (E.1), each θ_{ij} for a fixed j appears in the $\log f$ for all the other i . The derivative of (E.2) with respect to θ_{ij} is

$$\sum_{t=1}^T \sum_{k=1}^M \hat{p}_{kj,t} (I\{i = k\} - p_{ij,t}(\theta)) Z_t = \sum_{t=1}^T (\hat{p}_{ij,t}(1 - p_{ij,t}(\theta)) - (\hat{p}_{\bullet,j,t} - \hat{p}_{ij,t}) p_{ij,t}(\theta)) Z_t \quad (\text{E.3})$$

¹This assumes that the same explanatory variables are used in all the transitions. That can be relaxed with a bit of extra work.

where the inner sum collapses because the terms with $i \neq k$ are identical other than the $\hat{p}_{kj,t}$ which just add up to the $\hat{p}_{\bullet,j,t} - \hat{p}_{ij,t}$.² Zeroing this equalizes the predicted and actual (average) log odds ratios between moving to i or not moving to i from regime j .

The second derivative is also easily computed as:

$$- \sum_{t=1}^T \hat{p}_{\bullet,j,t} (1 - p_{ij,t}(\theta)) p_{ij,t}(\theta) Z_t' Z_t \quad (\text{E.4})$$

The one-step adjustment can be written as a weighted least squares regression of

$$\frac{(\hat{p}_{ij,t}(1 - p_{ij,t}(\theta)) - (\hat{p}_{\bullet,j,t} - \hat{p}_{ij,t})p_{ij,t}(\theta))}{\hat{p}_{\bullet,j,t}(1 - p_{ij,t}(\theta))p_{ij,t}(\theta)} \quad (\text{E.5})$$

on Z_t with weights $\hat{p}_{\bullet,j,t}(1 - p_{ij,t}(\theta))p_{ij,t}(\theta)$.

² $\hat{p}_{\bullet,j,t}$ is the empirical probability of having $S_{t-1} = j$. The \hat{p} sum to 1 across all i, j , so this downweights any observations that have very low $S_{t-1} = j$ and thus tell us relatively little about j to i transitions.

Probability Distributions

F.1 Univariate Normal

Parameters	Mean (μ), Variance (σ^2)
Kernel	$\sigma^{-1} \exp \left(-\frac{(x - \mu)^2}{2\sigma^2} \right)$
Support	$(-\infty, \infty)$
Mean	μ
Variance	σ^2
Main uses	Distribution of error terms in univariate processes. Asymptotic distributions. Prior, exact and approximate posteriors for parameters with unlimited ranges.
Density Function	<code>%DENSITY(x)</code> is the non-logged standard Normal density. More generally, <code>%LOGDENSITY(variance,u)</code> . Use <code>%LOGDENSITY(sigmasq,x-mu)</code> to compute $\log f(x \mu, \sigma^2)$.
CDF	<code>%CDF(x)</code> is the standard Normal CDF (running from 0 in the left tail to 1 in the right). To get $F(x \mu, \sigma^2)$, use <code>%CDF((x-mu)/sigma)</code> . <code>%ZTEST(z)</code> gives the two-tailed tail probability (probability a $N(0,1)$ exceeds z in absolute value).
Inverse CDF	<code>%INVNORMAL(p)</code> is the standard Normal inverse CDF.
Random Draws	<code>%RAN(s)</code> draws one or more (depending upon the target) independent $N(0, s^2)$. <code>%RANMAT(m,n)</code> draws a matrix of independent $N(0, 1)$.

