

# **RATS Handbook for Bayesian Econometrics**

Thomas A. Doan  
Estima

4th Revision  
December 16, 2024

Copyright © 2024 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

---

<b>Preface</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Bayesian Statistics: An Overview . . . . .	1
1.2 Single Parameter–Brute Force . . . . .	4
1.3 RATS Tips and Tricks . . . . .	7
1.1 Brute Force: Analyzing on a Grid . . . . .	10
<b>2 Linear Regression Model with Conjugate Prior</b>	<b>11</b>
2.1 LRM with a Single Variable . . . . .	11
2.2 Normal Linear Model: Theory . . . . .	14
2.3 Using Cross Product Matrices . . . . .	17
2.4 Calculations . . . . .	18
2.5 Simulations . . . . .	21
2.6 RATS Tips and Tricks . . . . .	23
2.1 Linear Model: Single Variable . . . . .	27
2.2 Multiple Regression: Conjugate Prior . . . . .	28
2.3 Multiple Regression with Conjugate Prior: Simulations . . . . .	30
<b>3 Normal Linear Model with Independent Prior</b>	<b>32</b>
3.1 Theory . . . . .	32
3.2 Calculations . . . . .	35
3.3 Diagnostics . . . . .	37
3.4 The Bayesian Approach to Hypothesis Testing . . . . .	39
3.5 Hypothesis Testing with the Linear Model . . . . .	40
3.6 RATS Tips and Tricks . . . . .	43
3.1 Linear Model with Independent Prior . . . . .	45
3.2 Linear Regression: Conjugate Prior with Restrictions . . . . .	47

<b>4</b>	<b>Nonlinear Regression: Introduction to Metropolis-Hastings</b>	<b>50</b>
4.1	Theory . . . . .	50
4.2	Calculations . . . . .	54
4.3	RATS Tips and Tricks . . . . .	57
4.1	Non-linear Regression: Random Walk MH . . . . .	59
4.2	Non-linear Regression: Independence MH . . . . .	60
<b>5</b>	<b>Linear Regression with Non-Spherical Errors</b>	<b>63</b>
5.1	Heteroscedasticity of Known Form . . . . .	63
5.2	Heteroscedasticity of Unknown Form. . . . .	69
5.3	Serially Correlated Errors. . . . .	71
5.4	Seemingly Unrelated Regressions . . . . .	73
5.5	RATS Tips and Tricks . . . . .	78
5.1	Heteroscedastic errors with a known form . . . . .	80
5.2	Heteroscedastic errors with a unknown functional form . . . . .	82
5.3	Linear regression with AR(1) errors . . . . .	84
5.4	Seemingly unrelated regression . . . . .	87
<b>6</b>	<b>Vector Autoregressions</b>	<b>89</b>
6.1	Flat Prior . . . . .	89
6.2	Antithetic Acceleration . . . . .	94
6.3	An Application: Blanchard-Quah Model . . . . .	96
6.4	Structural VAR's . . . . .	100
6.4.1	Waggoner-Zha Sampler . . . . .	106
6.5	Informative Prior for Univariate Autoregressions . . . . .	108
6.6	VAR with Informative Prior (BVAR) . . . . .	113
6.7	RATS Tips and Tricks . . . . .	116
6.1	Antithetic Acceleration . . . . .	118
6.2	VAR with flat prior . . . . .	120
6.3	Structural VAR: Importance sampling . . . . .	122
6.4	Structural VAR: Random Walk MH . . . . .	125
6.5	Structural VAR: Independence Chain MH . . . . .	127
6.6	Univariate autoregression with prior . . . . .	130
6.7	Univariate Autoregression: Out-of-sample forecast performance . . . . .	132
6.8	Bayesian VAR: Gibbs sampling . . . . .	134

<b>7</b>	<b>Cross Section and Panel Data</b>	<b>137</b>
7.1	Panel Data. . . . .	137
7.2	Probit and Tobit Models. . . . .	142
7.3	RATS Tips and Tricks . . . . .	146
7.1	Panel data: LSDV . . . . .	147
7.2	Panel data: Fixed Effects . . . . .	148
7.3	Panel data: Random Effects (hierarchical prior) . . . . .	150
7.4	Panel data: Random coefficients model . . . . .	153
7.5	Tobit model . . . . .	155
7.6	Probit model . . . . .	157
<b>8</b>	<b>State Space Models</b>	<b>159</b>
8.1	State-space model: Independence MH . . . . .	167
8.2	State Space Model: Gibbs sampling . . . . .	169
8.3	State space model: Time-varying coefficients . . . . .	171
<b>A</b>	<b>General Information on RATS Instructions</b>	<b>174</b>
<b>B</b>	<b>Probability Distributions</b>	<b>175</b>
B.1	Uniform . . . . .	175
B.2	Univariate Normal . . . . .	176
B.3	Univariate Student ( $t$ ) . . . . .	177
B.4	Beta distribution . . . . .	178
B.5	Chi-Squared Distribution . . . . .	179
B.6	(Scaled) Inverse Chi-Squared Distribution . . . . .	180
B.7	Gamma Distribution . . . . .	181
B.8	Inverse Gamma Distribution . . . . .	182
B.9	Bernoulli Distribution . . . . .	183
B.10	Multivariate Normal . . . . .	184
B.11	Multivariate Student ( $t$ ). . . . .	185
B.12	Wishart Distribution . . . . .	186
B.13	Inverse Wishart Distribution . . . . .	187
<b>C</b>	<b>Properties of Multivariate Normals</b>	<b>189</b>

Contents	iv
<b>D Non-Standard Matrix Calculations</b>	<b>192</b>
<b>Bibliography</b>	<b>195</b>
<b>Index</b>	<b>196</b>

---

## Preface

---

This workbook is based upon the content of the RATS e-course on Bayesian Econometrics, offered in spring 2009. It covers most of the most important methods now used in Bayesian analysis in econometrics, including Gibbs sampling, Metropolis-Hastings and importance sampling. The applications are to a broad range of topics, include time series, cross-section and panel data. It assumes that the user is comfortable with such basic instructions as **COMPUTE**, **DISPLAY**, **GRAPH**, **SCATTER** and **LINREG**, and can use simple programming techniques such as **DO** loops. In each chapter, there is a *Tips and Tricks* section which covers in greater detail any functions or instructions that might be unfamiliar.

The presentation is based largely on Gary Koop's (2003) *Bayesian Econometrics*. We've added to that in several areas, with a chapter on vector autoregressions, and examples from the literature for panel, cross-sectional data and state-space models. In most cases, we've included much of the statistical derivations from the book, presented in a way to highlight the calculations as they are done with RATS, so even those without the book can benefit.

We use bold-faced Courier (for instance, **LINREG**) for any use of RATS instruction names within the main text, and non-bolded Courier (%RANMVT) 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 as separate files are available separately.

This was revised in September 2012 to bring references up-to-date for RATS version 8, to improve the index and to fix some minor errors in some of the early chapters.

## Introduction

### 1.1 Bayesian Statistics: An Overview

Bayesian statistics assumes that the observed data ( $\mathbf{Y}$ ) are generated by a probability model which depends upon some unknown set of parameters which we'll call  $\theta$ . From this, we generate the likelihood function  $p(\mathbf{Y}|\theta)$ . This is exactly the same likelihood function as is used in most applications of conventional (non-Bayesian) econometrics. In that methodology, we maximize  $g(\theta) = \log p(\mathbf{Y}|\theta)$  over  $\theta$ . The *likelihood principle* is that a reasonable estimator for  $\theta$  is that value that makes the observed data as likely as possible. Under the proper conditions, this  $\hat{\theta}_{ML}$  can be shown to be consistent and asymptotically Normal. The final product of the estimation can be summarized (roughly) as

$$\hat{\theta}_{ML} - \theta \sim N(0, \mathbf{Q}) \quad (1.1)$$

Bayesian statistics uses the likelihood differently. Using Bayes' theorem, it's "inverted" to create a probability distribution for  $\theta$ . This takes the form

$$p(\theta|\mathbf{Y}) = \frac{p(\mathbf{Y}|\theta)p(\theta)}{p(\mathbf{Y})}$$

where  $p(\theta)$  is a *prior* (pre-data) density on  $\theta$  and

$$p(\mathbf{Y}) = \int p(\mathbf{Y}|\theta)p(\theta)d\theta$$

$p(\mathbf{Y})$  (which is called the *marginal likelihood*) is usually a very ugly expression if it can be computed at all (other than by approximating the integral). It has its uses in comparing models, but in analyzing  $\theta$  given a single model, it can be ignored, since it doesn't depend upon any single value of  $\theta$ . The Bayesian *posterior* (post-data) density on  $\theta$  is thus generally summarized by

$$p(\theta|\mathbf{Y}) \propto p(\mathbf{Y}|\theta)p(\theta)$$

The right side has a non-negative value everywhere, and if it has a finite integral,  $p(\theta|\mathbf{Y})$  will be a proper density function.

One possibility is that the posterior density will look something like

$$\theta \sim N(\hat{\theta}_{ML}, \mathbf{Q}) \quad (1.2)$$

Equations (1.1) and (1.2) say, effectively, the same thing. However, they are different views of similar information. In conventional econometrics, the interpretation of (1.1) is that  $\theta$  is a fixed (though unknown) vector and this is a

description of the sampling distribution of  $\hat{\theta}_{ML}$  over the space of  $Y$ 's. In (1.2), the Bayesian interpretation is that  $\hat{\theta}_{ML}$  is a vector calculated from the one data set that we actually have and (1.2) is a description of the uncertainty we have about the value of  $\theta$ .

In the early 20<sup>th</sup> century, when modern mathematical statistics was being developed, it was widely considered to be unreasonable to act as if there were a probability distribution over the fixed value  $\theta$  (and in Bayesian methods, there are two of these: the prior and the posterior). However, the development of game theory and statistical decision theory in the 1950's rehabilitated Bayesian methods. After all, in the end, the point of statistical analysis is to help make decisions. Should an economic theory be discarded? Should we sell a stock? Should taxes be raised? If you, as an econometrician, aren't making the decision yourself, you're advising someone else (or the profession as a whole). The key (and rather deep) result in statistical decision theory is that, if you have a loss function based upon the unknown  $\theta$ , the "rational" decision is to minimize that loss over the posterior distribution for  $\theta$ . The results of this form are known as *Complete Class Theorems*. A rather imprecise way of stating this is that any "reasonable" decision process has a Bayesian justification.

Despite this, Bayesian methods were still not widely used until more recently. There were two main reasons for this:

### Tractability

There are only a handful of cases where  $p(Y|\theta)p(\theta)$  has a form which produces a closed form function for  $\theta$ . First off, the likelihood by itself will only have a closed form expression for  $\theta$  for a very limited number of possible data generating processes. In econometrics, this really only happens for linear regressions and systems of equations with Normally distributed errors. For GARCH models, non-linear regressions, probit models, etc.,  $p(Y|\theta)$  will be an implicit function of  $\theta$ , and can only be evaluated for different values by a function evaluation over the whole sample. And even if we have one of those cases where the likelihood can be inverted successfully, there are only very limited forms of priors for which the product is a well-known distribution.

### Ignorance

How do we know enough about  $\theta$  pre-data to craft a reasonable prior?

The first of these has been overcome by the sheer power of modern computers and the development of new simulation techniques (in particular, Gibbs sampling and Metropolis-Hastings). This allows simulations to arbitrarily high degrees of accuracy in a reasonable amount of time.

In the case of the second, it's now better understood that the job of the prior is to help where the data have little information, and get out of the way where the data evidence is strong.



The tractability argument is now, in some cases, flipped over to favor Bayesian methods over classical. For instance, it's almost impossible to get reasonable results estimating a DSGE (Dynamic Stochastic General Equilibrium) model by maximum likelihood. The data simply have little useful information about some of the underlying parameters. The proper use of priors allows the parameter space to be restricted to values which make economic sense.

As another example, some forms of Markov switching models have unconditional likelihood functions which are quite messy because the value at time  $t$  depends upon the entire previous history of the states. Bayesian sampling methods have an easier time with this because they can draw a different history of the states each pass.

Finally, the conventional result (1.1) depends upon assumptions which may not be met in practice. Ideally for this result, the log likelihood should be globally concave. However, many important types of models have multiple modes in the likelihood function (very common in “switching” models). Reporting just a single mode can be misleading when there are others which have nearly the same likelihood. Conventional methods might even miss the existence of other peaks, and, in fact, could easily end up missing the actual highest peak. Modern Bayesian methods can often discover ill-behaved likelihoods where simple optimization procedures can't.

It's probably safe to say that most contemporary econometricians are comfortable with both conventional and Bayesian methods. One way to think about Bayesian methods is that they explore and describe the likelihood, the prior being used to highlight certain aspects of this. Where conventional methods work well, the underlying result is that

$$\log p(\theta) \approx \log p(\hat{\theta}_{ML}) - \frac{1}{2} \left( \theta - \hat{\theta}_{ML} \right)' \mathbf{Q}^{-1} \left( \theta - \hat{\theta}_{ML} \right)$$

If this describes the likelihood quite well, one would expect that most priors would reveal that. Any but the most dogmatic Bayesians can use and make sense of the (simpler) conventional analysis in this case.

## 1.2 Single Parameter–Brute Force

With a single parameter, it's possible to use graphs and approximations on a grid to analyze the model. This becomes less useful once you get to more than one parameter, and is almost worthless with more than about four. However, some of the more involved simulation methods need to attack parameters one at a time, so knowing how to work with these methods is still handy.

Our particular example is to analyze the median of a Cauchy distribution. (Cauchy's have no mean, thus the central measure is the median).

The density function for an (otherwise standard) Cauchy with median  $\theta$  is

$$p(x|\theta) = \frac{1}{\pi} \frac{1}{1 + (x - \theta)^2}$$

The likelihood function for a data set consisting of independent draws from this ( $\mathbf{Y} = \{x_1, \dots, x_N\}$ ) is thus

$$p(\mathbf{Y}|\theta) = \prod_{i=1}^N \frac{1}{\pi} \frac{1}{1 + (x_i - \theta)^2} \quad (1.3)$$

A couple of things to note about this. First, in computing the likelihood for Bayesian calculations, you can ignore integrating constants like  $\frac{1}{\pi}$ . These exist to make the density function of the data integrate to 1. Once we flip the likelihood around, they really don't help, and just wash into the  $p(\mathbf{Y})$  factor. The density functions built into RATS all include those for other purposes (such as conventional maximum likelihood), but if you need to compute your own density you can leave them out.

Second, if you compute (1.3) directly, you may very well underflow the computer. Underflow occurs when a number gets so close to zero that there is no representation for it. On machines with 64-bit floating point (which has been standard for the last 20+ years), this is around  $10^{-314}$ . This seems really tiny, but if you have 315 observations with densities on the order of .1, you'll underflow. The density on a Cauchy is so diffuse that you could easily underflow with 100 observations.

Conventional maximum likelihood avoids this problem by working entirely in logs. In logs,  $10^{-314}$  is a nice safe -720 (roughly). For Bayesian methods, however, we need the actual likelihood function, not its log. So how do we get around this? We again use the fact that we really only need something proportional to the actual likelihood. We can do the calculations in underflow-safe logs, and make some adjustment for scale before exp'ing back.

We generate the data with the following. The seed ensures that everyone running this gets the same data:

```
seed 53430
set x 1 20 = 0.5+%(rant(1.0))
```

This generates a series of 20 independent Cauchys.<sup>1</sup> We're going to graph everything on a grid running over the interval  $[-5.0, 5.0]$ . This is more than adequate: the likelihood is effectively zero beyond -3 and 3. In order to do scatter graphs, we need the grid in a series, so we do the following:

```
set thetax 1 200 = -5+.05*t
```

This does a grid from -5 to 5 by steps of .05. You can make this finer or coarser by increasing (decreasing) the number of points and decreasing (increasing) the .05 step size.

As is typical when you step outside the realm of Normally distributed errors, the likelihood function has no closed form inverse for  $\theta$ . So we'll have to evaluate the likelihood at each point in the grid by summing the log likelihood elements. That's most easily done with:

```
set dataf 1 200 = 0.0
do test=1,200
  sstats 1 20 -log((1+(x-thetax(test))^2))>>dataf(test)
end do test
```

See *Tips and Tricks* (Section 1.3) if you're not familiar with **SSTATS**.

At the end of this, we have the log likelihood for each point on our grid. We now need to convert this to a (scaled) non-logged value. The most straightforward way to do this is to compute the maximum of the log likelihoods (on the grid), and subtract that off before exp'ing.

```
ext dataf 1 200
set dataf 1 200 = exp(dataf-%maximum)
scatter(footer="Likelihood function (rescaled)",style=line)
# thetax dataf
```

The rescaled likelihood will show a maximum value of 1. Note that it's quite possible for the exp(..) calculation to underflow for the tail values. (That won't happen here, but could very easily happen with a thinner tailed distribution like the Normal). That's OK. Underflows are automatically turned into zeros. The zones with really low (relative) probabilities can be set to zero. What we need to avoid is underflowing the zones with *high* relative probabilities.

Note, by the way, that in some cases we have the opposite problem – the log likelihoods might *overflow* rather than underflow when exp'ed. (This can happen if the scale of the data is compressed). The same procedure works to avoid problems with that.

We're going to try two priors – a uniform across the  $[-5.0, 5.0]$  interval (effectively, an “I'm clueless” prior), and a much tighter standard Normal with mean 0.

---

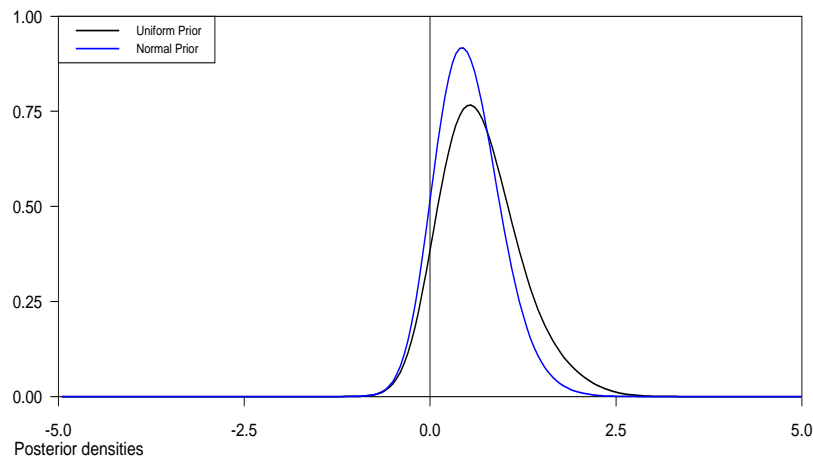
<sup>1</sup>A standard Cauchy is the same as a  $t$  with 1 degree of freedom.

In order to compare the posterior densities, we'll need to estimate that integrating constant so that we'll have actual densities. That's most easily done by using the **SSTATS** instruction to sum the (raw) posterior densities and scale by the grid width. This gives a crude integral. (It's possible to do a more accurate Simpson's rule, and you probably would want to do that if this were a more serious exercise).

Note that the uniform prior itself isn't scaled to be a density. Again, any scale factor there would wash, so we can be lazy.

```
set priorfu 1 200 = 1.0
set postfu 1 200 = priorfu*dataf
sstats      1 200 postfu*.05>>sumpostu
set postfu 1 200 = postfu/sumpostu
set priorfn 1 200 = exp(%logdensity(1.0,thetax))
set postfn 1 200 = priorfn*dataf
sstats 1 200 postfn*.05>>sumpostn
set postfn 1 200 = postfn/sumpostn
*
scatter(footer="Posterior densities",key=upleft,$
  klabels=||"Uniform Prior","Normal Prior"||,style=lines) 2
# thetax postfu
# thetax postfn
```

Note that the two posterior densities really aren't all that different. The modes are quite similar, with the Normal just shifting a bit towards zero — the main change is that the Normal prior pulls some mass out of the right tail. Even though the uniform prior gave most of its mass to values that the data found quite improbable, the end result is similar. This is quite common — the shape of the prior really matters only in the zone where the likelihood is high.



## 1.3 RATS Tips and Tricks

### Random Numbers and SEED

**SEED** allows you to control the set of random numbers generated by a program. We want to do that here because we want everyone running the program to get the same results. In practice, you would use it mainly in two situations:

1. You're trying to debug a program and need to control the random numbers in order to have results that are comparable from test to test. Note, however, if you make *any* change in the way that the numbers are drawn, the sequence of numbers will change.
2. You've written a "final" version of a paper and want to be able to reproduce the results exactly in case you need to regenerate graphs or tables.

Note that although it seems as if you should want to use a "random-looking" seed, it really doesn't matter: RATS scrambles the value you input quite a bit in generating the internal seed.

### The Instruction SSTATS

**SSTATS** is a handy instruction which can be used to compute the sum (or mean or maximum, etc.) of one or more general expressions. Since it accepts a formula, you don't have to take the extra step of generating a separate series with the needed values. By default, it does the sum, which is what we do twice in Example 1.1.