F.2 Beta distribution

Parameters	2 (called α and β below)
Kernel	$x^{\alpha-1} (1-x)^{\beta-1}$
Support	$[0, 1]$. If $\alpha > 1$, the density is 0 at $x = 0$; if $\beta > 1$, the density is 0 at $x = 1$.
Mean	$\frac{\alpha}{\alpha + \beta}$
Variance	$\frac{\alpha\beta}{(\alpha + \beta)^2 (\alpha + \beta + 1)}$
Main uses	priors and approximate posteriors for parameters that measure fractions, or probabilities, or autoregressive coefficients (when a negative value is unreasonable).
Density function	<code>%LOGBETADENSITY (x, a, b)</code>
Random Draws	<code>%RANBETA (a, b)</code> draws one or more (depending on the target) independent Betas.
Moment Matching	<code>%BetaParms(mean,sd)</code> (external function) returns the 2-vector of parameters for a beta with the given mean and standard deviation.
Extensions	If $x \sim \text{Beta}(\alpha, \beta)$ then $ax + b$ is distributed on $[a, a + b]$

F.3 Dirichlet distribution

Parameters	n , called α_i for $i = 1, \dots, n$ below, with $\Sigma_\alpha \equiv \alpha_1 + \dots + \alpha_n$.
Kernel	$\prod_i x_i^{\alpha_i-1}$. If the α are all 1, this is uniform on its support.
Support	$x_i \geq 0, \sum_i x_i = 1$
Mean	for component i , α_i / Σ_α
Variance	for component i , $\frac{\alpha_i(\Sigma_\alpha - \alpha_i)}{\Sigma_\alpha^2(\Sigma_\alpha + 1)}$. The larger the α , the smaller the variance.
Main uses	priors and posteriors for parameters that measure probabilities with more than two alternatives.
Density Function	<code>%LOGDIRICHLET(x,alpha)</code> is the log density at the VECTOR <code>x</code> (whose elements have to sum to 1) with parameter VECTOR <code>alpha</code> .
Random Draws	<code>%RANDIRICHLET(alpha)</code> draws a vector of Dirichlet probabilities with the <code>alpha</code> as the VECTOR of shape parameters.

F.4 (Scaled) Inverse Chi-Squared Distribution

Parameters	Degrees of freedom (ν) and scale (τ^2). An inverse chi-squared is the reciprocal of a chi-squared combined with scaling factor which represents a “target” variance that the distribution is intended to represent. (Note that the mean is roughly τ^2 for large degrees of freedom.)
Kernel	$x^{-(a+1)} \exp(-bx^{-1})$
Integrating Constant	$b^a / \Gamma(a)$
Support	$[0, \infty)$
Mean	$\frac{\nu\tau^2}{\nu-2}$ if $\nu > 2$
Variance	$\frac{2\nu^2\tau^4}{(\nu-2)^2(\nu-4)}$ if $\nu > 4$
Main uses	Prior, exact and approximate posterior for the variance of residuals or other shocks in a model. The closely-related inverse gamma (Appendix F.6) can be used for that as well, but the scaled inverse chi-squared tends to be more intuitive.
DensityFunction	<code>%LOGINVCHISQRDENSITY(x, nu, tausq)</code> . Added with RATS 9.1.
Moment Matching	<code>%InvChisqrParms(mean, sd)</code> (external function) returns the 2-vector of parameters ((ν, τ^2) in that order) for the parameters of an inverse chi-squared with the given mean and standard deviation. If <code>sd</code> is the missing value, this will return $\nu = 4$, which is the largest value of ν which gives an infinite variance.
Random draws	You can use <code>(nu*tausq)/%ranchisqr(nu)</code> . Note that you divide by the random chi-squared.

F.5 Gamma Distribution

Parameters	shape (a) and scale (b), alternatively, degrees of freedom (ν) and mean (μ). The RATS functions use the first of these. The relationship between them is $a = \nu/2$ and $b = \frac{2\mu}{\nu}$. The chi-squared distribution with ν degrees of freedom is a special case with $\mu = \nu$.
Kernel	$x^{a-1} \exp\left(-\frac{x}{b}\right)$ or $x^{(\nu/2)-1} \exp\left(-\frac{x\nu}{2\mu}\right)$
Support	$[0, \infty)$
Mean	ba or μ
Variance	b^2a or $\frac{2\mu^2}{\nu}$
Main uses	Prior, exact and approximate posterior for the precision (reciprocal of variance) of residuals or other shocks in a model
Density function	<code>%LOGGAMMADENSITY(x, a, b)</code> . For the $\{\nu, \mu\}$ parameterization, use <code>%LOGGAMMADENSITY(x, .5*nu, 2.0*mu/nu)</code>
Random Draws	<code>%RANGAMMA(a)</code> draws one or more (depending upon the target) independent Gammas with unit scale factor. Use <code>b*%RANGAMMA(a)</code> to get a draw from $Gamma(a, b)$. If you are using the $\{\nu, \mu\}$ parameterization, use <code>2.0*mu*%RANGAMMA(.5*nu)/nu</code> . You can also use <code>mu*%RANCHISQR(nu)/nu</code> .
Moment Matching	<code>%GammaParms(mean, sd)</code> (external function) returns the 2-vector of parameters ((a, b) parameterization) for a gamma with the given mean and standard deviation.

F.6 Inverse Gamma Distribution

Parameters	shape (a) and scale (b). An inverse gamma is the reciprocal of a gamma. The special case is the scaled inverse chi-squared (Appendix F.4 with parameters ν (degrees of freedom) and τ^2 (scale parameter) which has $a = \nu/2$ and $b = \nu\tau^2/2$.
Kernel	$x^{-(a+1)} \exp(-bx^{-1})$
Integrating Constant	$b^a/\Gamma(a)$
Support	$[0, \infty)$
Mean	$\frac{b}{(a-1)}$ if $a > 1$
Variance	$\frac{b^2}{(a-1)^2(a-2)}$ if $a > 2$
Main uses	Prior, exact and approximate posterior for the variance of residuals or other shocks in a model. For these purposes, it's usually simpler to directly use the scaled inverse chi-squared variation.
Density Function	<code>%LOGINVGAMMADENSITY(x, a, b)</code> . Added with RATS 9.1.
Moment Matching	<code>%InvGammaParms(mean, sd)</code> (external function) returns the 2-vector of parameters ((a, b) parameterization) for the parameters of an inverse gamma with the given mean and standard deviation. If <code>sd</code> is the missing value, this will return $a = 2$, which is the largest value of a which gives an infinite variance.
Random draws	You can use <code>b/%rangamma(a)</code> . A draw from a scaled inverse chi-squared is typically done with <code>nu*tausqr/%ranchisqr(nu)</code> .

F.7 Bernoulli Distribution

Parameters	probability of success (p)
Kernel	$p^x(1 - p)^{1-x}$
Range	0 or 1
Mean	p
Variance	$p(1 - p)$
Main uses	Realizations of 0-1 valued variables (usually latent states).
Random Draws	<p><code>%RANBRANCH(p1, p2)</code> draws one or more independent trials with (integer) values 1 or 2. The probability of 1 is $\frac{p1}{p1 + p2}$ and the probability of 2 is $\frac{p2}{p1 + p2}$. (Thus, you only need to compute the relative probabilities of the two states). The 1-2 coding for this is due to the use of 1-based subscripts in RATS .</p>

F.8 Multivariate Normal

Parameters	Mean (μ), Covariance matrix (Σ) or precision (H)
Kernel	$ \Sigma ^{-1/2} \exp \left(-\frac{1}{2} (x - \mu)' \Sigma^{-1} (x - \mu) \right)$ or $ H ^{1/2} \exp \left(-\frac{1}{2} (x - \mu)' H (x - \mu) \right)$
Support	\mathbb{R}^n
Mean	μ
Variance	Σ or H^{-1}
Main uses	Distribution of multivariate error processes. Asymptotic distributions. Prior, exact and approximate posteriors for a collection of parameters with unlimited ranges.
Density Function	<code>%LOGDENSITY(sigma,u)</code> . To compute $\log f(x \mu, \Sigma)$ use <code>%LOGDENSITY(sigma,x-mu)</code> . (The same function works for univariate and multivariate Normals).
Distribution Function	<code>%BICDF(x,y,rho)</code> returns $P(X \leq x, Y \leq y)$ for a bivariate Normal with mean zero, variance 1 in each component and correlation rho.
Random Draws	<code>%RANMAT(m,n)</code> draws a matrix of independent $N(0,1)$. <code>%RANMVNORMAL(F)</code> draws an n -vector from a $N(0, FF')$, where F is any factor of the covariance matrix. This setup is used (rather than taking the covariance matrix itself as the input) so you can do the factor just once if it's fixed across a set of draws. To get a single draw from a $N(\mu, \Sigma)$, use <code>MU+%RANMVNORMAL(%DECOMP(SIGMA))</code> <code>%RANMVPOST</code> , <code>%RANMVPOSTCMOM</code> , <code>%RANMVKRON</code> and <code>%RANMVKRONCMOM</code> are specialized functions which draw multivariate Normals with calculations of the mean and covariance matrix from other matrices.