It can be used to answer some (apparently) quite complicated questions. For instance,

```
sstats (min, smpl=peak+trough) start1 endl t>>tp0
```

gives `tp0` as the smallest entry for which either the series `peak` or `trough` (both dummies) is "true".

```
sstats 1 nobsp1*y>>plys p1>>pls p2*y>>p2ys p2>>p2s
```

computes four parallel sums. Without the **SSTATS**, this would require about eight separate instructions.

```
sstats / date<>date{1}>>daycount
```

computes the number of days in a data set with intra-day data. `date<>date1` is 1 when the value of `date(t)` is different from `date(t-1)` and 0 if it's the same. So the **SSTATS** is summing the number of changes in the date series.

## Initializing series entries

We use the instruction:

```
set dataf 1 200 = 0.0
```

before the **DO** loop that computes the posterior densities for each entry in the grid. We need that because the `dataf` is being set one element at a time by

```
sstats 1 20 -log((1+(x-thetax(test))^2))>>dataf(test)
```

This last line by itself wouldn't work, since `dataf` could be either a **SERIES** or a **VECTOR**. So one point of the **SET** instruction is to make sure it's understood that `dataf` is a **SERIES**. Now, there are other ways to do that, for instance:

```
declare series dataf
```

We create this using the **SET** instruction to get the proper range defined for the series. Different series can have different lengths depending upon how they are used. In this chapter's example, there are two ranges: the 1 to 20 for the data set and the 1 to 200 for the calculations on the grid points. The initial **SET** makes `dataf` one of that second group.

## Commenting

Your first goal is always to get the calculations correct. However, if you have a program (or part of one) that you expect that you might need again, it's a good idea to add comments. If you have a part of your program which you're not sure is correct, commenting it can often help you spot errors. (If you can't explain why you're doing something, that might be a good sign that you're doing it wrong).

Now it's possible to overdo the comments. You may have seen some programs with a very elaborate comment block like:

```
*****
*           This is a program that analyzes the population           *
*                   of kangaroos in New South Wales.                   *
*****
```

RATS provides several ways to help manage larger blocks of comments. The first are the comment block delimiters. Anything between `/*` and `*/` is a block of comments. So

```
/******
   This is a program that analyzes the population
   of kangaroos in New South Wales.
******/
```

You get a similar appearance, but you can more easily edit the comments.

If something not quite as pretty will do, you can use the *Format Comments* operation on the *Edit* menu. This takes a selected block of text and creates a

set of one or more comment lines of roughly equal length (except for the final one). If some of the lines are already comments, the \*’s and leading blanks are first stripped out before the lines get formatted. As an example, we pasted in the first few sentences from above. The result is:

```
* Your first goal is always to get the calculations correct. However, if
* you have a program (or part of one) that you expect that you might
* need again, it’s a good idea to add comments.
```

The desired line length can be set on the *Editor* tab in the *Preferences*. The comments in this book were set with a length of 70, which is a bit shorter than standard, but was chosen to fit the page. Note that the comment length isn’t a hard limit. In order to roughly balance the lines, it might go a bit above that.

There are also a couple of other “comment” operations on the *Edit* menu. *Comment-Out Lines* takes the selected text and inserts \* at the beginning of each; converting each into a comment. The reverse is *Uncomment Lines*, which strips the lead \* off any line which has one. These allow you to take a block of lines out of the execution and put them back in.

## Example 1.1 Brute Force: Analyzing on a Grid

```

*
* Draw a test data set which is Cauchy centered at .5
*
seed 53430
set x 1 20 = 0.5+%rant(1.0)
*
set thetax 1 200 = -5+.05*t
set dataf 1 200 = 0.0
*
* Evaluate the (log) likelihood (ignoring constants) at each of the 200
* test values for thetax. This is done by using sstats to sum over the
* 20 data points.
*
do test=1,200
  sstats 1 20 -log((1+(x-thetax(test))^2))>>dataf(test)
end do test
*
ext dataf 1 200
set dataf 1 200 = exp(dataf-%maximum)
scatter(footer="Likelihood function (rescaled)",style=line)
# thetax dataf
*
* Uniform[-5,5] prior
*
set priorfu 1 200 = 1.0
set postfu 1 200 = priorfu*dataf
sstats 1 200 postfu*.05>>sumpostu
set postfu 1 200 = postfu/sumpostu
*
* N(0,1) prior
*
set priorfn 1 200 = exp(%logdensity(1.0,thetax))
set postfn 1 200 = priorfn*dataf
sstats 1 200 postfn*.05>>sumpostn
set postfn 1 200 = postfn/sumpostn
*
scatter(footer="Posterior densities",key=upleft,$
  klabels=||"Uniform Prior","Normal Prior"||,style=lines) 2
# thetax postfu
# thetax postfn

```



## Linear Regression Model with Conjugate Prior

### 2.1 LRM with a Single Variable

This discusses the implementation of the linear regression model with a single variable (no intercept), covered in Koop's Chapter 2. There are two parameters: the slope coefficient and the variance. The data generating process is

$$y_i = \beta x_i + \varepsilon_i, \varepsilon_i \sim N(0, \sigma^2) \text{ i.i.d.}$$

As mentioned in the book, it turns out to be simpler to work with the precision, which is the reciprocal of the variance. The letter  $h$  is usually used for this, so our parameter set is  $\{\beta, h\}$ .

The likelihood is

$$p(\mathbf{Y}|\theta) \propto \prod_{i=1}^N h^{1/2} \exp\left(-\frac{h}{2}(y_i - \beta x_i)^2\right) \quad (2.1)$$

where we've ignored constant factors which don't include either of the parameters. As shown in the book, this can be manipulated into the following form:

$$\left\{ h^{1/2} \exp\left[-\frac{h}{2}(\beta - \hat{\beta})^2 \sum_{i=1}^N x_i^2\right] \right\} \left\{ h^{(N-1)/2} \exp\left[-\frac{h(N-1)}{2s^2}\right] \right\} \quad (2.2)$$

where  $\hat{\beta}$  is the standard OLS estimator for  $\beta$ , and  $s^2$  is the standard (unbiased) estimator for  $\sigma^2$

$$s^2 = \frac{1}{N-1} \sum_{i=1}^N (y_i - \hat{\beta} x_i)^2$$

Note that we originally had a factor of  $h^{N/2}$  in (2.1). However, when we look at the slightly rewritten "Normal" factor in (2.2)

$$\exp\left[-\frac{1}{2}(\beta - \hat{\beta})^2 \left(h \sum_{i=1}^N x_i^2\right)\right]$$

we can recognize that as being the kernel for  $\beta$  having a Normal distribution with precision  $h \sum x_i^2$ . The integrating constant for that density is  $\left(h \sum x_i^2\right)^{1/2}$ . Now the  $\left(\sum x_i^2\right)^{1/2}$  part of that doesn't involve the parameters, and so can be

ignored.<sup>1</sup> However, the  $h^{1/2}$  *does* involve our second parameter so we need to grab  $h^{1/2}$  out of the  $h^{N/2}$  in order to complete the density. This leaves  $h^{(N-1)/2}$  for the second factor.<sup>2</sup>

Don't worry if this seems a bit confusing. Chasing through the precision factors in the linear model with the Normal-gamma prior is probably the trickiest part of this.

The most convenient prior is the natural *conjugate prior*, which is, like the data, a gamma on  $h$  combined with a Normal on  $\beta$  with its own variance (or actually precision) conditioned on  $h$ . This is the most convenient, but not necessarily the most realistic.  $h$ , after all, is an indication of how noisy the data are. Why your prior information about  $\beta$  should depend upon that is unclear. (The conditional part means that if the data turn out to be rather noisy, we would increase the variance of our prior, and if the data turn out to be quite precise, we would decrease the variance of the prior).

The derivation of the posterior is a bit tedious, involving completing the square on  $\beta$ . Note that the posterior mean for  $\beta$  is a weighted average of the prior and the data, with the weights being the relative precisions.

What I'd like to do is to show how the degrees of freedom on the gamma come about. To simplify the notation a bit, let's write the (inverted) likelihood as

$$\left\{ h^{1/2} \exp \left[ -\frac{h}{2} (\beta - \hat{\beta})^2 \hat{H} \right] \right\} \left\{ h^{(N-1)/2} \exp \left[ -\frac{h(N-1)}{2s^{-2}} \right] \right\}$$

where we've used  $\hat{H} = \sum_{i=1}^N x_i^2$ . The prior (with unnecessary factors omitted) is

$$\left\{ h^{1/2} \exp \left[ -\frac{h}{2} (\beta - \underline{\beta})^2 \underline{H} \right] \right\} \left\{ h^{(\underline{\nu}-1)/2} \exp \left[ -\frac{h(\underline{\nu}-1)}{2\underline{s}^{-2}} \right] \right\}$$

When we multiply these, the two  $\beta$  kernels combine to form one. However, each of these has an  $h^{1/2}$  factor, only one of which we will still need. The other needs to be pulled into the product of the gammas. Thus, we have

$$h^{1/2} h^{(N-1)/2} h^{\nu/2-1} = h^{(N+\nu)/2-1}$$

This generates and graphs artificial data for Koop's example 2.7.

---

<sup>1</sup>If we really want to be formal, we can multiply by  $\left(\sum x_i^2\right)^{1/2} \left(\sum x_i^2\right)^{-1/2}$ , use the +1/2 factor to complete our integrating constant and wash the other one into the constant of proportionality, but you really don't want to go through that every time.

<sup>2</sup>If you haven't really dealt with the gamma density before, this might be a good time to read about it in Appendix B.7.

```

compute n=50
compute h=1.0,b=2.0
*
* Generate x, e, and y
*
set x 1 n = %ran(1.0)
set e 1 n = %ran(h^-.5)
set y 1 n = b*x+e
*
scatter(footer="Artificial Data")
# x y

```

The easiest way to get the sufficient statistics for the likelihood is to just run a `LINREG`. The information for the (informative) prior and the data are then organized (probably *over-organized* here). We convert the variances into precisions (`hprior` and `hdata`) for the uncertainty regarding the coefficient.

```

linreg y
# x
*
compute bprior =1.5
compute vprior =.25
compute hprior =1.0/vprior
compute nuprior =10.0
compute ssqprior=1.0/1.0
*
compute bdata =%beta(1)
compute vdata =%xx(1,1)
compute hdata =1.0/vdata
compute nudata =%ndf
compute ssqdata =%seesq

```

Given those, we now combine the information to get the analogous statistics for the posterior. The formula below for `SSQPOST` is equivalent and ends up being a bit simpler to compute in the multivariate case.

```

compute hpost =hdata+hprior
compute vpost =1.0/hpost
compute bpost =(hdata*bdata+hprior*bprior)/hpost
compute nupost =%nobs+nuprior
compute ssqpost =(ssqdata*nudata+ssqprior*nuprior+$
(bdata-bpost)^2*hdata+(bprior-bpost)^2*hprior)/nupost

```

In order to graph the various densities, we again need to make a grid. This time, it will be on the interval  $[1, 3]$ . The marginal posterior is a  $t$ . Use the function `%LOGTDENSITYSTD` to evaluate this.

```

@GridSeries(from=1.0,to=3.0,size=.02,pts=gpts) bgraph
*
set dataf 1 gpts = $
  exp(%logtdensitystd(ssqdata *vdata ,bgraph-bdata ,nudata))
set priorf 1 gpts = $
  exp(%logtdensitystd(ssqprior*vprior,bgraph-bprior,nuprior))
set postf 1 gpts = $
  exp(%logtdensitystd(ssqpost *vpost ,bgraph-bpost ,nupost))
scatter(style=lines,vmin=0.0,$
  footer="Marginal Prior and Posterior for Beta",$
  key=upleft,klables=||"Prior","Posterior","Likelihood"||) 3
# bgraph priorf
# bgraph postf
# bgraph dataf

```

## 2.2 Normal Linear Model: Theory

This is the same basic model as was just analyzed, but with multiple regressors. The DGP (data generating process) is now

$$y_i = \mathbf{X}_i\beta + \varepsilon_i, \varepsilon_i \sim N(0, \sigma^2) \text{ i.i.d.}$$

where  $\mathbf{X}_i$  is the  $k$ -vector of explanatory variables at observation  $i$ . The likelihood function is proportional to

$$\sigma^{-N/2} \exp\left(-\frac{1}{2\sigma^2} (y - \mathbf{X}\beta)' (y - \mathbf{X}\beta)\right)$$

where  $N$  is the number of data points and  $y$  and  $\mathbf{X}$  are the stacked versions of the dependent and explanatory variables.

Again using the precision in place of the variance, and expanding the exponent, we can write the likelihood as

$$p(y|h, \beta) \propto h^{N/2} \exp\left(-\frac{h}{2} (y'y - 2y'\mathbf{X}\beta + \beta'\mathbf{X}'\mathbf{X}\beta)\right)$$

Using the result in Appendix C, the exponent can be rewritten as

$$-\frac{1}{2} \left( (\beta - \hat{\beta})' (h\mathbf{X}'\mathbf{X}) (\beta - \hat{\beta}) + h (y - \mathbf{X}\hat{\beta})' (y - \mathbf{X}\hat{\beta}) \right)$$

The second term in this doesn't depend upon  $\beta$ , only upon  $h$  and the sum of squared least squares residuals. Let's write

$$\hat{\varepsilon} = y - \mathbf{X}\hat{\beta}$$

When we exp up, we get

$$p(y|h, \beta) \propto h^{N/2} \exp\left(-\frac{1}{2} (\beta - \hat{\beta})' (h\mathbf{X}'\mathbf{X}) (\beta - \hat{\beta})\right) \times \exp\left(-\frac{h}{2} \hat{\varepsilon}'\hat{\varepsilon}\right)$$

By inspection, this has

$$\beta|h, y \sim N\left(\hat{\beta}, (h\mathbf{X}'\mathbf{X})^{-1}\right)$$

The (multiplicative) integrating constant for that density is

$$|h\mathbf{X}'\mathbf{X}|^{1/2} = h^{k/2} |\mathbf{X}'\mathbf{X}|^{1/2}$$

While  $|\mathbf{X}'\mathbf{X}|^{1/2}$  is just data and can be washed into the proportionality constant,  $h^{k/2}$  isn't. We need to pull some powers of  $h$  out of the  $h^{N/2}$  in order to fill out our multivariate Normal density. We can write the (inverted) likelihood as

$$\begin{aligned} p(h, \beta|y) &\propto h^{k/2} \exp\left(-\frac{1}{2}(\beta - \hat{\beta})'(h\mathbf{X}'\mathbf{X})(\beta - \hat{\beta})\right) \\ &\times h^{(N-k)/2} \exp\left(-\frac{h}{2}\hat{\varepsilon}'\hat{\varepsilon}\right) \end{aligned} \quad (2.3)$$

(Koop has a typo in 3.7 – the first power of  $h$  is  $h^{k/2}$  not  $h^{1/2}$ ).

The most convenient form of prior is the conjugate prior which takes the same basic form as the likelihood

$$\beta|h \sim N(\underline{\beta}, (h\mathbf{H})^{-1}), h \sim G(\nu, s^{-2})$$

which expands to

$$p(\beta, h) \propto h^{k/2} \exp\left(-\frac{1}{2}(\beta - \underline{\beta})'(h\mathbf{H})(\beta - \underline{\beta})\right) \times h^{(\nu/2)-1} \exp\left(-\frac{h\nu}{2s^{-2}}\right) \quad (2.4)$$

When we multiply the likelihood (2.3) by the prior (2.4), we can isolate the  $\beta$  in a factor of the form:

$$\exp\left(-\frac{1}{2}\left((\beta - \hat{\beta})'(h\mathbf{X}'\mathbf{X})(\beta - \hat{\beta}) + (\beta - \underline{\beta})'(h\mathbf{H})(\beta - \underline{\beta})\right)\right) \quad (2.5)$$

Again, using the results in Appendix C, we see that this means that

$$\beta|h \sim N\left(\bar{\beta}, (h\bar{\mathbf{H}})^{-1}\right)$$

where

$$\bar{\beta} = (\mathbf{X}'\mathbf{X} + \mathbf{H})^{-1}(\mathbf{X}'\mathbf{X}\hat{\beta} + \mathbf{H}\underline{\beta}) \text{ and } \bar{\mathbf{H}} = (\mathbf{X}'\mathbf{X} + \mathbf{H})$$

Note that  $\bar{\beta}$  doesn't depend upon  $h$ . It's a precision-weighted average of the data and prior. Because of the conjugate nature of the prior, noisier data means the data are less informative, but we also doubt the precision of our prior information in exactly the same proportion.

The conditional posterior for  $\beta$  is fairly simple. The posterior for  $h$  is a bit more complicated. First, as before, the multiplicative integrating constant for the Normal density for  $\beta$  will have the form

$$|h\bar{\mathbf{H}}|^{1/2} = h^{k/2} |\bar{\mathbf{H}}|^{1/2}$$

We can wash out the constant  $|\bar{\mathbf{H}}|^{1/2}$ , but will have to allow for the  $h^{k/2}$  factor. Second, (2.5), when expanded, will include an extra term inside the  $\exp(\dots)$  which includes  $h$ . From Appendix C, the simplest way to find this is to evaluate the exponent at  $\bar{\beta}$ . If we collect all the factors of  $h$  (allowing for the  $h^{k/2}$  needed for the Normal), we get

$$h^{N/2} h^{(\nu/2)-1} h^{k/2} h^{-k/2} = h^{(N+\nu)/2-1} \equiv h^{(\bar{\nu}/2)-1}$$

There are several equivalent ways to write the exponent for the gamma. (All have  $-h/2$  premultiplying them).

$$\hat{\varepsilon}'\hat{\varepsilon} + \nu s^2 + (\bar{\beta} - \hat{\beta})' \mathbf{X}'\mathbf{X} (\bar{\beta} - \hat{\beta}) + (\bar{\beta} - \underline{\beta})' \mathbf{H} (\bar{\beta} - \underline{\beta}) \quad (2.6a)$$

$$\bar{\varepsilon}'\bar{\varepsilon} + \nu s^2 + (\bar{\beta} - \underline{\beta})' \mathbf{H} (\bar{\beta} - \underline{\beta}) \quad (2.6b)$$

$$\hat{\varepsilon}'\hat{\varepsilon} + \nu s^2 + (\hat{\beta} - \underline{\beta})' \left( (\mathbf{X}'\mathbf{X})^{-1} + \mathbf{H}^{-1} \right)^{-1} (\hat{\beta} - \underline{\beta}) \quad (2.6c)$$

where  $\bar{\varepsilon} = y - \mathbf{X}\bar{\beta}$ . While these seem rather ugly, they actually make quite a bit of sense. Each term is a weighted estimate of the variance, weighted by the amount of information that goes into computing it. The data will estimate  $\sigma^2$  using the sum of squared errors divided by observations (in (2.6b)) or by degrees of freedom (in (2.6a) or (2.6c)). Multiplying by the divisor gives us back the sum of squared errors. The prior on  $h$  estimates  $\sigma^2$  with  $s^2$ . We weight that by the degrees of freedom of that prior.<sup>3</sup> And we have another “estimate” of  $\sigma^2$  coming out of the prior on  $\beta$ . With  $\beta|h \sim N(\underline{\beta}, (h\mathbf{H})^{-1})$ , and  $\mathbf{H}^{1/2} (\mathbf{H}^{1/2})' = \mathbf{H}$  (any such matrix)

$$\mathbf{H}^{1/2}(\beta - \underline{\beta}) \sim N(0, \sigma^2 I_k)$$

With the elements of  $\mathbf{H}^{1/2}(\beta - \underline{\beta})$  being independent  $N(0, \sigma^2)$ , their average square will be an estimate of  $\sigma^2$ . But, evaluated at  $\bar{\beta}$ , that's

$$\frac{1}{k} (\bar{\beta} - \underline{\beta})' \mathbf{H} (\bar{\beta} - \underline{\beta})$$

This is a  $k$  term estimate of  $\sigma^2$ , so we weight it by  $k$  to give the final term in (2.6b). The other quadratic terms can be interpreted similarly. In (2.6c),  $\hat{\beta} - \underline{\beta}$  is the difference between two independent estimators for  $\beta$ , so the variance of the difference is the sum of the variances, thus  $\sigma^2 ((\mathbf{X}'\mathbf{X})^{-1} + \mathbf{H}^{-1})$ , which leads to the third term as an estimator of  $k\sigma^2$ . While it's not the most convenient of the three forms for actual calculation, (2.6c) gets the “degrees of freedom” correct. The first has  $N - k$ , the second  $\nu$  and the last  $k$ , summing to the total  $\bar{\nu} = N + \nu$ . (2.6a and 2.6b don't sum independent estimates for  $\sigma^2$  since  $\underline{\beta}$  and  $\bar{\beta}$  aren't independent).

---

<sup>3</sup>Higher degrees of freedom means a tighter prior around  $s^2$ .