F.9 Wishart Distribution

Parameters	Scaling \mathbf{A} (symmetric $n \times n$ matrix) and degrees of freedom (ν). This only has a proper density if $\nu > n - 1$ and \mathbf{A} is positive definite.
Kernel	$\exp\left(-\frac{1}{2}\text{trace}(\mathbf{A}^{-1}\mathbf{X})\right) \mathbf{X} ^{\frac{1}{2}(\nu-n-1)}$
Support	Positive definite symmetric matrices
Mean	$\nu \mathbf{A}$
Main uses	Prior, exact and approximate posterior for the precision matrix (inverse of covariance matrix) of residuals in a multivariate regression, though that is mainly in the inverse form (Appendix F.10) since that would be the distribution of the covariance matrix itself.
Random Draws	<p><code>%RANWISHART (n, nu)</code> draws a single $n \times n$ Wishart matrix with $\mathbf{A} = \mathbf{I}$ and degrees of freedom ν.</p> <p><code>%RANWISHARTF (F, nu)</code> draws a single $n \times n$ Wishart matrix with $\mathbf{A} = \mathbf{F}\mathbf{F}'$ and degrees of freedom ν. \mathbf{F} can be any factor of \mathbf{A}, but would typically be computed as the Cholesky factor using <code>%DECOMP</code>.</p>

F.10 Inverse Wishart Distribution

Parameters	Scaling Ψ (symmetric $n \times n$ matrix) and degrees of freedom (ν). This only has a proper density if $\nu > n - 1$ and Ψ is positive definite.
Kernel	$\exp\left(-\frac{1}{2}\text{trace}(\Psi\mathbf{X}^{-1})\right) \mathbf{X} ^{-\frac{1}{2}(\nu+n+1)}$
Support	Positive definite symmetric matrices
Mean	$\frac{1}{\nu-n-1}\Psi$
Main uses	Prior, exact and approximate posterior for the covariance matrix of residuals in a multivariate regression with Gaussian residuals.
Diffuse versions	The density function is improper if $\nu < n - 1$, but the improper prior with $\nu = 0$ and $\Psi = 0$ has kernel $ \mathbf{X} ^{-\frac{1}{2}(n+1)}$ which forms the Jeffrey's prior for inference on the covariance matrix.
Combining Densities	If $\mathbf{X} \sim IW(\Psi_1, \nu_1)$ and $\mathbf{X} \sim IW(\Psi_2, \nu_2)$ are inverse Wishart densities for \mathbf{X} , then the posterior from combining them has $\mathbf{X} \sim IW(\Psi_1 + \Psi_2, \nu_1 + \nu_2 + n + 1)$.
Random Draws	<code>%RANWISHARTI(F, nu)</code> draws a single $n \times n$ inverse Wishart matrix with $\mathbf{F}\mathbf{F}' = \Psi^{-1}$ and degrees of freedom ν . Note that \mathbf{F} needs to be a factor of the <i>inverse</i> . \mathbf{F} can be any factor matrix, but is typically the Cholesky factor, computed using <code>%DECOMP</code> .
Notes	The basic result has the data evidence on the covariance matrix of Gaussian residuals summarized as an inverse Wishart with $\Psi = T\hat{\Sigma}$, where T is the number of observations and $\hat{\Sigma}$ the sample covariance matrix of residuals (thus Ψ itself is the sum of the outer products of the residuals). The degrees of freedom for the inverse Wishart from the data itself are typically $T - (n + 1)$ (sometimes less some additional adjustments for regressors, depending upon the form of conditioning). The $n + 1$ is needed because you don't really have the ability to estimate a covariance matrix until you have that many observations. Combining data with the Jeffrey's prior "corrects" the degrees of freedom so the posterior value of $\nu = T$.

An informative prior is generally based upon a prior belief on the value of \mathbf{X} . Because \mathbf{X} is typically the covariance matrix Σ , call this Σ_0 . The corresponding value of Ψ for this is $\alpha\Sigma_0$ where the prior degrees of freedom are $\alpha + n + 1$. Combined with data, this gives an inverse Wishart with $T + \alpha$ degrees of freedom and Ψ matrix which is $T\hat{\Sigma} + \alpha\Sigma_0$, thus (roughly) sample and non-sample information on Σ weighted by the number of actual and “dummy” observations.

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

`<https://fsf.org/>`

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated

herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or

XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that

carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together

with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a

translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document

does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Bibliography

- Andrews D W K 1993 *Econometrica* **61**(4), 821–856.
- Andrews D W K & Ploberger W 1994 *Econometrica* **62**(6), 1383–1414.
- Balke N 2000 *Review of Economics and Statistics* **82**(2), 344–349.
- Balke N & Fomby T 1997 *International Economic Review* **38**(3), 627–645.
- Baltagi B 2002 *Econometrics* 3rd edn Springer.
- Cai J 1994 *Journal of Business and Economic Statistics* **12**(3), 309–316.
- Chan F & McAleer M 2003 *Applied Financial Economics* **13**(8), 581–592.
- Chib S 1996 *Journal of Econometrics* **75**(1), 79–97.
- Davies R B 1987 *Biometrika* **1987**, 33–43.
- Dempster A, Laird N & Rubin D 1977 *JRSS-B* **39**(1), 1–38.
- Diebold F X, Lee J H & Weinbach G 1994 in C Hargreaves, ed., ‘Nonstationary Time Series Analysis and Cointegration’ Oxford: Oxford University Press pp. 283–302.
- Dueker M S 1997 *Journal of Business and Economic Statistics* **15**(1), 26–34.
- Ehrmann M, Ellison M & Valla N 2003 *Economics Letters* **78**(3), 295–299.
- Enders W 2015 *Applied Econometric Time Series* 4th edn Wiley.
- Enders W & Granger C W J 1998 *Journal of Business and Economic Statistics* **16**(3), 304–311.
- Filardo A 1994 *Journal of Business and Economic Statistics* **12**(3), 299–308.
- Fruehwirth-Schnatter S 2006 *Finite Mixture and Markov Switching Models* Statistics Springer.
- Gray S F 1996 *Journal of Financial Economics* **42**(1), 27–62.
- Hamilton J 1989 *Econometrica* **57**(2), 357–384.
- Hamilton J 1994 *Time Series Analysis* Princeton: Princeton University Press.
- Hamilton J & Susmel R 1994 *Journal of Econometrics* **vol 64**(1-2), 307–333.

- Hansen B 1992 *Journal of Policy Modeling* **14**(4), 517–533.
- Hansen B 1997 *Journal of Business and Economic Statistics* **15**(1), 60–67.
- Hansen B 2000 *Journal of Econometrics* **97**(1), 93–115.
- Hansen B & Seo B 2002 *Journal of Econometrics* **110**(2), 293–318.
- Inclan C & Tiao G 1994 *Journal of American Statistical Association* **89**(427), 913–923.
- Kiefer N M 1978 *Econometrica* **46**(2), 427–434.
- Kiefer N M & Wolfowitz J 1956 *Annals of Mathematical Statistics* **27**(4), 887–906.
- Kim C J 1994 *Journal of Econometrics* **60**(1-2), 1–22.
- Kim C J & Nelson C R 1999 *State-Space Models with Regime Switching* MIT Press.
- Koop G, Pesaran H & Potter S 1996 *Journal of Econometrics* **74**(1), 119–147.
- Lam P s 1990 *Journal of Monetary Economics* **26**(3), 409–432.
- Nyblom J 1989 *Journal of American Statistical Association* **84**(405), 223–230.
- Patton A J 2011 *Journal of Econometrics* **160**(1), 246–256.
- Quandt R E 1960 *Journal of American Statistical Association* **55**(290), 324–330.
- Terasvirta T 1994 *Journal of American Statistical Association* **89**(425), 208–218.
- Tsay R 1989 *Journal of American Statistical Association* **84**(405), 231–240.
- Tsay R 1998 *Journal of American Statistical Association* **93**(443), 1188–1202.
- Tsay R 2010 *Analysis of Financial Time Series* Wiley.