## 2.3 Using Cross Product Matrices

The Standard Normal Linear Model has the great advantage (compared to most non-linear models) that its likelihood has a set of *sufficient statistics*: by writing the likelihood as

$$p(y|h, \beta) \propto h^{N/2} \exp \left( -\frac{h}{2} (y'y - 2y'X\beta + \beta'X'X\beta) \right)$$

all the data information for computing this for any combination of  $\beta, h$  can be summarized in the four pieces of information:  $N, y'y, X'y, X'X$ . Or even better, this is just  $N$  and

$$[X|y]' [X|y] = \begin{bmatrix} X'X & X'y \\ y'X & y'y \end{bmatrix}$$

This can be computed using the RATS instruction **CMOMENT**. Although you can input the list of variables to this directly, it's generally easier to just start out by estimating the linear regression of interest.

```
linreg price
# constant lot_siz bed bath stor
cmom(lastreg)
```

The **CMOM(LASTREG)** instruction creates a  $(k+1) \times (k+1)$  SYMMETRIC matrix with the block structure above (explanatory variables first, followed by dependent variable). This matrix is called %CMOM by default.

To pull information out of the cross product matrix in a flexible fashion, you can use the integer variables %NREG (Number of REGressors) defined by the **LINREG** and %NCMOM (Number of CMOM variables) defined by **CMOM** to set the positions. %XSUBMAT (eXtract SUBMATrix) can then be used to get the needed submatrices. This is a function with the five arguments:

```
%XSUBMAT(A, startrow, endrow, startcol, endcol)
```

See *Tips and Tricks* (Section 2.6) for more on this function.

The formula for the posterior mean with the conjugate prior is

$$\bar{\beta} = (X'X + H)^{-1} (X'X\hat{\beta} + H\beta)$$

We can simplify this a bit using

$$X'X\hat{\beta} = (X'X) (X'X)^{-1} (X'y) = X'y$$

For doing the Bayesian analysis of this model, we never actually have to compute the original linear regression, or compute the inverse of  $X'X$ ;  $X'X$  itself provides the precision of the estimate coming from the data, and it's the precision, and not the variance, that we use in the calculations.

In addition to %XSUBMAT for extracting submatrices, there are several "convenience" functions which do very specific and useful calculations. The one

expression which is key for evaluating the likelihood is

$$y'y - 2y'X\beta + \beta'X'X\beta$$

That can be obtained from the cross product matrix by evaluating the quadratic form:

$$\begin{bmatrix} -\beta' & 1 \end{bmatrix} \begin{bmatrix} X'X & X'y \\ y'X & y'y \end{bmatrix} \begin{bmatrix} -\beta \\ 1 \end{bmatrix} \quad (2.7)$$

The function `%RSSCMOM` can do this directly. `%RSSCMOM(%CMOM, BETA)` evaluates (2.7) for the vector of coefficients `BETA`. You just need to provide the cross product matrix and the  $k$ -vector  $\beta$ . `%RSSCMOM` takes care of the rest of the calculation.

There's also `%RANMVPOSTCMOM` which computes the posterior distribution given a cross product matrix representing the data and a Normal prior. We'll use that a bit later.

Both of these functions have big brothers (`%SIGMACMOM` and `%RANMVKRONCMOM`) which can be used for linear systems estimation – more than one dependent variable, but with the same explanatory variables in each, such as a VAR.

## 2.4 Calculations

The example used in the Koop book is a hedonic price regression for housing prices. This works with an intercept plus four regressors from a list of possible attributes for the house.

```
open data hprice.prn
data(format=prn,org=columns) 1 546 price lot_siz bed $
  bath stor drive recroom bment gas aircond gar desireloc
linreg price
# constant lot_siz bed bath stor
```

We can summarize the data evidence with the cross product matrix computed using

```
cmom(lastreg)
```

The prior consists of

1. mean vector for the regression coefficients
2. (relative) precision matrix for those coefficients
3. mean for the precision  $h$  of the residuals
4. degrees of freedom for  $h$

While the use of precisions is much more convenient for calculations than the use of variances, they are not the natural form for prior information. It's, in fact, standard errors that are the most natural, since they're in the same units



as the coefficients (for the prior there) and the data (for the prior on the residual precision).

If you read the thought process Koop uses in coming up with the prior, one thing to note is that it's all about choosing a reasonable range for each of the coefficients. The covariance matrix for a five variable regression has 15 slots, but we're coming up with a prior which is zero on the off-diagonal. Now, larger lot sizes and more stories are generally going to be associated with more bedrooms and bathrooms, so we would expect that the data evidence will show a negative correlation among the coefficients.<sup>4</sup> So why don't we try to model that? Aside from being rather difficult to fill in the off-diagonal, it's also not necessary. The data provide the data evidence, and that correlation of coefficients is due to the particular sample that we're analyzing.

The prior shouldn't try to mimic the data; the more "orthogonal" to the data it is, the more useful it will be. That is, it should provide help in the areas where the data are weak. In many cases (perhaps most cases with linear regressions), that's with the sizes of the coefficients. Data with a fair bit of collinearity have a tough time sorting out the relative merits of each. If 2 bedrooms tend to go with 1 bath and 4 bedrooms with 2 baths, is it the extra baths or the extra bedrooms that are most important? Even highly unreasonable looking coefficient combinations (say with a negative value for bedrooms and a compensating high value for bathrooms) might fit the data almost equally well as more reasonable values. A prior which gives some help regarding the sizes can produce superior results.

Given the prior standard deviations, we can square to create the prior variances and invert to create the prior precision. Koop includes an extra step here which is often not done with this type of model, but should be. The prior variances on the coefficients are our thoughts about the *marginal* distribution. However, we are using a prior where the distribution conditional on  $h$  is multivariate Normal, but the marginal is multivariate  $t$ . The covariance matrix of the multivariate  $t$  has the same basic form as its underlying Normal, but it's scaled up by  $\frac{\nu}{\nu-2}$  to compensate for the behavior of the division by the gamma. So we start out with the information on the prior in natural form:

```
compute s2prior=5000.0^2
compute nuprior=5.0
compute [vector] bprior=||0.0,10.0,5000.0,10000.0,10000.0||
compute [symm]   vprior=$
               %diag(||10000.0^2,5.0^2,2500.0^2,5000.0^2,5000.0^2||)
```

then back out the H that corresponds to these settings:

```
compute [symm]   vpriort=vprior*(nuprior-2)/(s2prior*nuprior)
compute [symm]   hprior =inv(vpriort)
```

---

<sup>4</sup>Positive correlation among regressors tends to produce a negative correlation among their coefficients.

So our prior is summarized with the four values:

1. BPRIOR (VECTOR, prior mean)
2. HPRIOR (SYMMETRIC, relative precision of prior)
3. S2PRIOR ("mean"<sup>5</sup> for prior for  $\sigma^2$ ).
4. NUPRIOR (degrees of freedom on prior for  $\sigma^2$ ).

The following computes the summary of the posterior for  $\beta$ :

```
compute [symm] hdata=%xsubmat(%cmom,1,%nreg,1,%nreg)
compute [symm] vpost=inv(hdata+hprior)
compute [vect] bpost=vpost*$
      (%xsubmat(%cmom,1,%nreg,%nreg+1,%nreg+1)+hprior*bprior)
```

and this does the same for  $h$ .

```
compute rsspost =%rsscmom(%cmom,bpost)
compute rssprior=%qform(hprior,bpost-bprior)
compute nupost  =(%nobs+nuprior)
compute s2post  =(rsspost+rssprior+nuprior*s2prior)/nupost
```

This uses formula (2.6b) which is the simplest of the three to compute with RATS.

VPOST is  $\bar{H}^{-1}$ . The precision of  $\beta|h$  is  $h\bar{H}$ . The mean of  $h$  is  $\bar{s}^{-2}$ , so the scale matrix for the posterior is  $(\bar{s}^{-2}\bar{H})^{-1} = \bar{s}^2\bar{H}^{-1}$  which will be S2POST\*VPOST using the variables defined above. The marginal for  $\beta$  is a multivariate- $t$  with S2POST\*VPOST as the scale matrix and NUPOST degrees of freedom. To get the standard errors of the marginal for  $\beta$ , we'll again have to correct for the degrees of freedom of the  $t$ , multiplying up by NUPOST/(NUPOST-2). Note that, unlike the situation with the prior, NUPOST is fairly large and so the correction factor is near one. The following shows the posterior mean and standard deviation for the five coefficients:

```
disp "Posterior mean, standard dev with informative prior"
do i=1,%nreg
    disp bpost(i) sqrt(s2post*vpost(i,i)*nupost/(nupost-2.0))
end do i
```

The marginal for each coefficient is just a  $t$  (shifted in mean and scaled).  $P(\beta_j > 0|y)$  can be computed rather easily using the CDF of the  $t$ . The argument for this just has to be standardized to take into account the non-zero mean and scale. The standard  $t$  CDF is %TCDF(x,nu).

The HPDI's are most conveniently done using the %INVTTEST(alpha,nu) function, where alpha is the desired tail probability (thus 1-the desired coverage). – and + that number of the (uncorrected) standard errors gives the range.

---

<sup>5</sup>"mean" is in quotes because, technically, the prior is a gamma distribution for  $h$  (not for  $\sigma^2$ ) and the reciprocal of S2PRIOR is the mean for that. However, "reciprocal of the mean of the prior on the reciprocal" is a bit of a mouthful.

```

compute hpdi95=%invtttest(.05,nupost), $
      hpdi99=%invtttest(.01,nupost)
do i=1,%nreg
  compute sb=sqrt(s2post*vpost(i,i))
  disp %tcdf(bpost(i)/sb,nupost) $
      bpost(i)-hpdi95*sb bpost(i)+hpdi95*sb $
      bpost(i)-hpdi99*sb bpost(i)+hpdi99*sb
end do i

```

## 2.5 Simulations

This model has a posterior density from which draws can be made by conventional means. There are two ways of doing this:

1. if you only need  $\beta$  and not  $h$ , you can draw directly from the multivariate  $t$ .
2. If you need  $h$  (or  $\sigma^2$ ) as well, you can draw  $h$  from its gamma, and  $\beta$  from the conditional multivariate Normal. Note that you do not draw  $h$  from the gamma and then independently draw  $\beta$  from the multivariate  $t$ ; that would break the connection between the two sets of parameters.

To draw from a (correlated) multivariate Normal, you need a factor (any factor) of its covariance matrix, that is, a matrix  $F$  such that  $FF' = V$ . Unless you need a specific form of  $F$  for other reasons, the simplest and quickest factor is the Cholesky factor (also known as the Cholesky decomposition), which chooses  $F$  to be lower triangular. This is computed in RATS using the `%DECOMP(V)` function. The RATS function for drawing from a (mean zero) multivariate Normal with factor matrix  $F$  is `%RANMVNORMAL(F)`. Note that you input  $F$ , not  $V$ . If you input  $V$ , then RATS would have to take the factor, and would have to repeat that every time you needed a draw. Simulations can take quite a bit of time (depending upon the number of repetitions), so making the process efficient, particularly by simple adjustments, can be important. Here, we pull the calculation of the factor outside the loop. The more that we can do first (before starting to compute draws), the faster we'll execute.

The RATS function for drawing from a mean zero multivariate  $t$  whose underlying Normal has factor matrix  $F$  is `%RANMVT(F, nu)`.

The simulation here isn't all that interesting, since the posterior means and covariance fairly reasonably describe the posterior distribution. If, however, we were interested in some non-linear function of the  $\beta, h$ , this would be the way to do it.

Example 2.3 repeats the calculation of the posterior from Example 2.2. To calculate the mean and covariance matrix, we need to sum the draws, and their outer product ( $\beta\beta'$ ). `BSUM` has the sum of the draws and `BBSUM` the sum

of the outer products. These are converted into the mean and variance in the standard way. This does 10000 replications. You can change this by editing the first line.

```
compute ndraws=10000
compute fpost=%decomp(s2post*vpost)
compute [vector] bsum =%zeros(%nreg,1)
compute [symm]    bbsum=%zeros(%nreg,%nreg)
do reps=1,ndraws
  compute [vector] bdraw=bpost+%ranmvt(fpost)
  compute bsum =bsum+bdraw
  compute bbsum=bbsum+%outerxx(bdraw)
end do reps
*
compute drawmean=bsum/ndraws
compute drawvar =bbsum/ndraws-%outerxx(drawmean)
```

## 2.6 RATS Tips and Tricks

### Function Wizard

This chapter introduced quite a few functions, some of which have many parameters. If you don't remember how a particular function works, or if you recall that there was a function to do "X", but don't remember its name, the function wizard can help. You can use either the `F()` toolbar icon or the *Wizards—Functions* menu operation to bring this up. This has both an alphabetized list of *all* the functions available and a set of categories which reduce the list. Many of the functions in this section are in the *Bayesian Methods* category. Note, by the way, that functions can be in more than one category.

### In-line matrices

The `||...||` notation allows you to create matrices element by element. For instance, we're using:

```
compute [vector] bprior=||0.0,10.0,5000.0,10000.0,10000.0||
compute [symm]   vprior=%diag(||10000.0^2,5.0^2,2500.0^2,5000.0^2,5000.0^2||)
```

The first of these creates a 5-vector and saves it into `bprior`. The second creates a different 5-vector, and creates a diagonal matrix from it, saving that into `vprior`. Note from the second one that you can include expressions in the elements. This is one of the great advantages of using in-line matrices. The first of the two could have been done with

```
dec vector bprior(5)
input bprior
  0.0 10.0 5000.0 10000.0 10000.0
```

That has its uses, particularly when the vector or matrix gets quite large. However, **INPUT** can only accept numbers, not expressions. While the values for `vprior` are constants, they're much easier to read (and change) in the form above.

If you need multiple rows, you can use a single `|` to separate rows. For instance,

```
compute phimat=||phi,phi^2,phi^3,phi^4|$
                1.0, 0.0 , 0.0 , 0.0 |$
                0.0, 1.0 , 0.0 , 0.0 |$
                0.0, 0.0 , 1.0 , 0.0||
```

creates

$$\begin{bmatrix} \phi & \phi^2 & \phi^3 & \phi^4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

### Sub-matrix Functions

RATS includes several functions for pulling blocks of information out an an array. The most flexible of these is %XSUBMAT; there are simpler ones that you might also find to be useful.

<b>%XCOL (A, n)</b>	Returns column $n$ of $A$ .
<b>%XROW (A, n)</b>	Returns row $n$ of $A$ .
<b>%XDIAG (A)</b>	Returns the diagonal of a matrix as an vector.
<b>%XSUBVEC (A, i, j)</b>	Returns the sub-vector $A(i), \dots, A(j)$
<b>%XSUBMAT (A, i, j, k, l)</b>	Returns the sub-matrix of $A$ from $(i,k)$ to $(j,l)$ , that is, $i$ to $j$ is the row range and $k$ to $l$ is the column range.

If you need the number of rows or columns for the upper bounds on %XSUBVEC or %XSUBMAT, use %ROWS (A) and %COLS (A). For instance, to get a sub-vector from position 2 on, you would use %XSUBVEC (A, 2, %ROWS (A)).

### CMOM and the %XSUBMAT function

The **CMOM(LASTREG)** instruction creates a  $(k+1) \times (k+1)$  SYMMETRIC matrix with the explanatory variables first, followed by dependent variable. This matrix is called %CMOM by default. You can use the MATRIX option to pick a different target. For instance, this will put the information into the matrix XYYX.

```
cmom(lastreg,matrix=xyyx)
```

%NREG is the number of regressors, and %NCMOM is the size of the cross-product matrix, which here will be one more than %NREG. So we can pull the needed components out of %CMOM using:

```
X'X      %XSUBMAT (%CMOM, 1, %NREG, 1, %NREG)
X'y      %XSUBMAT (%CMOM, 1, %NREG, %NREG+1, %NREG+1)
y'y(scalar) %CMOM (%NREG+1, %NREG+1) or %CMOM (%NCMOM, %NCMOM).
```

### The REPORT instruction

We didn't pretty up the output from Example 2.2, using only **DISPLAY** in a loop. This is certainly fine to start, and, of course, the first thing to do always is to make sure that you get the calculations done right. But you want to get used to using the **REPORT** instruction to organize output. That will be done with most examples from now on. Let's show what we can do with one of the output blocks from 2.2.

```

disp "Posterior mean, standard deviation with informative prior"
do i=1,%nreg
    disp bpost(i) sqrt(s2post*vpost(i,i)*nupost/(nupost-2.0))
end do i

```

This produces:

Posterior mean, standard deviation with informative prior	
-4035.05276	3530.16095
5.43162	0.36625
2886.81217	1184.92501
16965.23537	1708.02297
7641.23418	997.01655

When we're first writing the program, that's enough. Our numbers match Koop's (Table 3.1, final column), which is what we need. However, it's not very useful otherwise. For one thing, you'll never report all those digits. It also has no column or row labels.

To build a report, you start with `REPORT(ACTION=DEFINE)` and end with `REPORT(ACTION=SHOW)`. In between, you add content and apply formatting. In this case, let's start by producing the same table, but with row and column labels. The long first line is done using the option `SPAN`, so we start with

```

report(action=define)
report(atrow=1,atcol=1,span) $
    "Posterior mean, standard deviation with informative prior"
report(atrow=2,atcol=2) "Coeff" "StdErr"

```

This leaves the 2,1 cell (above the labels) blank. We will now loop over the regressors (just like in the `DISPLAY` loop). If we want to include the variable names to label the rows, we can do that using the `%EQNREGLABELS` function. `%EQNREGLABELS(0)(i)` is the regression label used in the RATS output for the  $i^{th}$  coefficient.<sup>6</sup> The content can be inserted with:<sup>7</sup>

```

do i=1,%nreg
    report(row=new,atcol=1) %eqnreglabels(0)(i) $
        bpost(i) sqrt(s2post*vpost(i,i)*nupost/(nupost-2.0))
end do i

```

To show the numbers at two decimal places, use

```
report(action=format,picture="*.##")
```

The report is now finished, and we can do

```
report(action=show)
```

to display the results:

---

<sup>6</sup>The "0" in this case means the most recent regression. This function can also be used with other equations that you create, for instance, with `%eqnreglabels(myeqn)(i)`.

<sup>7</sup>There's no `ACTION` option on `REPORT` when you're adding content.

Posterior mean, standard deviation with informative prior		
	Coeff	StdErr
Constant	-4035.05	3530.16
LOT_SIZ	5.43	0.37
BED	2886.81	1184.93
BATH	16965.24	1708.02
STOR	7641.23	997.02

You can also export this directly to a text file or spreadsheet with something like:

```
report(action=show,format=xls)
```

There are quite a few formats that you can choose. Note that if you use `FORMAT=XLS` (or `FORMAT=WKS`), the numbers are exported at full precision. The specific format is used only for displaying or exporting to a text file.

Finally, you can also bring the report up in a separate window using the *Restore Report* operation on the *Window* menu. RATS keeps a queue of the most recently generated reports, whether done directly by the user using **REPORT** or by other instructions. From the window, you can copy information out to paste into another program, or you can export to another file with the *Export* operation on the *File* menu.



**Example 2.1 Linear Model: Single Variable**

```

compute n=50
compute h=1.0,b=2.0
*
* Generate x, e, and y
*
set x 1 n = %ran(1.0)
set e 1 n = %ran(h^-.5)
set y 1 n = b*x+e
*
scatter(footer="Artificial Data")
# x y
*
linreg y
# x
*
compute bprior      =1.5
compute vprior      =.25
compute hprior      =1.0/vprior
compute nuprior     =10.0
compute ssqprior    =1.0/1.0
*
compute bdata       =%beta(1)
compute vdata       =%xx(1,1)
compute hdata       =1.0/vdata
compute nudata      =%ndf
compute ssqdata     =%seesq
*
compute hpost       =hdata+hprior
compute vpost       =1.0/hpost
compute bpost       =(hdata*bdata+hprior*bprior)/hpost
compute nupost      =%nobs+nuprior
compute ssqpost     =(ssqdata*nudata+ssqprior*nuprior+$
    (bdata-bpost)^2*hdata+(bprior-bpost)^2*hprior)/nupost
*
@GridSeries(from=1.0,to=3.0,size=.02,pts=gpts) bgraph
*
set dataf 1 gpts = $
    exp(%logtdensitystd(ssqdata*vdata,bgraph-bdata,nudata))
set priorf 1 gpts = $
    exp(%logtdensitystd(ssqprior*vprior,bgraph-bprior,nuprior))
set postf 1 gpts = $
    exp(%logtdensitystd(ssqpost*vpost,bgraph-bpost,nupost))
scatter(footer="Marginal Prior and Posterior for Beta",style=lines,$
    vmin=0.0,key=upleft,klabels=|"Prior","Posterior","Likelihood"|) 3
# bgraph priorf
# bgraph postf
# bgraph dataf

```

## Example 2.2 Multiple Regression: Conjugate Prior

```

open data hprice.prn
data(format=prn,org=columns) 1 546 price lot_siz bed bath stor drive $
  recroom bment gas aircond gar desireloc
*
* This is the equation we're analyzing
*
linreg(vcv) price
# constant lot_siz bed bath stor
*
* Prior for variance. This is most conveniently kept in direct form, not
* precision.
*
compute s2prior=5000.0^2
compute nuprior=5.0
*
* Prior mean and variance
*
compute [vector] bprior=||0.0,10.0,5000.0,10000.0,10000.0||
compute [symm]   vprior=%diag(||10000.0^2,5.0^2,2500.0^2,5000.0^2,5000.0^2||)
*
* Back out the precision for the Normal-gamma prior by inverting the
* solution for V-bar.
*
compute [symm]   vpriori=vprior*(nuprior-2)/(s2prior*nuprior)
compute [symm]   hprior =inv(vpriori)
*
* Compute the cross product matrix from the regression. This will
* include the dependent variable as the final row.
*
cmom(lastreg)
*
* Compute the posterior precision (up to the scaling by the residual
* precision) and the posterior mean. The top %nreg x %nreg corner of
* %cmom is the X'X, and the first %nreg elements of the %nreg+1 column
* is X'y.
*
compute [symm] hdata=%xsubmat(%cmom,1,%nreg,1,%nreg)
compute [symm] vpost=inv(hdata+hprior)
compute [vect] bpost=vpost*$
  (%xsubmat(%cmom,1,%nreg,%nreg+1,%nreg+1)+hprior*bprior)
*
* Compute the posterior scale for the variance
*
compute rsspost =%rsscmom(%cmom,bpost)
compute rssprior=%qform(hprior,bpost-bprior)
compute nupost  =(%nobs+nuprior)
compute s2post  =(rsspost+rssprior+nuprior*s2prior)/nupost
*
* This is a quick and dirty display of the information on the posterior
* (from table 3.1). The marginal posterior for beta is t with <<nupost>>
* degrees of freedom, so we need to correct for the variance.
*

```

```

disp "Posterior mean, standard deviation with informative prior"
do i=1,%nreg
    disp bpost(i) sqrt(s2post*vpost(i,i)*nupost/(nupost-2.0))
end do i
*
* For computing HPDI's, it's simplest to use the %invtttest function. To
* get .95 coverage, use an argument of .05 (%invtttest and similar
* "inverse test" functions are based upon tail probabilities)
*
disp
disp "Posterior probability of beta>0. HPDI's"
compute hpdi95=%invtttest(.05,nupost),hpdi99=%invtttest(.01,nupost)
do i=1,%nreg
    compute sb=sqrt(s2post*vpost(i,i))
    disp %tcdf(bpost(i)/sb,nupost)  bpost(i)-hpdi95*sb  bpost(i)+hpdi95*sb $
                                     bpost(i)-hpdi99*sb  bpost(i)+hpdi99*sb
end do i

```

### Example 2.3 Multiple Regression with Conjugate Prior: Simulations

```

open data hprice.prn
data(format=prn,org=columns) 1 546 price lot_siz bed bath stor $
    drive recroom bment gas aircond gar desireloc
*
linreg(vcv) price
# constant lot_siz bed bath stor
*
compute s2prior=5000.0^2
compute nuprior=5.0
compute [vector] bprior=||0.0,10.0,5000.0,10000.0,10000.0||
compute [symm]   vprior=%diag(||10000.0^2,5.0^2,2500.0^2,5000.0^2,5000.0^2||)
compute [symm]   vprior=vprior*(nuprior-2)/(s2prior*nuprior)
compute [symm]   hprior =inv(vprior)
*
cmom(lastreg)
*
compute [symm] hdata=%xsubmat(%cmom,1,%nreg,1,%nreg)
compute [symm] vpost=inv(hdata+hprior)
compute [vect] bpost=vpost*$
    (%xsubmat(%cmom,1,%nreg,%nreg+1,%nreg+1)+hprior*bprior)
*
compute rsspost =%rsscmom(%cmom,bpost)
compute rssprior=%qform(hprior,bpost-bprior)
compute nupost  =(%nobs+nuprior)
compute s2post  =(rsspost+rssprior+nuprior*s2prior)/nupost
*****
*
* Draws directly from posterior for beta
*
compute nreps=10000
compute fpost=%decomp(s2post*vpost)
dec vect tdraw(%nreg)
compute [vector] bsum =%zeros(%nreg,1)
compute [symm]   bbsum=%zeros(%nreg,%nreg)
do reps=1,nreps
    compute [vector] bdraw=bpost+fpost*(tdraw=%rant(nupost))
    compute bsum =bsum+bdraw
    compute bbsum=bbsum+%outerxx(bdraw)
end do reps
*
compute [vector] drawmean=bsum/nreps
compute drawvar =bbsum/nreps-%outerxx(drawmean)
*
report(action=define)
report(atrow=1,atcol=1,span) "Simulations from Posterior"
report(atrow=2,atcol=1,span) "10000 repetitions"
report(atrow=3,atcol=1,fillby=rows) drawmean
report(atrow=4,atcol=1,fillby=rows) %sqrt(%xdiag(drawvar))
report(action=format,picture="*.##")
report(action=show)

```

```

*
* Same simulation done with the two-step (draw h, draw beta|h). Note
* that the FPOST is computed with the unscaled version of VPOST. This is
* because we'll get the scale put in when we divide by the draw for h.
*
compute nreps=10000
compute fpost=%decomp(vpost)
dec vect tdraw(%nreg)
compute [vector] bsum=%zeros(%nreg,1)
compute [symm]  bbsum=%zeros(%nreg,%nreg)
do reps=1,nreps
    compute hdraw=%ranchisqr(nupost)/(nupost*s2post)
    compute [vector] bdraw=bpost+%ranmvnormal(fpost)/sqrt(hdraw)
    compute bsum =bsum+bdraw
    compute bbsum=bbsum+%outerxx(bdraw)
end do reps
*
compute drawmean=bsum/nreps
compute drawvar =bbsum/nreps-%outerxx(drawmean)
*
report(action=define)
report(atrow=1,atcol=1,span) "Simulations from Posterior"
report(atrow=2,atcol=1,span) "10000 repetitions from (beta,h)"
report(atrow=3,atcol=1,fillby=rows) drawmean
report(atrow=4,atcol=1,fillby=rows) %sqrt(%xdiag(drawvar))
report(action=format,picture="*.##")
report(action=show)

```

## Normal Linear Model with Independent Prior

### 3.1 Theory

This is the same model as before, with likelihood

$$p(y|h, \beta) \propto h^{N/2} \exp \left( -\frac{1}{2} (\beta - \hat{\beta})' (h\mathbf{X}'\mathbf{X}) (\beta - \hat{\beta}) \right) \times \exp \left( -\frac{h}{2} \hat{\varepsilon}'\hat{\varepsilon} \right) \quad (3.1)$$

Now, however, instead of the conjugate prior

$$\beta|h \sim N(\underline{\beta}, (h\mathbf{H})^{-1}), h \sim G(\nu, s^{-2})$$

which has the somewhat unrealistic feature that our prior gets noisier if the data do (because of the presence of  $h$  in the prior precision matrix), we make the priors on  $\beta$  and  $h$  independent. Thus we have:

$$\beta \sim N(\underline{\beta}, \mathbf{H}^{-1}), h \sim G(\nu, s^{-2})$$

or

$$p(\beta, h) \propto \exp \left( -\frac{1}{2} (\beta - \underline{\beta})' \mathbf{H} (\beta - \underline{\beta}) \right) \times h^{(\nu/2)-1} \exp \left( -\frac{h\nu}{2s^{-2}} \right) \quad (3.2)$$

As before, we multiply the likelihood (3.1) by the prior (3.2), and isolate the  $\beta$  in a factor of the form:

$$\exp \left( -\frac{1}{2} \left( (\beta - \hat{\beta})' (h\mathbf{X}'\mathbf{X}) (\beta - \hat{\beta}) + (\beta - \underline{\beta})' \mathbf{H} (\beta - \underline{\beta}) \right) \right)$$

And again, as before, we can read from this:

$$\beta|h \sim N(\bar{\beta}, \bar{\mathbf{H}}^{-1})$$

where we define

$$\bar{\beta} = (h\mathbf{X}'\mathbf{X} + \mathbf{H})^{-1} (h\mathbf{X}'\mathbf{X}\hat{\beta} + \mathbf{H}\underline{\beta}), \quad \bar{\mathbf{H}} = (h\mathbf{X}'\mathbf{X} + \mathbf{H})$$

However,  $h$  now doesn't cancel out of the  $\bar{\beta}$ . If  $h$  is small (data are noisy), we downweight the data and move more towards the prior; conversely, if  $h$  is high, we trust the data more than our prior.

The “problem” now comes when we try to get the posterior distribution for  $h$ . With the conjugate prior, the precision of the posterior for  $\beta$  took the form  $h\bar{\mathbf{H}}$ . Since that depends upon  $h$ , we had to allow for that in the constant of

integration for the  $\beta|h$  density. However, in that case, the determinant of  $h\bar{\mathbf{H}}$  depends upon  $h$  in a very simple way. Now, however, we need an integrating constant of

$$|h\mathbf{X}'\mathbf{X} + \mathbf{H}|^{1/2}$$

If we multiply and divide by that factor and rearrange terms, we get the posterior (proportional to):

$$|h\mathbf{X}'\mathbf{X} + \mathbf{H}|^{1/2} \exp\left(-\frac{1}{2}(\beta - \bar{\beta})'(h\mathbf{X}'\mathbf{X} + \mathbf{H})(\beta - \bar{\beta})\right) \times \\ |h\mathbf{X}'\mathbf{X} + \mathbf{H}|^{-1/2} h^{N/2} h^{(\nu/2)-1} \exp\left(-\frac{h}{2}(\bar{\varepsilon}'\bar{\varepsilon} + \nu s^2) + (\bar{\beta} - \beta)' \mathbf{H}(\bar{\beta} - \beta)\right)$$

Because  $\bar{\beta}$  (and thus  $\bar{\varepsilon}'\bar{\varepsilon}$ ) now depend upon  $h$  (in a highly non-linear way) the terms inside the bottom  $\exp(\dots)$  factors are very unpleasant functions of  $h$ . The kernel of the unconditional distribution for  $h$  is

$$|h\mathbf{X}'\mathbf{X} + \mathbf{H}|^{-1/2} h^{N/2} h^{(\nu/2)-1} \exp\left(-\frac{h}{2}(\bar{\varepsilon}'\bar{\varepsilon} + \nu s^2) + (\bar{\beta} - \beta)' \mathbf{H}(\bar{\beta} - \beta)\right)$$

Because of that determinant term and the non-linearity in the  $\exp$  term, this is not a known density.

A simpler way to approximate the joint density is by means of Gibbs sampling—the simplest form of Markov Chain Monte Carlo (MCMC). The MCMC techniques, combined with sufficient computing power, were really what brought Bayesian methods into standard practice. They made it possible to work with posterior densities which couldn't be handled analytically (which means almost all realistic ones).

Gibbs sampling 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.

The idea behind Gibbs sampling is that we partition our set of parameters into two or more groups. Here the two groups are  $h$  and  $\beta$ . Now, we already figured out how to draw  $\beta|h$ . The problem was that we couldn't get an unconditional draw for  $h$ . Suppose, however, that we look at the posterior for  $h$  conditional on  $\beta$ . Eliminating factors in the prior that don't depend upon  $h$  gives us the much simpler:

$$p(h) \propto h^{(\nu/2)-1} \exp\left(-\frac{h\nu}{2s^2}\right) \quad (3.3)$$

For our purposes now, the original form of the likelihood is easier:

$$p(y|h, \beta) \propto h^{N/2} \exp\left(-\frac{h}{2}(y'y - 2y'\mathbf{X}\beta + \beta'\mathbf{X}'\mathbf{X}\beta)\right) \quad (3.4)$$

The posterior for  $h|\beta$  can be obtained by multiplying (3.3) and (3.4) and rearranging to get

$$p(h|\beta, y) \propto h^{(N+\nu)/2-1} \exp\left(-\frac{h}{2} (\nu s^2 + \varepsilon(\beta)' \varepsilon(\beta))\right)$$

where we're defining  $\varepsilon(\beta) = y - \mathbf{X}\beta$ .

This is a gamma with degrees of freedom  $N + \nu$  and (reciprocal) mean

$$\frac{\nu s^2 + \varepsilon(\beta)' \varepsilon(\beta)}{N + \nu}$$

Thus we have well-understood posteriors for  $\beta|h$  and  $h|\beta$ . Unfortunately, you can't just combine those two to get the joint distribution. We would either need  $p(\beta, h) = p(\beta|h)p(h)$  or  $p(\beta, h) = p(h|\beta)p(\beta)$ . But neither of the marginal densities  $p(\beta)$  or  $p(h)$  is a known type.

With Gibbs sampling, we don't need that. The insight is that if you draw  $\beta|h$  and then  $h|\beta$ , the pair  $\beta, h$  should be closer to their unconditional distribution than before. Repeat it often enough and it should converge to give draws from the joint distribution. This turns out to be true in most situations.

Now there may be quite a bit of calculation to get just one pair of  $\beta, h$ . Fortunately, if the level of correlation in the chain of draws isn't extremely high, you can just take the statistics across a single run of draws. 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  $p(\beta, h)$ , you (of necessity) have a draw from the marginals  $p(\beta)$  and  $p(h)$ , so now the conditional draw procedure will give you a new draw from the desired distribution. They're just not *independent* draws. With enough "forgetfulness", however, the sample averages will converge.

This is a case where Gibbs sampling generally works quite well, and you can get reasonable results without a huge number of draws. Gibbs sampling works best when parameters which are (strongly) correlated with each other are in the same block. In this case, we would be doing that, since we do all the regression coefficients together.  $h$  and  $\beta$  are only loosely connected. If we get an outlying draw for  $\beta$  (one with a high sum of squared errors), that will tend to produce a smaller draw for  $h$  in the next round, but even with a small value of  $h$  (high variance), the distribution for  $\beta$  is still centered around the mean, so another outlying draw isn't that likely. If, instead, you have a sampler which tries to do two highly correlated parameters separately, it will be hard for the pair to move away from one region since each will be tied to the last value for the other.



## 3.2 Calculations

The calculations aren't all that different from the simulations in the case of the conjugate priors with the two-step draws (for  $h$  and then  $\beta|h$ ). We just have to recompute the mean for the gamma for  $h$  at each step (which can be done easily using the `%RSSCMOM` function to compute  $\varepsilon(\beta)' \varepsilon(\beta)$ ). However, because the draws are correlated, it's usually a good idea to keep the entire record, in order to do diagnostics. In the simulations of the conjugate prior, we just kept the sum of the draw and its outer product. Here, we keep the entire record.

With all the MCMC estimators, we'll use the following basic structure for the simulation loops:

```
compute nburn =# of burn-in
compute ndraws=# of keepers
do draw=-nburn,ndraws
  *
  * Do the draws here
  *
  if draw<=0
    next
  *
  * Do the bookkeeping here.
  *
end do draw
```

This will actually do one more burn-in draw than the value of `NBURN`, not that that matters. It's just much easier when the positive values of `DRAW` are the keepers.

The “bookkeeping” is whatever we need to do with the draws. To keep the whole record, it's easier to create series, with `VECTOR` elements for the regression coefficients and just standard real-valued series for the precision estimates. You need to prepare the series in advance, so we do the following:

Outside the loop:

```
dec series[vect] bgibbs
dec series      hgibbs
gset bgibbs 1 ndraws = %zeros(%nreg,1)
set  hgibbs 1 ndraws = 0.0
```

and inside the loop:

```
compute bgibbs(draw)=bdraw
compute hgibbs(draw)=hdraw
```

Because the RATS function `%RANGAMMA` uses the “other” parameterization (shape and scale rather than degrees of freedom and mean), the simplest way to draw the required  $h$  is to use `%RANCHISQR`. `%RANCHISQR(nu)` draws from a gamma

with `nu` degrees of freedom and mean `nu`. To get a gamma draw with `nu` degrees of freedom and mean `mu`, we need `%RANCHISQR(nu)*mu/nu`. But since the desired `mu` is the reciprocal of

$$\frac{\nu s^2 + \varepsilon(\beta)' \varepsilon(\beta)}{N + \nu}$$

the multiplier becomes

$$\frac{N + \nu}{\nu s^2 + \varepsilon(\beta)' \varepsilon(\beta)} \times \frac{1}{N + \nu} = \frac{1}{\nu s^2 + \varepsilon(\beta)' \varepsilon(\beta)} \quad (3.5)$$

The code for getting `hdraw` (conditioned on the current value of `bdraw`) is thus:

```
compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
compute hdraw  =%ranchisqr(nuprior+%nobs)/rssplus
```

where `rssplus` computes the denominator in the last expression in (3.5).

We've seen in the earlier example how to compute the posterior mean and variance for the regression coefficients and how to draw from that distribution. However, in the earlier example, the variance matrix for the Normal was the same in each draw except for scaling by the draw for the overall variance; the Normal distribution had the same basic shape, and we could make the calculation more efficient by doing much of the work outside the loop. Here, however, each different value of  $h$  produces a completely different covariance matrix. There's nothing to be gained by keeping the mean and variance, since they'll have to be computed new the next time.

When you just need a "one-off" draw from a Normal posterior with Normal prior, you can use the specialized RATS function

```
%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
```

This takes as input

1. the cross product matrix for the data
2. the current draw for the residual
3. the precision matrix for the prior
4. the mean of the prior

and computes the posterior mean and variance, and makes a random draw from that distribution. We will use this function quite a bit, since in almost any type of MCMC, the draws are this type of one-off.

After we're done with the draws, we need to process the information in the `BGIBBS` and `HGIBBS` series. We could have just done the summing inside the loop as before. However, since we're saving the whole record anyway, it makes more sense to keep the loop as clean as possible and do the calculations later:

```

dec symm vpost(%nreg,%nreg)
dec vect bpost(%nreg)
do i=1,%nreg
  sstats(mean) 1 ndraws bgibbs(t)(i)>>bpost(i)
end do i
compute vpost=%zeros(%nreg,%nreg)
do t=1,ndraws
  compute vpost=vpost+%outerxx(bgibbs(t)-bpost)
end do t
compute vpost=vpost/ndraws

```

`bpost` and `vpost` will be the MCMC estimates of the mean and variance of the coefficients.

### 3.3 Diagnostics

Because the MCMC draws are correlated, we need to check that we are getting adequate “forgetfulness”. The sampling variance for the sample mean from a sequence with autocorrelation of unknown form can be computed using the long-run variance  $S(0)$  so widely used in econometrics (for instance, in the Newey-West covariance matrix). It’s computed most conveniently in RATS using the **MCOV** instruction.<sup>1</sup> To compute the diagnostic measure on a single set of coefficient draws, we need to get the deviations from their mean, hence:

```
set u 1 ndraws = bgibbs(t)(i)-bpost(i)
```

Remember that you need to run this **SET** instruction across the whole record of draws (1 to `ndraws`). `bgibbs(t)` is the **VECTOR** of draws at entry `t`, so we need entry `i` of that when analyzing coefficient `i`. We apply **MCOV** to that series with the constant as the only input. We use the Bartlett (Newey-West) window with (to be conservative) many lags.

```
mcov(lags=fix(.04*ndraws),lwindow=bartlett,matrix=t_s0) 1 ndraws u
# constant
```

**MCOV** produces a sum (not the mean), so it needs to be divided by the *square* of `ndraws` to get an estimate of the variance of the sample mean. `t_s0` is (a  $1 \times 1$ ) matrix which is why we need the  $(1, 1)$  subscripts.

```
compute nse=sqrt(t_s0(1,1)/ndraws^2)
```

The Geweke CD measure does this calculation separately for early and late subsamples of the record. To get the first 10% and final 40%, the following computes the lower and upper entry limits:

```
compute l1=1,u1=fix(.1*ndraws)
compute u2=ndraws,l2=u2-fix(.4*ndraws)+1
```

---

<sup>1</sup>See this chapter’s *Tips and Tricks* (Section 3.6) for more on **MCOV**.