Index

- Absorbing state, 316
- Andrews, D., 32, 35
- Andrews-Ploberger statistic, 35
- Andrews-Quandt statistic, 35
- @APBreakTest** procedure, 33
- @ARCHTEST** procedure, 20
- Arranged autoregression test, 56
 - multivariate, 78
- Balke, N., 72, 80, 97, 99, 104
- BALT3P193.RPF example, 38
- Bernoulli distribution, 325
- Beta distribution, 138, 320
- BETASYS coefficient matrix, 194
- BFGS algorithm, 137
- BHHH algorithm, 137
- %BICDF function, 326
- BOOT** instruction, 91
- Bootstrap
 - fixed regressor, 43, 83
 - residual, 86
- BOXJENK** instruction, 6
 - OUTLIER option, 39
- Brownian bridge, 13
- Burn-in, 313
- Cai, J., 268
- %CDF function, 319
- Chi-squared distribution, 322
- Chib, S., 153
- CMOMENT** instruction, 18
 - ZXMATRIX option, 36
- CONSTANT.RPF example, 18
- Crash (trend break), 3
- Davies, R., 134
- %DENSITY function, 319
- DERIVES option, 16
- @DFSETUP** procedure, 284
- Dickey-Fuller test
 - arranged, 73
- Dirichlet distribution, 139, 321
- Dueker, M. S., 280
- Ehrmann, M., 196
- Ellison, M., 196
- Enders, W., 54
- Endogenous break, 1
- %EQNXVECTOR function, 10
- Eventual forecast function
 - for TVECM, 76
- FFBS algorithm, 153
- Fixed regressor bootstrap, 43, 83
- Fluctuation tests, 13
- @FLUX** procedure, 16, 28
- Fomby, T., 72, 97, 99
- FORECAST** instruction
 - eventual forecast, 76
 - PATHS option, 63
 - with TAR model, 59
- Forward filtering, 152
- Fruehwirth-Schnatter, S., 132, 140
- FUNCTION**
 - user-defined, 132
- Gamma distribution, 323, 324
- GAMMASYS coefficient matrix, 194
- GARCH** instruction, 16
 - with mean shift, 21
 - with variance shifts, 23
- Gibbs sampling, 313
- GIRF, 62
- Granger, C.W.J., 54
- Gray, S. F., 280
- Hamilton, J., 177, 214, 232, 268
- Hansen, B., 18, 35, 43, 94
- @HansenSeo** procedure, 95
- Hidden Markov model, 148
- @HSLMTest** procedure, 96
- ICSS algorithm, 19
- Impulse response