The calculations of the sample mean and the numerical variance (nse squared) are done separately for each of the subsamples:

```
sstats(mean) 11 u1 bgibbs(t)(i)>>b1
set u 11 u1 = bgibbs(t)(i)-b1
mcov(lags=fix(.04*ndraws),lwindow=bartlett,matrix=t_s0) 11 u1 u
# constant
compute nv1=t_s0(1,1)/%nobs^2
sstats(mean) 12 u2 bgibbs(t)(i)>>b2
set u 12 u2 = bgibbs(t)(i)-b2
mcov(lags=fix(.04*ndraws),lwindow=bartlett,matrix=t_s0) 12 u2 u
# constant
compute nv2=t_s0(1,1)/%nobs^2
```

The CD measure is just the standard  $z$ -score for testing a difference between means from independent samples.<sup>2</sup>

```
compute cd=(b1-b2)/sqrt(nv1+nv2)
```

As you can see from the example program, there is actually more code *after* the end of the simulation loop than there is inside it. And the post-processing is that type of analysis that we're likely to see many times, in different applications. This suggests that a procedure to do these calculations would be helpful. That's what @MCMCPOSTPROC does.

```
@mcmcpstproc(ndraws=ndraws,mean=bmean,$
              stderrs=bstderrs,cd=bcd,nse=bnse) bgibbs
```

The one parameter here is the SERIES[VECT] that keeps the full set of draws for one of the vectors. You have to provide the ndraws value. The others are options:

MEAN=VECTOR *of sample means*

STDERRS=VECTOR *of sample standard errors*

CV=SYMMETRIC *estimated covariance matrix*

NSE=VECTOR *of numerical standard errors of each component*

CD=VECTOR *of CD measures*

Note that NSE and (especially) CD can take some time to compute if you use many draws (say 25000). This is not uncommon. Certain post-processing calculations use calculations that go up more than linearly with the number of

---

<sup>2</sup>Koop has an error in 4.14 – the denominator should be the square root of the sum of the squares of the standard errors.

draws. You can speed this up by reducing the number of lags to a smaller multiplier than `.04*ndraws` on the `MCOV`'s. The procedure actually cuts this off at 400, regardless of the number of draws.

### 3.4 The Bayesian Approach to Hypothesis Testing

The Bayesian approach to deciding between hypotheses (or models)  $H_1$  and  $H_2$  given data  $Y$  is to formulate likelihood functions  $p_1(Y|\theta_1)$  and  $p_2(Y|\theta_2)$ , with priors  $p_1(\theta_1)$  and  $p_2(\theta_2)$  over the respective parameter spaces. The two hypotheses themselves have prior probabilities  $p(H_1)$  and  $p(H_2)$ . Let  $\ell_1$  be the loss in choosing  $H_2$  if  $H_1$  is true and  $\ell_2$  be the loss in choosing  $H_1$  if  $H_2$  is true. The expected loss in choosing  $H_1$  will then be  $\ell_2 p(H_2|Y)$  and in choosing  $H_2$  will be  $\ell_1 p(H_1|Y)$ . By Bayes Rule,

$$p(H_1|Y) = p(Y|H_1)p(H_1)/p(Y)$$

(interpreted as densities if needed) and similarly for  $H_2$ . Since  $p(Y)$  is independent of the hypotheses, the decision comes down to: choose  $H_1$  if

$$\ell_1 p(H_1)p(Y|H_1) > \ell_2 p(H_2)p(Y|H_2)$$

equivalently

$$\frac{p(Y|H_1)}{p(Y|H_2)} > \frac{\ell_2 p(H_2)}{\ell_1 p(H_1)} \quad (3.6)$$

The left side of this is known as the Bayes factor, which summarizes the data evidence regarding the two hypotheses.  $p(Y|H_1)$  is the marginal likelihood of  $H_1$ . It's the rather ugly denominator with the constants of integration that we try to ignore when doing inference about  $\theta_1$ . It's of little value when we are analyzing a single model; it comes into play only when, as here, we have competing models.

The most important thing to note about this is how different it is from conventional hypothesis testing. The hypotheses don't have to be nested. There are no null and alternative hypotheses. We don't evaluate the sampling distribution under just one (the null); we treat the two hypotheses symmetrically. If you want to "favor" one over the other, you can do that by adjusting the loss function or the prior probabilities. Note that two people, even if they agree on the models and the priors for the parameters, might disagree on the decision if the values on the right side of (3.6) differ. As a result, it's very common for the output of the Bayesian hypothesis test to be the Bayes factor or a similar summary of the relative merits of the two hypotheses. That way, readers can make their own decisions.

The most common form of hypothesis test in classical statistics is the likelihood ratio test. Just as above, this starts with the likelihood functions  $p_1(Y|\theta_1)$  and  $p_2(Y|\theta_2)$ . The data evidence regarding the two is summarized by the likelihood

ratio:

$$\frac{\max_{\theta_1} p(Y|\theta_1)}{\max_{\theta_2} p(Y|\theta_2)}$$

The Bayes factor is similar, but uses the integral of the likelihood over the parameters (weighted by the prior), rather than the maximum value as the “index of support”.

$$\frac{\int p(Y|\theta_1)p(\theta_1)d\theta_1}{\int p(Y|\theta_2)p(\theta_2)d\theta_2}$$

Note that this can be quite sensitive to the choice of the prior. Suppose, for instance, that we compare two priors:  $U(-a, a)$  and  $U(-2a, 2a)$ . If  $p(Y|\theta)$  is effectively zero outside  $[-a, a]$ , our inference regarding  $\theta$  (in isolation) will be the same for the two priors, since the shape of the priors is identical (that is, flat) in the area where the likelihood is high. The marginal likelihood, however, will be half the value with the wider prior, since we’re averaging in values that the data have told us are unlikely.

One thing to note, however, is that with most data sets  $p(Y|\theta)$  will eventually (with enough data), look like a spike at the maximizer<sup>3</sup> This behavior is the source of the Schwarz Bayesian Criterion (SBC, BIC, SIC) which computes a large-sample version of the (log) Bayes factor using only the maximum of the log likelihood.

The marginal likelihood

$$\int p(Y|\theta)p(\theta)d\theta$$

is only computable analytically in a few situations. One of those is the Normal-gamma conjugate prior, which we’ll examine more closely in the next section.

### 3.5 Hypothesis Testing with the Linear Model

With the linear model, with conjugate Normal-gamma priors, we get a marginal likelihood of the form:

$$\pi^{-N/2} \frac{\Gamma(\bar{\nu}/2) |\bar{\mathbf{V}}|^{1/2} (\bar{\nu}\bar{s}^2)^{-\bar{\nu}/2}}{\Gamma(\nu/2) |\mathbf{V}|^{1/2} (\nu s^2)^{-\nu/2}}$$

This is Koop, 3.34, with some rearranging. As with most calculations like this, it’s generally a good idea to compute this in log form. Ignoring the  $\pi$  factor

---

<sup>3</sup>Remember that  $p(Y|\theta)$  is not in log form: it’s a product of  $N$  terms, so even if a value of  $\theta$  gives likelihood elements that are quite close to the maximum (say 98% per factor), at  $N = 200$ , it will be  $.98^N \approx .02$  relative to the maximum.

(which will be the same regardless of model), the log of this can be computed with

```
compute logml=$
%lngamma(.5*nupost) -.5*nupost *log(nupost *s2post) + $
.5*log(%det(vpost))- $
%lngamma(.5*nuprior)+.5*nuprior*log(nuprior*s2prior)- $
.5*log(%det(vprior))
```

While it's possible to rearrange the regression and the prior and redo the calculations for the restricted model, I'll show you a sneaky way to do this using a high-tech calculation rather than manual recoding. This uses the exact inverse formula for matrices with “infinite” components (see Appendix D). In RATS, if you have a matrix pair  $\{A, B\}$ , you can use the procedure

```
@EXACTINVERSE A B C D
```

to get the exact inverse of  $A + \kappa B$  as  $\kappa \rightarrow \infty$ . For the case of testing the submodel which eliminates one parameter, you use an input mean of the zero vector with infinite precision on the diagonal element being knocked out.

For testing coefficient  $i$ , this does the following: the mean and precision of the restriction space are:

```
compute hrest=%outerxx(%unitv(%nreg,i))
compute brest=%zeros(%nreg,1)
```

We then adjust the prior mean (to meet the restriction, trivial here, but not with more general restrictions) and get the (log) determinant of the finite part of the precision. The negative of this will be the log det of the restricted prior variance. (@EXACTINVERSE computes that log det as a side effect).

```
@exactinverse hprior hrest vpost vinf
compute logdetprior=-%logdet
compute [vect] bpriorr=vpost*hprior*bprior+vinf*hrest*brest
```

Compute the posterior mean with the restriction imposed. Again, get -the log det of the finite part of the precision.

```
@exactinverse hdata+hprior hrest vpost vinf
compute [vect] bpost=vpost*
(%xsubmat(%cmom,1,%nreg,%nreg+1,%nreg+1)+hprior*bprior)+$
vinf*hrest*brest
compute logdetpost=-%logdet
```

This is all the same (since we've kept the same form for the regression):

```
compute rsspost =%rsscmom(%cmom,bpost)
compute rssprior=%qform(hprior,bpost-bpriorr)
compute nupost =(%nobs+nuprior)
compute s2post =(rsspost+rssprior+nuprior*s2prior)/nupost
```

The marginal likelihood uses the `logdetpost` and `logdetprior` that we got from doing the exact inverse:

```
compute logmlx = $
  %lngamma(.5*nupost) -.5*nupost *log(nupost *s2post) +.5*logdetpost-$
  %lngamma(.5*nuprior)+.5*nuprior*log(nuprior*s2prior)-.5*logdetprior
```

The Bayes factor (for the restricted versus the unrestricted model) is then  $\exp(\text{logmlx}-\text{logml})$ .



### 3.6 RATS Tips and Tricks

#### The instruction **MCOV**

If you ever have a need to compute a long-run variance, or do a calculation with a Newey–West or Bartlett window, **MCOV** is your instruction. It does a wide variety of calculations which are designed to be robust against serial correlation, heteroscedasticity or specification error. In general, it computes

$$\sum_{l=-L}^L \sum_t w_l (\mathbf{Z}'_t u_t u_{t-l} \mathbf{Z}_{t-l})$$

where the  $w_l$  are a set of weights. In the simplest case where  $\mathbf{Z}_t \equiv 1$ , this will give you  $(T \times)$  the long-run variance of  $u$ .

$L$  is non-zero when you're dealing with serial correlation, and 0 when you're allowing just for heteroscedasticity or specification error. Note that this produces the sum (not the mean), so you need to take that into account if you need a variance. Results using such matrices often are stated in the form:

$$\sqrt{T} (\theta - \hat{\theta}) \rightarrow N(0, \mathbf{A}^{-1} \mathbf{B} \mathbf{A}^{-1})$$

To get a proper theorem, the  $\mathbf{A}$  and  $\mathbf{B}$  matrices need to be  $O(1)$ , so the  $\mathbf{B}$  (which is the matrix in this that would come from **MCOV**) is the sum divided by  $T$ . However, the way this formal statement gets used is

$$\hat{\theta} \approx N\left(\hat{\theta}, \frac{1}{T} \mathbf{A}^{-1} \mathbf{B} \mathbf{A}^{-1}\right) = N\left(\hat{\theta}, (T\mathbf{A})^{-1} (T\mathbf{B}) (T\mathbf{A})^{-1}\right)$$

so if we keep both of the  $\mathbf{A}$  and  $\mathbf{B}$  in their sum forms, we get the correct scale on the variance. In the use of **MCOV** in this chapter, the  $\mathbf{A}$  is trivially 1, so we need to divide the sum by  $T^2$  to get the proper estimate of the variance.

#### White space: Making Programs More Readable

Well-chosen spaces and line breaks can make it easier to read a program, and will go a long way towards helping you get your calculations correct. At a minimum, you should get into the habit of indenting loops and the like. This makes it much easier to follow the flow of the program, and also makes it easier to skip more easily from one part of the calculation to the next.

Two operations on the *Edit* menu can be helpful with this. *Indent Lines* adds (one level) of indentation at the left; *Unindent Lines* removes one level. The number of spaces per level is set in the preferences in the *Editor* tab. All the programs in this book are done with 3 space indenting, which seems to work well for the way that RATS is structured. In the following, if you select the four lines in the body of the loop and do *Edit–Indent*

```

do reps=1,nreps
compute hdraw=%ranchisqr(nupost)/(nupost*s2post)
compute [vector] bdraw=bpost+%ranmvnormal(fpost)/sqrt(hdraw)
compute bsum =bsum+bdraw
compute bbsum=bbsum+%outerxx(bdraw)
end do reps

```

you'll get

```

do reps=1,nreps
  compute hdraw=%ranchisqr(nupost)/(nupost*s2post)
  compute [vector] bdraw=bpost+%ranmvnormal(fpost)/sqrt(hdraw)
  compute bsum =bsum+bdraw
  compute bbsum=bbsum+%outerxx(bdraw)
end do reps

```

When you have to break a line, it's usually best to keep the pieces of the calculation as intact as possible. If there's any parallel structure, putting those onto separate lines is also helpful. For instance, we had a very long expression for the mean of the posterior:

```

compute [vect] bpost=vpost*$
  (%xsubmat(%cmom,1,%nreg,%nreg+1,%nreg+1)+hprior*bprior)

```

Consider the difference in readability if we just ran lines out as far as we could:

```

compute [vect] bpost=vpost*(%xsubmat(%cmom,1,%nreg,%nreg+1,$
  %nreg+1)+hprior*bprior)

```

In the second line, we're breaking up the argument list for `%xsubmat`, and also (unnecessarily) breaking up the parenthetical expression.

Another example of a good use of line breaks and spacing is:

```

compute logmlx  = $
  %lngamma(.5*nupost) -.5*nupost*log(nupost*s2post) +.5*logdetpost-$
  %lngamma(.5*nuprior)+.5*nuprior*log(nuprior*s2prior)-.5*logdetprior

```

The two lines are parallel constructions, except the signs are opposite. By adding the few extra spaces to get things to line up, an error will be noticeable almost immediately.

### Example 3.1 Linear Model with Independent Prior

```

open data hprice.prn
data(format=prn,org=columns) 1 546 price lot_siz bed bath stor drive $
  recroom bment gas aircond gar desireloc
*
* This is the equation we're analyzing
*
linreg price
# constant lot_siz bed bath stor
*
* Prior for variance.
*
compute s2prior=5000.0^2
compute nuprior=5.0
*
* Prior mean and variance. Because the variance of the prior is now
* independent of the variance of the residuals, the <<hprior>> can be
* taken by just inverting the <<vprior>> without the degrees of freedom
* adjustment.
*
compute [vector] bprior=||0.0,10.0,5000.0,10000.0,10000.0||
compute [symm]   vprior=%diag(||10000.0^2,5.0^2,2500.0^2,5000.0^2,5000.0^2||)
compute [symm]   hprior =inv(vprior)
*
* Compute the cross product matrix from the regression. This will
* include the dependent variable as the final row. The cross product
* matrix has all the sufficient statistics for the likelihood (except
* the number of observations, which is %nobs).
*
cmom(lastreg)
*
* The two obvious places to start the Gibbs sampler are the OLS
* estimates and the prior.
*
compute bdraw =%beta
compute s2draw=%seesq
*
* We'll use the following for all MCMC estimators. This ends up taking
* nburn+1 burn-ins.
*
compute nburn =1000
compute ndraws=10000
*
dec series[vect] bgibbs
dec series      hgibbs
gset bgibbs 1 ndraws = %zeros(%nreg,1)
set  hgibbs 1 ndraws = 0.0
*
do draw=-nburn,ndraws
  *
  * Draw residual precision conditional on previous beta
  *
  compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)

```

```

compute hdraw =%ranchisqr(nuprior+%nobs)/rssplus
*
*   Draw betas given hdraw
*
compute bdraw =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
if draw<=0
  next
*
*   Do the bookkeeping here.
*
compute bgibbs(draw)=bdraw
compute hgibbs(draw)=hdraw
end do draw
*
* Compute mean and covariance matrix of draws for coefficients
*
dec symm vpost(%nreg,%nreg)
dec vect bpost(%nreg)
do i=1,%nreg
  sstats(mean) 1 ndraws bgibbs(t)(i)>>bpost(i)
end do i
compute vpost=%zeros(%nreg,%nreg)
do t=1,ndraws
  compute vpost=vpost+%outerxx(bgibbs(t)-bpost)
end do t
compute vpost=vpost/ndraws
*
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" $
  "Std Error" "NSE" "CD"
do i=1,%nreg
  set u 1 ndraws = bgibbs(t)(i)-bpost(i)
  *
  * Full sample statistics
  *
  mcov(lags=fix(.04*ndraws),lwindow=bartlett,matrix=t_s0) 1 ndraws u
  # constant
  compute nse=sqrt(t_s0(1,1)/ndraws^2)
  *
  * Geweke CD measure. Redo calculation of the mean and numerical
  * variance over the first 10% and last 40% of sample
  *
  compute l1=1,u1=fix(.1*ndraws)
  compute u2=ndraws,l2=u2-fix(.4*ndraws)+1
  *
  * Early sample
  *
  sstats(mean) l1 u1 bgibbs(t)(i)>>b1
  set u l1 u1 = bgibbs(t)(i)-b1
  mcov(lags=fix(.04*ndraws),lwindow=bartlett,matrix=t_s0) l1 u1 u
  # constant
  compute nv1=t_s0(1,1)/%nobs^2
  *
  * Late sample

```

```

*
sstats(mean) 12 u2 bgibbs(t)(i)>>b2
set u 12 u2 = bgibbs(t)(i)-b2
mcov(lags=fix(.04*ndraws),lwindow=bartlett,matrix=t_s0) 12 u2 u
# constant
compute nv2=t_s0(1,1)/%nobs^2
*
compute cd=(b1-b2)/sqrt(nv1+nv2)
*
report(row=new,atcol=1) %eqnreglabels(0)(i) bpost(i) $
      sqrt(vpost(i,i)) nse cd
end do i
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.##")
report(action=show)
*
* Same done with MCMCPostProc
*
@mcmcpstproc(ndraws=ndraws,mean=bmean,$
      stderrs=bstderrs,cd=bcd,nse=bnse) bgibbs
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" $
      "Std Error" "NSE" "CD"
do i=1,%nreg
      report(row=new,atcol=1) %eqnreglabels(0)(i) bmean(i) $
            bstderrs(i) bnse(i) bcd(i)
end do i
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.##")
report(action=show)

```

### Example 3.2 Linear Regression: Conjugate Prior with Restrictions

```

open data hprice.prn
data(format=prn,org=columns) 1 546 price lot_siz bed bath stor drive $
      recroom bment gas aircond gar desireloc
*
linreg(vcv) price
# constant lot_siz bed bath stor
*
compute s2prior=5000.0^2
compute nuprior=5.0
*
* Prior mean and variance
*
compute [vector] bprior=||0.0,10.0,5000.0,10000.0,10000.0||
compute [symm]   vprior=%diag(||10000.0^2,5.0^2,2500.0^2,5000.0^2,5000.0^2||)
*
compute [symm]   vprior=vprior*(nuprior-2)/(s2prior*nuprior)
compute [symm]   hprior =inv(vprior)
*
cmom(lastreg)

```

```

*
compute [symm] hdata=%xsubmat(%cmom,1,%nreg,1,%nreg)
compute [symm] vpost=inv(hdata+hprior)
compute [vect] bpost=vpost*$
    (%xsubmat(%cmom,1,%nreg,%nreg+1,%nreg+1)+hprior*bprior)
*
compute rsspost=%rsscmom(%cmom,bpost)
compute rssprior=%qform(hprior,bpost-bprior)
compute nupost=(%nobs+nuprior)
compute s2post=(rsspost+rssprior+nuprior*s2prior)/nupost
*****
* Through this point, this is the same as a previous example
*****
*
* Compute the (log) marginal likelihood for the unrestricted
* model(ignoring the pi term, which is the same for all models). Note
* that this needs to use the <<vprior>> matrix.
*
compute logml=$
    %lngamma(.5*nupost) -.5*nupost*log(nupost*s2post) +.5*log(%det(vpost))-
    %lngamma(.5*nuprior)+.5*nuprior*log(nuprior*s2prior)-.5*log(%det(vprior))
*
* Impose restrictions that beta(i)=0, one at a time
*
report(action=define)
report(atrow=1,atcol=1,span) "Posterior Odds for beta(i)=0"
do i=1,%nreg
    compute hrest=%outerxx(%unitv(%nreg,i))
    compute brest=%zeros(%nreg,1)
    *
    * Adjust the prior mean (to meet the restriction) and get the (log)
    * determinant of the finite part of the precision. -this will be the
    * log det of the restricted prior variance.
    *
    @exactinverse hprior hrest vpost vinf
    compute logdetprior=-%logdet
    compute [vect] bpriorr=vpost*hprior*bprior+vinf*hrest*brest
    *
    * Compute the posterior mean with the restriction imposed. Again, get
    * -the log det of the finite part of the precision.
    *
    @exactinverse hdata+hprior hrest vpost vinf
    compute [vect] bpost=vpost*(%xsubmat(%cmom,1,%nreg,%nreg+1,%nreg+1)+
        hprior*bprior)+vinf*hrest*brest
    compute logdetpost=-%logdet
    *
    * Compute the posterior scale for the variance
    *
    compute rsspost=%rsscmom(%cmom,bpost)
    compute rssprior=%qform(hprior,bpost-bpriorr)
    compute nupost=(%nobs+nuprior)
    compute s2post=(rsspost+rssprior+nuprior*s2prior)/nupost
    *
    * Compute the log marginal likelihood for the restricted model

```

```

*
compute logmlx =$
  %lngamma(.5*nupost) -.5*nupost *log(nupost *s2post) +.5*logdetpost-$
  %lngamma(.5*nuprior)+.5*nuprior*log(nuprior*s2prior)-.5*logdetprior
  report (atrow=i+1,atcol=1) %eqnreglabels(0)(i) exp(logmlx-logml)
end do i
report (action=format,width=20)
report (action=show)
*****
*
* Restriction that b3=b4 (additional bedroom adds the same as additional
* bathroom)
*
compute r=||0.0,0.0,1.0,-1.0,0.0||
compute hrest=%outerxx(r)
compute brest=%zeros(%nreg,1)
*
@exactinverse hprior hrest vpost vinf
compute logdetprior=-%logdet
compute [vect] bpriorr=vpost*hprior*bprior+vinf*hrest*brest
*
* Because we started with the same prior as before, which had prior
* means of 5000 on b3 and 10000 on b4, with variance on b4 four times
* that on b3, the adjusted prior has a mean of 6000 on each. (Closer to
* b3 of the tighter unrestricted prior on it).
*
disp "Restricted Prior Mean"
disp bpriorr
*
@exactinverse hdata+hprior hrest vpost vinf
compute [vect] bpost=vpost*(%xsubmat(%cmom,1,%nreg,%nreg+1,%nreg+1)+$
  hprior*bprior)+vinf*hrest*brest
compute logdetpost=-%logdet
disp "Restricted Posterior Mean"
disp bpost
*
* Compute the posterior scale for the variance
*
compute rsspost =%rsscmom(%cmom,bpost)
compute rssprior=%qform(hprior,bpost-bpriorr)
compute nupost =(%nobs+nuprior)
compute s2post =(rsspost+rssprior+nuprior*s2prior)/nupost
*
* Compute the log marginal likelihood for the restricted model
*
compute logmlx =$
  %lngamma(.5*nupost) -.5*nupost *log(nupost *s2post) +.5*logdetpost-$
  %lngamma(.5*nuprior)+.5*nuprior*log(nuprior*s2prior)-.5*logdetprior
disp "Posterior Odds on b3=b4"
disp exp(logmlx-logml)

```

## Nonlinear Regression: Introduction to Metropolis-Hastings

### 4.1 Theory

As mentioned earlier, the linear model with Normal errors has two very convenient (and closely related) properties:

1. It has a set of sufficient statistics
2. The likelihood can be inverted easily to form a kernel for the density of the parameters.

Because it has that set of sufficient statistics (the crossproduct matrix), you can do even millions of simulations in a very modest amount of time, because each simulation involves a set of calculations with matrices of modest size (number of regressors). There aren't any calculations the size of the data set in the inner loop.

In general, a non-linear regression model has neither of these nice properties. If

$$y_i = f(X_i, \gamma) + \varepsilon_i, \varepsilon_i \sim N(0, h^{-1}) \text{ i.i.d.}$$

the log likelihood is

$$-\frac{N}{2} \log 2\pi + \frac{N}{2} \log h - \frac{h}{2} \sum_{i=1}^N (y_i - f(X_i, \gamma))^2$$

While that last term has a  $\mathbf{Y}'\mathbf{Y}$  component, that won't help much. Now, there is a class of special cases which at least has sufficient statistics. These are *non-linear in parameters* (NLIP) models. This is short for "linear in the data, non-linear in the parameters". For these models,

$$f(X_i, \gamma) = X_i g(\gamma)$$

The final term in the log-likelihood for this is now:

$$-\frac{h}{2} \sum_{i=1}^N (y_i - X_i g(\gamma))^2 = -\frac{h}{2} (\mathbf{Y}'\mathbf{Y} - 2\mathbf{Y}'\mathbf{X}g(\gamma) + g(\gamma)' \mathbf{X}'\mathbf{X}g(\gamma))$$

so we have the same set of sufficient statistics as before. This can be a big time saver if it's available.



The lack of sufficient statistics mainly affects the time required to do a certain number of simulations. The lack of an easily identifiable kernel for the posterior is a more serious problem. We were able to do Gibbs sampling in the linear model with independent Normal-gamma prior because we had that. Assume that we again have an independent Normal-gamma prior.

$$p(\gamma, h) \propto \exp\left(-\frac{1}{2}(\gamma - \underline{\gamma})' \mathbf{H}(\gamma - \underline{\gamma})\right) \times h^{(v/2)-1} \exp\left(-\frac{h\nu}{2s^{-2}}\right)$$

The conditional posterior for  $h|\gamma$  is exactly the same as before:

$$p(h|\gamma, y) \propto h^{(N+v)/2-1} \exp\left(-\frac{h}{2}(\nu s^2 + \varepsilon(\gamma)' \varepsilon(\gamma))\right)$$

where

$$\varepsilon(\gamma) = y - f(X, \gamma)$$

The difference is that, unless we have an NLIP model, we'll have to run a loop over observations to compute  $\varepsilon(\gamma)' \varepsilon(\gamma)$ .

It's  $\gamma|h$  that causes the problems. Getting rid of factors that don't depend upon  $\gamma$ , we get

$$p(\gamma|h, y) \propto \exp\left(-\frac{1}{2}(\gamma - \underline{\gamma})' \mathbf{H}(\gamma - \underline{\gamma})\right) \exp\left(-\frac{h}{2}\varepsilon(\gamma)' \varepsilon(\gamma)\right) \quad (4.1)$$

While this can be computed easily enough (again, requiring a loop over observations to evaluate the second factor), it can't be written in the form of a known density.

Unlike the Gibbs sampler in the linear model, anything we do here is going to require at least some ingenuity or at least some experimentation. The most straightforward way to handle this is to use a more advanced form of MCMC called Metropolis-Hastings (or more correctly here, Metropolis within Gibbs, since we're using this for just part of a Gibbs sampler, with standard methods taking care of draws for  $h|\gamma$ ).

The idea behind this is as follows: suppose that our current draw for  $\gamma$  is called  $\gamma^{(i-1)}$ . We want to generate a new draw  $\gamma^{(i)}$  from  $p(\gamma|h, y)$ , but there's no known procedure for drawing from that. Instead, let's choose  $\gamma^*$  from a more convenient density  $q(\gamma)$  (often called the *proposal density*). We can evaluate both the density  $p(\gamma^*|h, y)$  and the density  $q(\gamma^*)$ . Compute

$$\alpha = \frac{p(\gamma^*|h, y)}{p(\gamma^{(i-1)}|h, y)} \times \frac{q(\gamma^{(i-1)})}{q(\gamma^*)} \quad (4.2)$$

With probability  $\alpha$ , we accept the new draw and make  $\gamma^{(i)} = \gamma^*$ , otherwise we stay with our previous value and make  $\gamma^{(i)} = \gamma^{(i-1)}$ . Note that it's possible to have  $\alpha > 1$ , in which case, we just accept the new draw.

The first ratio in (4.2) 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

$$\alpha = \frac{p(\gamma^*|h, y)/q(\gamma^*)}{p(\gamma^{(i-1)}|h, y)/q(\gamma^{(i-1)})}$$

$p/q$  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 ( $p$ ) 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  $p$  is low and  $q$  is high, we might as well stay put; we may revisit that one at a later time. Note that  $1/\alpha$  is the probability of moving back if we took  $\gamma^*$  and drew  $\gamma^{(i-1)}$  as a candidate.

This is known as *Independence Chain* M-H, where the candidates are drawn independently of the previous value. In greater generality, suppose that proposal density is dependent upon the previous value  $\gamma^{(i-1)}$ :  $q(\gamma^*|\gamma^{(i-1)})$ . In that case  $\alpha$  takes the form:

$$\alpha = \frac{p(\gamma^*|h, y)/q(\gamma^*|\gamma^{(i-1)})}{p(\gamma^{(i-1)}|h, y)/q(\gamma^{(i-1)}|\gamma^*)}$$

Note that if it's easier to go from  $\gamma^{(i-1)} \rightarrow \gamma^*$  than vice versa, (top  $q$  bigger than bottom  $q$ ), we'll tend to stay put. A special case of this is *Random Walk* M-H, where the proposal density is centered at  $\gamma^{(i-1)}$ . When  $q(\gamma^*|\gamma^{(i-1)}) = q(\gamma^{(i-1)}|\gamma^*)$ , which will be the case for multivariate Normal or  $t$  with a covariance matrix that's independent of  $\gamma$ , the formula for  $\alpha$  simplifies to

$$\alpha = \frac{p(\gamma^*|h, y)}{p(\gamma^{(i-1)}|h, y)}$$

We can just as easily move from one to the other, so we don't need the counterweighting for the draw density.

A few things to note about this:

1. We don't need  $p$  and  $q$  to be complete densities. Because they always are used in ratios, any common constants of integration will drop out.
2. Keep the calculations in logs as long as possible. In high dimensional problems, it's possible for the pieces to over- and underflow.

In implementing a chain, we need to decide upon the method and the proposal density. It's often easier to pick a reasonable proposal for Independence Chain M-H. With that, we're trying to pick from a density which comes fairly close to mimicking the shape of the posterior itself. If we have a respectable amount of data, we can use the asymptotic theory of non-linear regression to get an approximate density for  $\gamma$  as

$$\gamma \sim N \left( \hat{\gamma}, \hat{s}^2 \left( \sum_{i=1}^N \frac{\partial f(X_i, \gamma)'}{\partial \gamma} \frac{\partial f(X_i, \gamma)}{\partial \gamma} \right)^{-1} \right) \quad (4.3)$$

which would be the output from a conventional non-linear least squares optimization. If the prior is fairly flat, we can get by with that. If not, we can combine this with the posterior using the Normal prior-Normal data to get an approximate Normal posterior. If the prior might very well dominate the data for some parameters, you may need to maximize the posterior (4.1) directly (you would maximize the log of this) and take a Normal approximation from that optimization.

One of these Normal approximations might work just fine, but it's often a good idea to "fatten" up the tails a bit. If we look at the desirability measure:

$$p(\gamma^*|h, y)/q(\gamma^*)$$

it can be quite high even for candidates with a fairly low value of  $p$ , if  $q$  is relatively lower than that. Normals have fairly thin tails, so if we manage to stumble across a tail  $\gamma$  where the  $p$  function is fatter than the Normal, we can stay at that one spot for quite a while. Having overly thick tails in the proposal density isn't as big a problem because, if the tails of  $q$  are too fat, there will be places to which we won't move easily, but we won't get stuck anywhere.

However you do it, it's important to monitor how frequently you accept new draws. If that's low (below 20%), you need a better proposal density. With Independence Chain M-H, it's hard for an acceptance ratio to be too *high*. After all, if you hit the posterior density spot on ( $q = p$ ),  $\alpha = 1$  for all draws.

Independence Chain is generally the better procedure if the posterior is fairly well-behaved, in the sense that it has a single maximum and generally falls off from there in all directions. If the posterior is multi-modal, or has directions in which it falls off very slowly, it won't work as well, since a single (convenient) density will have a hard time matching those features. That's where Random Walk M-H becomes useful. Picking the proposal density there, however, is quite a bit more challenging. You don't want to be drawing from a distribution based upon the overall spread of the posterior. If you do that, then you won't really have a chance to explore those odd shapes in the posterior. Once you locate one, if you use too wide a spread, the next draw will likely be rejected (since you're in a small region with a high posterior); if it's accepted, you'll probably be moving somewhere else, perhaps with a small probability of going back. There's quite a bit more tuning involved with this, since a low acceptance rate can mean that you need to spread your draw more (you've found one mode and can't move off of it because your draws don't span to the other one), or can mean that you need to spread your draw less. With Random Walk M-H, a high acceptance rate is also not good. It probably means you have too small a spread and are not moving very far. Somewhere in the range of .25 to .50 is generally best.

The same matrix as generated above in (4.3) can be useful to give some idea of the relative scales of the parameters. Taking the diagonal elements only (and possibly scaling down a bit) isn't a bad starting place.

## 4.2 Calculations

Our model is

$$y_i = f(X_i, \gamma) + \varepsilon_i, \varepsilon_i \sim N(0, h^{-1}) \text{ i.i.d.}$$

For purposes of drawing from the posterior for  $\gamma$ , it's most convenient to have  $\gamma$  represented as a `VECTOR`. For writing the  $f$  function, however, it's usually most convenient to use descriptive variables (`alpha`, `sigma`, etc.). That makes it easier to change the representation, by adding or deleting parameters. Fortunately, it's relatively easy to switch between the two.

The example used is of a CES production function:

$$y_i = \gamma_1 + (\gamma_2 x_{1i}^{\gamma_4} + \gamma_3 x_{2i}^{\gamma_4})^{1/\gamma_4} + \varepsilon_i$$

We'll set up the function using the descriptive variable names `gamma1`, etc.

```
nonlin(parmset=cesparms) gamma1 gamma2 gamma3 gamma4
*
compute gamma4=1.0
compute gamma1=0.0
compute gamma2=gamma3=0.5
*
frml ces y = gamma1+(gamma2*x1^gamma4+gamma3*x2^gamma4)^(1/gamma4)
nlls(frml=ces, parmset=cesparms)
```

To estimate this model successfully with **NLLS**, we need a positive guess value for  $\gamma_4$  (otherwise the function can't even be computed) and for  $\gamma_2$  or  $\gamma_3$  (otherwise,  $\gamma_4$  drops out of the function). Note that we used a named `PARMSET`. This is important so we can set the values of all the parameters from a `VECTOR`.

The setup for the prior is the same (other than numbers) as it is for the linear model:

```
compute s2prior=1.0/10.0
compute nuprior=12.0
*
compute [vector] bprior=||1.0,1.0,1.0,1.0||
compute [symm] vprior=.25*%identity(4)
compute [symm] hprior =inv(vprior)
```

We'll again start at the least-squares estimates and set up a `SERIES[VECT]` and `SERIES` to collect the draws. In this example, we're doing 25000 draws with 5000 burn-ins. We have a new addition to this: we also need to keep a count of the number of acceptances.

```

compute bdraw=%beta
compute s2draw=%seesq
compute nburn=5000
compute ndraws=25000
dec series[vect] bgibbs
dec series      hgibbs
gset bgibbs 1 ndraws = %zeros(%nreg,1)
set  hgibbs 1 ndraws = 0.0
compute accept=0

```

Example 4.1 does Random Walk M-H. This is using a Normal with mean zero and covariance matrix taken from **NLLS** as the increment in the proposal density. Since the covariance matrix is fixed across draws, we take its Cholesky factorization outside the loop. (Given the level of effort in the other calculations, this doesn't matter that much, but it's a simple enough optimization).

```

compute fxx=%decomp(%seesq*%xx)

```

With the set of options used in **NLLS**, **%XX** does not have the scale factor for the residual variance incorporated; hence the scaling by **%SEESQ**. If you use **ROBUSTERRORS** on **NLLS**, you would skip that, because the relationship with the residuals is built into the calculation.

The draw for the precision of the residuals is pretty much the same as with the linear model, other than the need to calculate the sum of squared residuals. Note the use of the **%PARMSPOKE** function. This is why we assigned a named **PARMSET** to the parameters. See this chapter's *Tips and Tricks* (Section 4.3) for more on this function.

When you use a **FRML** like **CES** in an expression, it does not include the dependent variable – it computes the right-hand side of  $y = f(X, \gamma)$ , not the residuals.

```

compute %parmspoke(cesparms,bdraw)
sstats / (y-ces(t))^2>>rssbeta
compute rssplus=nuprior*s2prior+rssbeta
compute hdraw  =%ranchisqr(nuprior+%nobs)/rssplus

```

We now draw a candidate based upon the previous value of **bdraw** and evaluate the sum of squared residuals at that.

```

compute btest=bdraw+%ranmvnormal(fxx)
compute %parmspoke(cesparms,btest)
sstats / (y-ces(t))^2>>rsstest

```

We next compute the (logs) of the posterior densities and exp up to get the acceptance probability.

```

compute logptest=-.5*hdraw*rsstest-.5*qform(hprior,btest-bprior)
compute logplast=-.5*hdraw*rssbeta-.5*qform(hprior,bdraw-bprior)
compute alpha   =exp(logptest-logplast)

```

This is the code for the randomized acceptance. If we accept, we replace `bdraw` with the test vector and increment `accept`.

```
if %ranflip(alpha)
    compute bdraw=btest, accept=accept+1
```

The remainder of the loop is the same as with the linear model, as is the post-processing for finding the mean and variance of the posterior. The one addition is that we need to display the fraction of acceptances:

```
disp "Acceptance Rate" accept/(ndraws+nburn+1.0)
```

Independence Chain M-H has the additional requirement that we compute the (kernel) of the proposal density. Fortunately, RATS is set up to handle that automatically. If you arrange the calculation so the draw is done in a single function call, you can fetch the log of the kernel of that previous draw using the function `%RANLOGKERNEL()`.

Since `%RANLOGKERNEL()` only has the correct value until the next set of draws, we need to keep track of it in case we hang onto the previous draw. If your initializer for the Gibbs sampler is the mean of your proposal density (from a Normal or  $t$ ), you can start with the log kernel of 0.

The setup is the same until we get to:

```
compute fxx      =%decomp(%seesq*%xx)
compute nuxx     =10
compute logqlast=0.0
```

We're fattening the tails by using a  $t$  with 10 degrees of freedom. `logqlast` will be used to hold the  $q$  value for the last draw; we'll use `logqtest` for the  $q$  value for the test draw.

Most of the internals of the draw loop are the same. `compute btest` is different because it is centered at `%beta` rather than `bdraw`; we add the `compute logqtest` to grab the log kernel while it's available; `alpha` is computed with the more complicated expression and, if we accept, we hang onto the value of  $\log q$ .

```
compute btest=%beta+%ranmvt(fxx, nuxx)
compute logqtest=%ranlogkernel()
compute %parmspoke(cesparms, btest)
sstats / (y-ces(t))^2>>rsstest
compute logptest=-.5*hdrow*rsstest-.5*qform(hprior, btest-bprior)
compute logplast=-.5*hdrow*rssbeta-.5*qform(hprior, bdraw-bprior)
compute alpha =exp(logptest-logplast+logqlast-logqtest)
if %ranflip(alpha) {
    compute bdraw=btest, accept=accept+1
    compute logqlast=logqtest
}
```

### 4.3 RATS Tips and Tricks

#### Using PARMSETS

A **PARMSET** is an object which organizes a set of parameters for non-linear estimation. In most cases where you do non-linear optimization (with instructions like **NLLS** or **MAXIMIZE**), you don't need a "named" parameter set. The **NONLIN** instruction without a **PARMSET** option creates the default unnamed parameter set which is what will be used on the non-linear estimation instruction, if *it* doesn't have a **PARMSET** option:

```
nonlin gamma1 gamma2 gamma3 gamma4
nlls(frml=ces)
```

The named **PARMSET** is useful when you need to be able to work with the parameters as a **VECTOR**. The functions for this are **%PARMSPEEK(parmset)** and **%PARMSPOKE(parmset, vector)**. The first of these returns the current settings as a **VECTOR**; the second (which we used in this chapter), resets the parameters based upon the value in the **VECTOR**.

```
nonlin(parmset=cesparms) gamma1 gamma2 gamma3 gamma4
compute %parmspoke(cesparms, ||0.0, 0.5, 0.5, 1.0||)
compute gamma4=0.7
compute v=%parmspeek(cesparms)
```

Here, the **%PARMSPOKE** will make  $\gamma_1 = 0, \gamma_2 = .5$ , etc. At the end, **v** will be **[0, .5, .5, .7]**.

**PARMSETS** are also helpful if you would like to break a larger parameter set up into more manageable pieces. The following defines one **PARMSET** for the parameters governing the mean and another for the **GARCH** parameters. Those are "added" (combined) on the **MAXIMIZE** instruction to form the full parameter set for the estimation:

```
linreg us3mo
# constant us3mo{1}
frml(lastreg, vector=ar1) meanf
nonlin(parmset=meanparms) ar1
*
nonlin(parmset=garchparms) gamma alpha beta lambda
maximize(parmset=meanparms+garchparms) logl 2 *
```

**The instruction FIND**

In Example 4.2, we do independence M-H using the coefficients and a (fattened up) asymptotic covariance matrix from an **NLLS** to provide the mean and covariance matrix for the proposal density. That works reasonably well since the prior is fairly loose; as a result, the posterior density (which is what we're trying to match) is dominated by the data.