- generalized, 62
 - non-linear, 62
- Inclan, C., 19
- Independence chain M-H, 314
- %INDEX function, 140
- Inverse Wishart distribution, 328
- %INVNORMAL function, 319
- Kiefer, N., 132
- Kim filter, 233
- Kim, C.-J., 233, 250
- Label switching, 131, 140
- Lam, P.-S., 232
- %LOGDENSITY function, 319, 326
- %LOGISTIC function, 115
- Logistic parameterization, 160
- M-H, *see* Metropolis-Hastings
- Markov chain, 148
- Markov Chain Monte Carlo, 313
- MAXIMIZE** instruction
 - REJECT option, 133
 - START option, 134
- MCMC, *see* Markov Chain Monte Carlo
- @MCMCPOSTPROC** procedure, 315
- %MCREGIME function, 151
- Measurement equation, 231
- Metropolis-Hastings, 314
- Mixture model, 131
- @MSDrawP** procedure, 165
- @MSEMSetpStd** procedure, 164
- @MSFilterInit** procedure, 160
- %MSLagState function, 270
- %MSLOGISTICP function, 161
- %MSPLOGISTIC function, 160
- %MSPROB function, 152
- %MSREGFVec function, 177
- MSREGIME variable, 154
- @MSRegInitial** procedure, 178
- %MSRegInitVariances function, 178
- @MSRegParmset** procedure, 178
- @MSRegRelabel** procedure, 183
- @MSRegResids** procedure, 182
- @MSRegression** procedure, 176
- MSREGRESSION.SRC file, 176
- @MSSample** procedure, 154
- @MSSetup** procedure, 151
- @MSSSmoothed** procedure, 153
- @MSSysRegFilter** procedure, 202
- @MSSysRegFixedCMOM** procedure, 205
- @MSSysRegFixResids** procedure, 205
- %MSSysRegFVec function, 194
- @MSSysRegInitial** procedure, 196
- %MSSysRegInitVariances function, 197
- MSSysRegModel model, 202
- MSSysRegNFix variable, 205
- @MSSysRegResids** procedure, 201
- @MSSysRegression** procedure, 193
- @MSSysRegSetModel** procedure, 195
- @MSSysRegSwitchCMOM** procedure, 205
- %MSUPDATE function, 152
- @MSVARInitial** procedure, 217
- @MSVARPARMSET** procedure, 217
- MU VECTOR of means
 - in MSVAR, 216
- Multi-Move Sampling, 153
- Multivariate Normal distribution, 326
- Nelson, C. R., 250
- NEXPAND variable, 270
- %NFREE variable, 10
- NLAGS variable, 151
- NLLS** instruction, 114
- Non-linear impulse response, 62
- Normal distribution, 319
 - multivariate, 326
- NREGIMES variable, 151
- %NREGSYSTEM variable, 9
- Nyblom, J., 15
- %NYBLOMTEST function, 16
- Observation equation, 231
- ONEBREAK.RPF example, 34
- Outliers
 - additive, 5
 - innovational, 5

Overflow, 156, 315

P matrix

for transition probabilities, 151

PANEL instruction, 90

Particle filter, 289

PHI lag coefficients

in MSVAR, 216

PHIV lag coefficients

in MSVAR, 216

Ploberger, W., 35

Prediction step

Markov switching, 151

Probability distributions

Bernoulli, 325

beta, 138, 320

Dirichlet, 139, 321

gamma, 323

inverse chi-squared, 322

inverse gamma, 324

inverse Wishart, 328

multivariate normal, 326

normal, 319

Wishart, 327

PT_T filtered probabilities, 153

PT_T1 predicted probabilities, 153

%RAN function, 319

%RANBRANCH function, 138, 325

Random walk M-H, 315

%RANMAT function, 319, 326

%RANMVNORMAL function, 326

%RANWISHART function, 327

%RANWISHARTF function, 327

%RANWISHARTI function, 328

@RegHBreak procedure, 33

@REGStrTest procedure, 117

REJECT option, 133

Residuals

studentized, 38

RLS instruction, 56, 73

Rolling sample analysis, 26

%ROUND function, 65

Round-off problems, 54

Saddlepoint approximation, 16

Sampling

Multi-move, 153

Single-move, 154

Seo, B., 94

SERIES[VECT], 137

SETAR model, 54

SIGMA covariance matrix

in MSVAR, 216

SIGMAV covariance matrices, 194

in MSVAR, 216

Single-Move sampling, 154

SSTATS instruction, 137

@STABTEST procedure, 18

STAR model, 114

@STARTEST procedure, 116

State equation, 231

State variable, 231

Structural break, 1

Studentized residuals, 38

Susmel, R., 268

SWARCH.RPF example, 269

Sweep

in MCMC, 313

SWEEP instruction, 9, 74, 79, 82

TAR model

Self-Exciting, 54

@TAR procedure, 57

Terasvirta, T., 115

@ThreshTest procedure, 33, 56

Tiao, G., 19

Transition equation, 231

Transition matrix, 150

Tsay, R., 56, 77, 102, 177, 275

@TSAYTEST procedure, 56, 72

Underflow, 156, 315

Update step

Markov switching, 152

Valla, N., 196

Wishart distribution, 327

inverse, 328

Wolfowitz, J., 132

%ZTEST function, 319