Suppose instead that we have a more informative prior. The **NLLS** information might no longer be a good match for the posterior. What we can do instead is to find the *posterior mode*: the coefficient vector which maximizes the (log) posterior density. We can use that, plus the Hessian from that optimization (possibly with some fattening) to provide a proposal density which might better match the posterior. To compute the log posterior density, start by using the “evaluation” method for the instruction (see appendix A). For **NLLS**, this is `METHOD=EVALUATE`. This will give you the log likelihood. Add to that the log density for the prior. If the prior is in several independent pieces, you'll have to add each of those in. This code fragment estimates the posterior mode for the model used in this chapter. Note that you need to use the `METHOD=BFGS` and `STDERRS` options on the **FIND** if you want to get the Hessian. (The default estimation method for **FIND** is `METHOD=SIMPLEX`, which is derivative-free).

```
compute [vector] bprior=||1.0,1.0,1.0,0.5||
compute [symm]   vprior=.09*%identity(4)
compute [symm]   hprior =inv(vprior)
*
declare real logpostdensity
find(parmset=cesparms,method=bfgs,stderrs) max logpostdensity
    nlls(frml=ces,parmset=cesparms,method=evaluate,noprint)
    compute logpostdensity=%logl+$
        %logdensity(vprior,%parmspeek(cesparms))
end find
```



**Example 4.1 Non-linear Regression: Random Walk MH**

```

open data cesdata.xls
data(format=xls,org=columns) 1 123 y x1 x2
*
nonlin(parmset=cesparms) gamma1 gamma2 gamma3 gamma4
*
compute gamma4=1.0
compute gamma1=0.0
compute gamma2=gamma3=0.5
*
frml ces y = gamma1+(gamma2*x1^gamma4+gamma3*x2^gamma4)^(1.0/gamma4)
*
nlls(frml=ces,parmset=cesparms)
*
* Prior for variance.
*
compute s2prior=1.0/10.0
compute nuprior=12.0
*
* Prior mean and variance. Because the variance of the prior is now
* independent of the variance of the residuals, the <<hprior>> can be
* taken by just inverting the <<vprior>> without the degrees of freedom
* adjustment.
*
compute [vector] bprior=||1.0,1.0,1.0,1.0||
compute [symm] vprior=.25*%identity(4)
compute [symm] hprior =inv(vprior)
*
compute bdraw =%beta
compute s2draw=%seesq
*
compute nburn =5000
compute ndraws=25000
*
dec series[vect] bgibbs
dec series hgibbs
gset bgibbs 1 ndraws = %zeros(%nreg,1)
set hgibbs 1 ndraws = 0.0
compute accept=0
*
compute fxx=%decomp(%seesq*%xx)
do draw=-nburn,ndraws
  *
  * Draw residual precision conditional on current bdraw
  *
  compute %parmspoke(cesparms,bdraw)
  sstats / (y-ces(t))^2>>rssbeta
  compute rssplus=nuprior*s2prior+rssbeta
  compute hdraw =%ranchisqr(nuprior+%nobs)/rssplus
  *
  * Random walk MC
  *
  compute btest=bdraw+%ranmvnormal(fxx)

```

```

compute %parmspoke(cesparms,btest)
sstats / (y-ces(t))^2>>rsstest
compute logptest=-.5*hdrow*rsstest-.5*qform(hprior,btest-bprior)
compute logplast=-.5*hdrow*rssbeta-.5*qform(hprior,bdraw-bprior)
compute alpha =exp(logptest-logplast)
if %ranflip(alpha)
    compute bdraw=btest, accept=accept+1

if draw<=0
    next
*
*   Do the bookkeeping here.
*
compute bgibbs(draw)=bdraw
compute hgibbs(draw)=hdrow
end do draw
*
disp "Acceptance Rate" accept/(ndraws+nburn+1.0)
*
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,$
    cd=bcd,nse=bnse) bgibbs
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" "Std Error" $
    "NSE" "CD"
do i=1,%nreg
    report(row=new,atcol=1) "Gamma"+i bmean(i) bstderrs(i) bnse(i) bcd(i)
end do i
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.##")
report(action=show)

```

## Example 4.2 Non-linear Regression: Independence MH

```

open data cesdata.xls
data(format=xls,org=columns) 1 123 y x1 x2
*
nonlin(parmset=cesparms) gamma1 gamma2 gamma3 gamma4
*
compute gamma4=1.0
compute gamma1=0.0
compute gamma2=gamma3=0.5
*
frml ces y = gamma1+(gamma2*x1^gamma4+gamma3*x2^gamma4)^(1.0/gamma4)
*
nlls(frml=ces,parmset=cesparms)
*
* Prior for variance.
*
compute s2prior=1.0/10.0
compute nuprior=12.0
*
* Prior mean and variance. Because the variance of the prior is now
* independent of the variance of the residuals, the <<hprior>> can be

```

```

* taken by just inverting the <<vprior>> without the degrees of freedom
* adjustment.
*
compute [vector] bprior=||1.0,1.0,1.0,1.0||
compute [symm]   vprior=.25*identity(4)
compute [symm]   hprior =inv(vprior)
*
compute bdraw=%beta
compute s2draw=%seesq
*
compute nburn =5000
compute ndraws=25000
*
dec series[vect] bgibbs
dec series      hgibbs
gset bgibbs 1 ndraws = %zeros(%nreg,1)
set  hgibbs 1 ndraws = 0.0
compute accept=0
*
* 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 fxx=%decomp(%seesq*%xx)
compute nuxx=10
dec vector tdraw(%nreg)
*
* Since we're starting at %BETA, the kernel of the proposal density is 1.
*
compute logqlast=0.0
*
do draw=-nburn,ndraws
  *
  * Draw residual precision conditional on current bdraw
  *
  compute %parmspoke(cesparms,bdraw)
  sstats / (y-ces(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(cesparms,btest)
  sstats / (y-ces(t))^2>>rsstest
  compute logptest=-.5*hdraw*rsstest-.5*qform(hprior,btest-bprior)
  compute logplast=-.5*hdraw*rssbeta-.5*qform(hprior,bdraw-bprior)
  compute alpha =exp(logptest-logplast+logqlast-logqtest)
  if %ranflip(alpha) {
    compute bdraw=btest, accept=accept+1
    compute logqlast=logqtest
  }
  if draw<=0

```

```

        next
    *
    *   Do the bookkeeping here.
    *
    compute bgibbs(draw)=bdraw
    compute hgibbs(draw)=hdraw
end do draw
*
disp "Acceptance Rate" accept/(ndraws+nburn+1.0)
*
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,$
    cd=bcd,nse=bnse) bgibbs
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" "Std Error" $
    "NSE" "CD"
do i=1,%nreg
    report(row=new,atcol=1) "Gamma"+i bmean(i) bstderrs(i) bnse(i) bcd(i)
end do i
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.##")
report(action=show)

```

## Linear Regression with Non-Spherical Errors

### 5.1 Heteroscedasticity of Known Form

Our model is

$$y_i = X_i\beta + \varepsilon_i, \varepsilon_i \sim N(0, h^{-1}\lambda_i^{-1}) \text{ i.i.d.}$$

The residual precision  $h\lambda_i$  changes from observation to observation. The log likelihood of an element in this is (up to a constant)

$$\frac{1}{2} \log h + \frac{1}{2} \log \lambda_i - \frac{1}{2} h \lambda_i (y_i - X_i\beta)^2$$

Adding across observations, we get

$$\frac{N}{2} \log h + \frac{1}{2} \sum \log \lambda_i - \frac{h}{2} \left( \sum \lambda_i y_i^2 - 2\beta' \left( \sum \lambda_i X_i' y_i \right) + \beta' \left( \sum \lambda_i X_i' X_i \right) \beta \right) \quad (5.1)$$

If  $\lambda_i$  are known (or we condition on them), we can read off:

$$\beta|h, \lambda \sim N\left(\hat{\beta}_{GLS}, h^{-1}\hat{\mathbf{H}}_{GLS}^{-1}\right)$$

$$\hat{\beta}_{GLS} = \left(\sum \lambda_i X_i' X_i\right)^{-1} \sum \lambda_i X_i' y_i, \hat{\mathbf{H}}_{GLS} = \left(\sum \lambda_i X_i' X_i\right)$$

These are the standard formulas for conventional weighted least squares. Note that when they are expressed this way, they are another example of the precision weighted average: data point  $i$  is weighted in the formula by its (relative) residual precision  $\lambda_i$ .

If we have a prior for  $\beta$  which is independent of  $\lambda_i$  and  $h$ , then, as usual, the posterior is

$$\beta|y, h, \lambda \sim N\left(\bar{\beta}, \bar{\mathbf{H}}^{-1}\right)$$

$$\bar{\mathbf{H}} = h\hat{\mathbf{H}}_{GLS} + \mathbf{H}, \bar{\beta} = \bar{\mathbf{H}}^{-1} \left( h\hat{\mathbf{H}}_{GLS}\hat{\beta}_{GLS} + \mathbf{H}\beta \right)$$

Going back to (5.1) and now conditioning on  $\beta$  and  $\lambda_i$ , we get the data evidence on  $h$  as

$$p(h|\beta, \lambda) \propto h^{N/2} \exp\left(-\frac{h}{2} \left(\sum \lambda_i (y_i - X_i\beta)^2\right)\right)$$

With the standard prior

$$h \sim G(\nu, s^{-2})$$

we get the posterior for  $h$  conditional on  $\{\beta, \lambda\}$  as

$$h \sim G(\bar{\nu}, \bar{s}^{-2})$$

$$\bar{\nu} = N + \nu, \bar{s}^2 = \frac{\nu s^2 + \sum \lambda_i (y_i - X_i \beta)^2}{N + \nu}$$

So if the  $\lambda_i$  are fully known (which they might be in cases where the observations are themselves sample averages or sums, so the precision would be a known function of the subsample size), we can do inference on  $\{\beta, h\}$  using the Gibbs sampling technique for the linear model. The only change is that we need the weighted cross product matrix:

$$\begin{bmatrix} \sum \lambda_i X_i' X_i & \sum \lambda_i X_i' y_i \\ \sum \lambda_i y_i X_i & \sum \lambda_i y_i^2 \end{bmatrix}$$

which can be done with RATS using one of `CMOM (SPREAD=1.0/LAMBDA, other options)`  
`CMOM (WEIGHT=LAMBDA, other options)`

In most cases, however, the precision (or variance) of the observations is not known exactly, but is modeled as a function of some set of variables—usually (but not necessarily) some subset of the explanatory variables. It's important to note that the single most important case of heteroscedasticity in modern econometrics (ARCH/GARCH) does not fit into this framework. For those models, the heteroscedasticity is a function of the (lagged) residuals, and hence depends both upon the random data  $y$  and the other parameters  $\beta$ .

This part of the model is usually done in term of the variance rather than the precision. Whether that matters depends upon the functional form. Since we need this formula to generate a positive number, the two most common are:

$$1/\lambda_i = (Z_i \alpha)^2 \quad (5.2a)$$

$$1/\lambda_i = \exp(Z_i \alpha) \quad (5.2b)$$

The second of these can be used either for the variance or the precision, since you can switch from one to the other by flipping the signs of the  $\alpha$ . The first, however, would be quite different if it here applied as a model for the precision, rather than the variance.

When a model like this is estimated conventionally, either by a two-step feasible GLS procedure (see the RATS example `hetero.rpf`) or by maximum likelihood, the formula used generally is designed to model the actual variance at the observation, not the variance up to the scale factor ( $1/h$ ). It's possible to do the same thing in the Bayesian model by pegging the scale factor on  $h$  to one, and forcing all the scale adjustments into the variance model, which would produce the same result. (We can only identify the products  $h\lambda_i$ , so we need to peg the scale of one or the other). However, aside from the fact that we would need to adjust the whole process to handle that, there are good reasons to allow for the common scale factor. Putting the scale and “shape” of the variance function

into separate sets of parameters will help with the simulations since you won't have as much variation in the shape function.

In order to peg the scale of the variance function, we set the “alpha” on the constant to 1 in (5.2a) and to zero in (5.2b). The form used in Koop is

$$1/\lambda_i = (1 + \alpha_1 z_{1i} + \dots + \alpha_p z_{pi})^2$$

Now equation (5.1), when looked at as a function of the  $\alpha$ , takes no known form. However, we can easily compute the (log) density function for any test value of  $\alpha$  (conditional on  $\beta$  and  $h$ ), since it's just a sum of log densities of normals. Thus, this is a job for the one of the M-H procedures. What do we need in order to compute (5.1)? We need the individual residuals  $\varepsilon_i = y_i - X_i\beta$ . Cross products won't help here, because we need to change the weights with each test setting of  $\alpha$ . And, we need to be able to compute the series of  $\lambda_i$  given the current  $\alpha$ . For the latter, we'll do the following (which allows us flexibility to apply this to other problems):

```
linreg(define=baseeqn) price
# constant lot_siz bed bath stor
compute nbeta=%nreg
*
set usq = %resids^2/%seesq-1
linreg(define=vareqn) usq
# lot_siz bed bath stor
compute nalpha=%nreg
compute zxx=%sqrt(%xdiag(%xx))
*
frml(lastreg,vector=avect) zfrml
frml varfrml = (1+zfrml(t))^2
```

As before, we start by estimating the equation of interest. We then take the normalized and centered squared residuals and regress them on the variables that we want in the variance function. (This would be the type of step we would take for a feasible weighted least squares, though we would add a `CONSTANT` to that second regression). The first of the two **FRML** instructions defines a formula which evaluates

$$\alpha_1 LotSize_i + \dots + \alpha_4 Stor_i$$

where the  $\alpha$  are the elements of the `VECTOR AVECT`. **FRML (REGRESSORS)** and **FRML (LASTREG)** are very handy when you need a formula which evaluates a linear combination of observed data, as you can adjust the set of variables by just changing the preceding regression (for `LASTREG`) or trailing list of variables (for `REGRESSORS`). The second **FRML** defines our variance shape formula as a function of the `ZFRML` defined in the first.

Note that if you would prefer the `exp(...)` form for the shape functions, the only changes you would need to make are:

```

set logusq = log(%resids^2/%seesq)
linreg(define=vareqn) logusq
...
frml varfrml = exp(zfrml(t))

```

What's the point of:

```

compute zxx=%sqrt(%xdiag(%xx))

```

$\%XX$  is the  $Z'Z^{-1}$  from the regression of squared residuals on the  $Z$ 's.  $\%XDIAG$  takes the diagonal from that and  $\%SQRT$  takes the element by element square root. So this will create a VECTOR with the square roots of the diagonal elements of  $Z'Z^{-1}$ . This will give us some idea as to the scale of the coefficients. In the data set, for instance, the lot size is in square feet. If that were changed to acres or hectares, the coefficients associated with that would scale up tremendously (on the order of  $10^5$ ). Using a scale-dependent measure will help us to more easily tune the Random Walk M-H.

This will be the first really time-consuming example. Although the non-linear least squares example couldn't use any tricks (such as use of the cross-product matrix) to significantly reduce the calculation time, if you actually do it step by step, you'll find that much of the time is spent on the post processing (in particular, the calculation of the CD measure). This one, however, has quite a bit more  $O(N)$  calculation inside the simulation loop. Because of this, we'll want some visual feedback on how things are going. Our simulation loop will now take the following general form (which we will use when appropriate from now on):



```

infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Walk MH"
do draw=-nburn,ndraws

  do the conditional draws, compute acceptance probability

  if %ranflip(alpha)
    compute adraw=avect,accept=accept+1,recalc=1
  else
    compute recalc=0
  infobox(current=draw) $
    %strval(100.0*accept/(draw+nburn+1),"##.#")
  if draw<=0
    next

  do the bookkeeping

  compute bgibbs(draw)=bdraw
  compute agibbs(draw)=adraw
  compute hgibbs(draw)=hdraw
end do draw
infobox(action=remove)

```

This adds the three **INFOBOX** instructions, which display a progress box on screen. You can tell from this how long the program will take to run, and, since this has a *Cancel* button, you can kill the loop if it's either taking too long, or if you're seeing that it isn't working the way that you would like. See this chapter's *Tips and Tricks* (Section 5.5) for more on **INFOBOX**.

So how do we do the calculations?  $\beta$  and  $h$  aren't that much different than before. However, we can't do a single cross-product matrix outside the loop, since the weights change every time the  $\alpha$ 's do. It's still handy to use a (weighted) cross product matrix, but we'll need to keep recomputing it. We don't want to do an  $O(N)$  calculation unless we have to, so the code in the example is:

```

if recalc {
  compute avect=adraw
  cmom(equation=baseeqn,spread=varfrml)
}

```

`recalc` is set to 1 whenever we need a new cross product matrix and to 0 otherwise. It starts as 1 (so we do the initial calculation) and is set to 1 when we accept a change to `adraw` (which we're using for the current setting of the  $\alpha$ ) and set to 0 when we reject. Because **VARFRML** uses **AVECT** as its vector of parameters, we need to reset **AVECT** each time we want to evaluate **VARFRML** at a new set of  $\alpha$ .

Given that cross-product matrix, the draws for  $\beta$  and  $h$  take the familiar form:

```

compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
compute hdraw  =%ranchisqr(nuprior+%nobs)/rssplus
compute bdraw  =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)

```

We can probably do either Independence Chain or Random Walk M-H for  $\alpha$ . The output from that feasible GLS regression done above will probably work nicely as the basis for the Independence Chain. However, we'll do the Random Walk instead, starting with  $\alpha = 0$  (which would be ordinary least squares). For the increment, we'll use the "re-diagonalized"  $ZXX$  vector described above, as the factor for the multivariate normal. Thus, the increment will have independent Normal components with scales sensitive to the scales of the variables used. Note that it might be necessary to adjust this up or down by some scale factor. Koop uses  $cI$  as the increment and adjusts  $c$  if the acceptance probability is poor. We're just using a scale dependent alternative to  $I$ .

Because the prior on  $\alpha$  is diffuse, the calculation is relatively simple. We first compute the residuals at the current value of `bdraw`. We have to do this every time, since  $\beta$  changes in each pass. The simplest way to do this is with:

```

compute %eqnsetcoeffs(baseeqn,bdraw)
set resids = price-%eqnprj(baseeqn,t)

```

The `%EQNSETCOEFFS(eqn,b)` function resets the coefficient vector for the equation. Then `%EQNPRJ(eqn,t)` returns the value for  $X(t)\beta$  at entry `t`.

For Random Walk M-H, we need to evaluate the posterior density at the old and candidate settings for  $\alpha$ . Again, we do this in log form. Since `VARFRML` uses `AVECT`, we need to reset `AVECT` for each calculation.

```

compute avect=adraw
sstats / %logdensity(varfrml/hdraw,resids)>>logplast
compute avect=adraw+%ranmvnormal(f)
sstats / %logdensity(varfrml/hdraw,resids)>>logptest

```

If we had an informative prior, we would now need to include its log density in the two calculations (evaluated at `adraw` and added to `logplast`, evaluated at the new `avect` and added to `logptest`).

The acceptance logic is now:

```

compute alpha =exp(logptest-logplast)
if %ranflip(alpha)
  compute adraw=avect,accept=accept+1,recalc=1
else
  compute recalc=0

```

Note that we now set the `recalc` variable to 1 or 0 depending upon whether we accept or reject the draw.

Everything else is similar, except that we now do the extra bookkeeping to keep track of the draws for  $\alpha$ .

## 5.2 Heteroscedasticity of Unknown Form

This is the first example of a model for which a Bayesian approach can provide information that can't be obtained easily using a conventional modeling procedure. Every model done previously could have been estimated using maximum likelihood. ML couldn't take into account prior information; and could easily provide different information about the shape of the likelihood function if its log wasn't locally quadratic at the maximum, but these were models for which ML is used routinely.

We'll use the same model as in the last section:

$$y_i = X_i\beta + \varepsilon_i, \varepsilon_i \sim N(0, h^{-1}\lambda_i^{-1}) \text{ i.i.d.}$$

but now the  $\lambda_i$  are going to be allowed to vary "freely". We need some type of informative prior on the  $\lambda_i$  (if they're just unknown constants, we have more parameters than data); in this case, they're assumed to be independent draws from a common gamma distribution. The *hyperparameter* governing this is the degrees of freedom of that gamma (called  $\nu_\lambda$ ).

With flat priors everywhere else, this is equivalent to the  $\varepsilon_i$  being independent  $t$ 's. That model can be estimated with maximum likelihood. It would be an example of a robust estimation procedure for the linear model. (The Student- $t$  has fatter tails than the Normal, and hence is less sensitive to outliers). However, the Bayesian approach also provides estimates of the  $\lambda_i$  for the individual data points, not just a robust estimate for  $\beta$ .<sup>1</sup>  $\beta$  and  $h$  (conditional on  $\lambda_i$ ) are handled as before. Given  $\{\beta, h, \nu_\lambda\}$ , the  $\lambda_i$  are quite simple, since there's only one term in the likelihood that involves each:

$$\lambda_i^{1/2} \exp\left(-\frac{\lambda_i}{2} h (y_i - X_i\beta)^2\right)$$

Combining with the prior

$$\lambda_i^{(\nu_\lambda/2)-1} \exp\left(-\frac{\lambda_i}{2} \nu_\lambda\right)$$

gives

$$p(\lambda_i) \propto \lambda_i^{(\nu_\lambda+1)/2-1} \exp\left(-\frac{\lambda_i}{2} (h (y_i - X_i\beta)^2 + \nu_\lambda)\right)$$

which is gamma with  $\nu_\lambda + 1$  degrees of freedom and mean  $\frac{1 + \nu_\lambda}{h (y_i - X_i\beta)^2 + \nu_\lambda}$ .

You can see the importance of the prior here: if  $\nu_\lambda$  were zero (as it would be if we were trying to do ML on this), this would be undefined in case of a zero residual, and would be extremely large for one very close to zero. That would

---

<sup>1</sup>In conventional statistics, heteroscedasticity of unknown form is generally handled by using least squares for estimation and "correcting" the standard errors using the Eicker-White procedure.

give very high weight to those data points, so instead of the estimates being thrown off by a few outliers, they would be thrown off by the *inliers*. Note that the standard non-Bayesian “trick” for handling this situation is to put a hard limit on the minimum value allowed for  $(y_i - X_i\beta)^2$ .

As discussed earlier, drawing from the gamma as defined by Koop and in most other Bayesian books is most easily done with %RANCHISQR. We need an independent draw for each observation, so:

```
compute %eqnsetcoeffs(baseeqn,bdraw)
set resids = price-%eqnprj(baseeqn,t)
set lambda = %ranchisqr(nudraw+1)/(resids^2*hdraw+nudraw)
```

The tricky part of this model is  $\nu_\lambda$ . We’ve never estimated a degrees of freedom parameter before. It shows up in integrating constants that we’ve been able to ignore until now. With a gamma prior of its own (Koop uses one with a fixed degrees of freedom of 2 and mean  $\nu_\lambda$ , which makes it an exponential), the (log) posterior is the rather ugly:

$$\log p(\nu_\lambda|\lambda) = \frac{N\nu_\lambda}{2} \log \frac{\nu_\lambda}{2} - N \log \Gamma\left(\frac{\nu_\lambda}{2}\right) - \nu_\lambda \left(\frac{1}{\nu_\lambda} + \frac{1}{2} \sum (\lambda_i - \log \lambda_i)\right)$$

This can be handled by M-H. It might be possible to do Independence Chain with a gamma proposal density, if we had any clue what the likely value was. (We could, for instance, estimate the linear regression with independent Student-*t* errors with the degrees of freedom being a parameter). We’ll follow Koop in using M-H with a Normal increment. This could allow the parameter to stray negative; since that’s impossible, we just gives those values an acceptance probability of zero.

In almost all cases up to now, the mean and standard deviation were probably an adequate description of the estimated parameters. That will definitely not be the case of the  $\nu_\lambda$  parameter. Any parameter which is constrained by its nature to be positive or otherwise bounded, will tend to be skewed. Depending upon what you need to do, it’s better to report either percentiles or a density graph. These can be done with:

```
stats(fractiles) ngibbs 1 ndraws
density(grid=automatic,maxgrid=100,smoothing=2.0) $
  ngibbs 1 ndraws xnu fnu
scatter(style=line,vmin=0.0,$
  footer="Figure 6.1 posterior density for degrees of freedom")
# xnu fnu
```

### 5.3 Serially Correlated Errors

While this can be done in greater generality, the model with AR(1) errors remains most important. (Anything more than that is probably better handled by other time series techniques). Our model here is

$$y_t = X_t\beta + \varepsilon_t, \varepsilon_t = \rho\varepsilon_{t-1} + u_t, u_t \sim N(0, h^{-1}) i.i.d.$$

The RATS instruction for estimating using AR(1) errors is **AR1**. That's still handy here, since, with the option `RHO=input value`, you can evaluate the log likelihood at a test value for the serial correlation parameter.  $h$  and  $\beta$  given  $\rho$  are done in a fairly standard way, except that now they apply to the filtered specification:

$$y_t - \rho y_{t-1} = (X_t - \rho X_{t-1})\beta + u_t$$

However, rather than do all the filtering to work these out, it's simpler to use **AR1**.

```
ar1(rho=rhodraw(1),equation=baseeqn,noprint)
```

This does the filtering, computes the mean and (relative) precision of the  $\beta$  given the current value of  $\rho$ . Those are retrievable as `%BETA` (for the mean) and `INV(%XX)` for the precision. Since the example in the book uses a flat prior on  $\rho$ , we don't even need to invert `%XX`. We can get the draws by

```
compute bdraw=%beta+%ranmvnormal(%decomp(%xx/hdraw))
```

Since `%xx` and `hdraw` both change with each pass, there's no advantage in doing any of this calculation outside the loop.

The  $\rho$  parameter is subject to a (Normal) prior. It's most convenient to attack this using a cross-product matrix on the raw (unfiltered) residuals  $y_t = X_t\beta + \varepsilon_t$ . We compute that with:

```
compute %eqnsetcoeffs(baseeqn,bdraw)
set rawresids = pct-%eqnprj(baseeqn,t)
cmom
# rawresids{1} rawresids{0}
```

We do the cross-product matrix in that order (dependent variable last), because that's what the `%RANMVPOSTCMOM` expects.

```
compute rhodraw=%ranmvpostcmom(%cmom,hdraw,$
(1.0/c)*hrhoprior,brhoprior)
```

(The `1.0/c` allows for prior sensitivity calculations).

This isn't all that we need here, though. Because this is a Normal posterior, it could stray outside the stationarity boundaries. We need to truncate it to  $|\rho| < 1$ . With Normal (inverted) likelihood and truncated Normal prior, the posterior is truncated Normal with the same truncation. There's a function

which we'll introduce later (`%RANTRUNCATE`) to take draws from a truncated Normal. However, in this case we'll demonstrate draws using the *acceptance* or *rejection* or *acceptance-rejection* method (all synonyms). In this case, this means simply, draw, test, if it fails the test, draw again. While this isn't as sophisticated as what `%RANTRUNCATE` does, it will work fine here where the out-of-range values will be rare.

If you're really confident in your programming and are sure that the rejection method will work acceptably, you can write this as:

```
loop {
  compute rhodraw=%ranmvpostcmom(%cmom,hdraw,$
    (1.0/c)*hrhoprior,brhoprior)
  if abs(rhodraw(1))<1.0
    break
}
```

**LOOP** creates a perpetual loop. This will only break when the test succeeds, otherwise it will go on until you force-quit. (You can use the *Stop icon*, or, if you have the progress bar, the *Cancel* button on that).

A safer way of handling this is to replace the **LOOP** with a **DO** loop with a large number of repetitions. If this makes it out with `reps==the limit`, you have a problem with either your programming or your rejection algorithm.

```
do reps=1,100
  compute rhodraw=...
  if abs(rhodraw(1))<1.0
    break
end do reps
if reps==100 {
  disp "Made a programming error"
  break
}
```

This example does some additional post-processing that is often displayed. In addition to computing a density function:

```
set rseries = rgibbs(t)(1)
density(grid=automatic,maxgrid=100,smoothing=2.0) rseries / xrho frho
scatter(style=line,vmin=0.0)
# xrho frho
```

the following computes the (maximizing) likelihood at the grid values of  $\rho$

```

set likely 1 100 = 0.0
do t=1,100
    ar1(noprint,rho=xrho(t),equation=baseeqn)
    compute likely(t) = exp(%logl)
end do t
*
scatter(style=line,vmin=0.0,overlay=line,omin=0.0,$
    footer="Posterior vs Likelihood") 2
# xrho frho
# xrho likely

```

This isn't quite the same as the marginal density for the data for  $\rho$ , since it's the maximum density value, not the marginal. But it should give some clue as to how the prior and the data are interacting for that parameter.

## 5.4 Seemingly Unrelated Regressions

This is probably the most important model that we've done so far, since a special case of this is the vector autoregression. And this is our first case with more than one disturbance. Since we're heading in the direction of VAR's, we'll use a slightly different subscripting than Koop, with the more standard indexing of equation ( $i$ ) and time ( $t$ ).

The model takes the form:

$$y_{it} = x'_{it}\beta_i + \varepsilon_{it}; i = 1, \dots, M; t = 1, \dots, T$$

Within a time period, this stacks into

$$\begin{bmatrix} y_{1t} \\ y_{2t} \\ \vdots \\ y_{Mt} \end{bmatrix} = \begin{bmatrix} x'_{1t} & 0 & \dots & 0 \\ 0 & x'_{2t} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & x'_{Mt} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_M \end{bmatrix} + \begin{bmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \\ \vdots \\ \varepsilon_{Mt} \end{bmatrix}$$

or

$$y_t = X_t\beta + \varepsilon_t \tag{5.3}$$

The usual assumption regarding the disturbances is that

$$\varepsilon_t \sim N(0, \mathbf{H}^{-1}), i.i.d.$$

where  $\mathbf{H}$  is the precision matrix (this is more frequently written in the variance form  $N(0, \Sigma)$ ).

This is called a seemingly unrelated regression (SUR) model because there is no obvious connection among the equations (assuming that the  $\beta_i$  have no restrictions across equations). So where do we get an advantage by estimating these jointly?

We haven't really talked about the behavior of the  $x$ 's in the models that we've examined. However, it's important to note that Bayesian methods don't fix underlying problems with the specification of a model. If you examine the typical consumption function

$$c_t = \beta_0 + \beta_1 y_t + \varepsilon_t$$

using the techniques from lesson 2, you will not get an estimate of the marginal propensity to consume in  $\beta_1$ —there's still simultaneous equations bias. Because of that, the standard Normal likelihood is wrong for the situation; if we use it, we get the wrong inference.

For the SUR model, a more proper statement of our assumption is:

$$\varepsilon_t | X_t \sim N(0, \mathbf{H}^{-1}), i.i.d. \quad (5.4)$$

What we gain by using the system is that we have the additional information that  $E(\varepsilon_{it} | X_{jt}) = 0$  for all  $i, j$ , while one-at-a-time estimation uses only  $E(\varepsilon_{it} | X_{it}) = 0$  for each  $i$ .

It's a well-known result that if the explanatory variables are identical in each equation, that is,  $X_{it} = X_{jt}$  for all  $i, j$ , then there's no difference between least squares equation by equation and the (conventional) SUR estimator. That can be shown by doing the calculation (which we'll do), but it's also intuitive from the source of the gain in doing systems estimation: if the  $X$ 's are all the same, then the other equations don't give us any new information about  $\varepsilon_{it}$ .

Even with identical explanatory variables, if we have a proper prior on  $\mathbf{B}$ , we can end up with the systems estimates being different from one-at-a-time unless the prior takes a very specific form. As pointed out by Leamer (1974), the whole point of the model is that there is gain only from the existence of prior information. After all, if  $E(\varepsilon_{it} | X_{jt}) = 0$ , we can legitimately include the  $X_{jt}$  as explanatory variables in equation  $i$ . The fact that they're excluded means that we have (very strong) prior information that they don't belong; which is the same thing as saying that we have a prior with mean zero and infinite precision (zero variance).

The likelihood for observation  $t$  under (5.3) and (5.4) is

$$p(y_t | X_t, \beta, \mathbf{H}) \propto |\mathbf{H}|^{1/2} \exp \left( -\frac{1}{2} (y_t - X_t \beta)' \mathbf{H} (y_t - X_t \beta) \right)$$

For some purposes, this can usefully be rewritten as

$$p(y_t | X_t, \beta, \mathbf{H}) \propto |\mathbf{H}|^{1/2} \exp \left( -\frac{1}{2} \text{trace} \mathbf{H} (y_t - X_t \beta) (y_t - X_t \beta)' \right)$$

which better separates out the two parameters  $\mathbf{H}$  and  $\beta$ .<sup>2</sup>

<sup>2</sup>This uses the result that  $\text{trace } \mathbf{AB} = \text{trace } \mathbf{BA}$  if the two products make sense.



Multiplying across  $T$  observations gives

$$p(y|X, \beta, \mathbf{H}) \propto |\mathbf{H}|^{T/2} \exp \left( -\frac{1}{2} \text{trace} \mathbf{H} \sum_t (y_t - X_t \beta) (y_t - X_t \beta)' \right)$$

If we look at this as a density for  $\mathbf{H}$ , it's our first example of a Wishart distribution (see Appendix B.7). The Wishart is a generalization of the gamma to several variables—it provides a density over the positive definite symmetric matrices (that is, proper covariance matrices or precision matrices). If we compare with the general form:

$$\exp \left( -\frac{1}{2} \text{trace} (\mathbf{A}^{-1} \mathbf{X}) \right) |\mathbf{X}|^{\frac{\nu-n-1}{2}}$$

( $n$  is the dimension, so  $M$  in our case) we can read off

$$\nu - M - 1 = T \Rightarrow \nu = T + M + 1$$

$$\mathbf{A}^{-1} = \sum_t (y_t - X_t \beta) (y_t - X_t \beta)' \Rightarrow \mathbf{A} = \left( \sum_t (y_t - X_t \beta) (y_t - X_t \beta)' \right)^{-1}$$

Now as little as we know about the variances of the individual components, we probably know even less about the complete covariance matrix. So the prior that is almost always used for  $\mathbf{H}$  is the non-informative prior:

$$|\mathbf{H}|^{-\frac{M+1}{2}}$$

If we combine the likelihood with this prior, we get exactly the same distribution as before, but with the degrees of freedom corrected down to  $\nu = T$ .

So how do we get draws from a Wishart? A Wishart with the scale  $\mathbf{A} = \mathbf{I}_M$  and integer degrees of freedom  $\nu$  could be done by

$$\mathbf{H} = \sum_{i=1}^{\nu} Z_i Z_i' \quad \text{where} \quad Z_i \sim N(0, \mathbf{I}_M) \text{ i.i.d.}$$

This is the generalization of the chi-squared being the sum of squared independent Normals. However, for large degrees of freedom (and we're looking at  $T$  degrees of freedom), that's rather inefficient, and it doesn't apply to the (admittedly) rare case with non-integer  $\nu$ . There's a much more efficient way to do this which draws a Cholesky factor of  $\mathbf{H}$ , which ends up being just a mix of univariate gammas and Normals. You don't need to know the details of this as long as you're working with a program like RATS that has the Wishart draws built in.

The function `%RANWISHART(M, nu)` draws a “standard” Wishart with shape parameter  $\mathbf{I}_M$  and degrees of freedom  $\text{nu}$ . To convert that to a more general shape parameter, you need to sandwich it between (any) factor of  $\mathbf{A}$ . If  $\mathbf{A} = \mathbf{F}\mathbf{F}'$  (the Cholesky factor is usually the simplest to compute), then a draw from a Wishart with shape  $\mathbf{A}$  and degrees of freedom  $\nu$  can be done by  $\mathbf{F}\mathbf{X}\mathbf{F}'$ , where  $\mathbf{X}$

is a standard Wishart with  $\nu$  degrees of freedom. Since that's the most natural use of the Wishart, RATS provides a separate function, `%RANWISHARTF(F, nu)` where `F` is a factor of the scale matrix and `nu` is the degrees of freedom. (The size of the target matrix is clear from the dimension of `F`). As with the draws from the multivariate Normal, using the factor as the input might allow you to pull part of the calculation outside the loop. If that won't help (that is, if you only need one draw with a particular scale matrix), you can just use `%RANWISHARTF(%DECOMP(A), NU)`.

To get the scale parameter for the Wishart, we need to compute:

$$\sum_t (y_t - X_t \beta) (y_t - X_t \beta)' \quad (5.5)$$

which is the sum of the outer products  $(M \times 1)(1 \times M)$  of the residuals

$$\varepsilon_t = y_t - X_t \beta$$

Let's look first at the important special case where the explanatory variables are all the same (such as in a VAR). We can then rewrite the model in the more convenient form:

$$y_t = \begin{bmatrix} \beta'_1 \\ \vdots \\ \beta'_M \end{bmatrix} x'_{\bullet t} + \varepsilon_t = \mathbf{B} x'_{\bullet t} + \varepsilon_t$$

where  $x'_{\bullet t}$  is the  $k$ -vector of explanatory variables at  $t$ . The coefficients are now written more compactly as the  $M \times k$  matrix  $\mathbf{B}$ . (5.5) then converts to

$$\sum_t (y_t - \mathbf{B} x'_{\bullet t}) (y_t - \mathbf{B} x'_{\bullet t})'$$

which expands to:

$$\left( \sum y_t y'_t \right) - \mathbf{B} \left( \sum x'_{\bullet t} y'_t \right) - \left( \sum y_t x_{\bullet t} \right) \mathbf{B}' + \mathbf{B} \left( \sum x'_{\bullet t} x_{\bullet t} \right) \mathbf{B}'$$

The sums here are all submatrices of the cross product matrix:

$$\sum \begin{bmatrix} x'_{\bullet t} \\ y_t \end{bmatrix} \begin{bmatrix} x_{\bullet t} & y'_t \end{bmatrix}$$

For a given set of coefficients, we can compute (5.5) using

$$\begin{bmatrix} -\mathbf{B} & \mathbf{I}_M \end{bmatrix} \begin{bmatrix} \mathbf{X}'\mathbf{X} & \mathbf{X}'\mathbf{y}' \\ \mathbf{y}\mathbf{X} & \mathbf{y}\mathbf{y}' \end{bmatrix} \begin{bmatrix} -\mathbf{B}' \\ \mathbf{I}_M \end{bmatrix} \quad (5.6)$$

This rids us of an  $O(T)$  calculation inside the loop.

And if the  $x$ 's aren't identical, we can do the same thing by making  $x'_{\bullet t}$  a single vector with all the explanatory variables (ideally eliminating repetitions). The  $\mathbf{B}$  matrix will have zeros in it where variables are left out of an equation, but those are always in the same positions. If we have a draw for  $\beta$ , we can just "poke" its values into the proper slots in  $\mathbf{B}$  and use (5.6).

We'll leave the special case of the VAR until Chapter 6. For the general SUR, you start by defining the collection of equations that make up the model:

```
linreg(define=yankeeeqn) y_pct
# constant y_obp y_slg y_era
linreg(define=redsoxeqn) r_pct
# constant r_obp r_slg r_era
*
group baseball yankeeeqn redsoxeqn
```

This estimates the two equations by OLS, and (as a side effect) defines them as `yankeeeqn` and `redsoxeqn`. The **GROUP** instruction then defines the model `baseball` as the combination of the two.

The procedure `@SURGibbsSetup` examines the model and figures out the positions in the grand B matrix for the coefficients.

```
@SURGibbsSetup baseball
```

Two other procedures or functions are used inside the simulation loop to help with the draws. The function `SURGibbsSigma(bdraw)` computes (5.5) given the current value of `bdraw`. And the procedure

```
@SURGibbsDataInfo hdraw hdata hbdata
```

uses the current value of `hdraw` to compute the data evidence on the precision of  $\beta$  (`hdata`) and that precision times the mean (`hbdata`).

The draw for  $\beta$  is then done using a fairly standard set of calculations. With this model, we can do straight Gibbs sampling, and with no  $O(T)$  calculation inside the loop, it goes very quickly.

```
compute hpost=hdata+hprior
compute vpost=inv(hpost)
compute bpost=vpost*(hbdata+hprior*bprior)
compute bdraw=bpost+%ranmvnormal(%decomp(vpost))
```

## 5.5 RATS Tips and Tricks

### The INFOBOX instruction

You can use **INFOBOX** if you want feedback about what is happening during a lengthy operation. As always, the first step is to see that the loop is doing the calculation correctly with a small number of draws. When you ramp up to a large number of draws, however, it's generally a good idea to include the **INFOBOX**. This brings up a dialog box which sits on top of the other RATS windows. You can display a graphical *progress bar* and up to two lines of text in the box. There are three stages to this: one before the loop to set up the dialog box, one inside it to update the progress, and one at the end to remove it from the screen:

```
infobox(action=define,other options) "top messagestring"
infobox(other options) "bottom messagestring"
infobox(action=remove)
```

In our typical case, these three lines are:

```
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
    "Random Walk MH"
infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")
infobox(action=remove)
```

The first defines the progress bar as running from `-nburn` to `ndraws` which is the range of the `do draw` loop. The messagestring on this is the top line on the dialog, and cannot be changed later. The **INFOBOX** in the middle of the loop updates the progress bar with the current value of `draw`. It also (in this case), shows what our running acceptance probability is for M-H. The progress bar has a *Cancel* button. If the acceptance rate is running too low, you can kill the process and try retuning your settings. Note that `lower`, `upper` and `current` must be integers. The final **INFOBOX** outside the loop gets rid of the progress bar.

### The %EQN function family

An **EQUATION** is an object describing the relationship  $y_t = X_t\beta$ . The family of functions with names beginning **%EQN** can be used to take information out of and put it into an equation. By using these, you can more easily adjust a model—adding and removing variables—without otherwise having to change your program. The full list can be found in the *Equation/Regression Lists* category on the *Functions* wizard. The most important of these are:

<b>%EQNSIZE (eqn)</b>	Returns the number of variables in $X$
<b>%EQNREGLABELS (eqn)</b>	Returns a <b>VECTOR[STRINGS]</b> with the labels for the $X$ variables, formatted as they are in RATS output

<b>%EQNXVECTOR (eqn, t)</b>	Returns the VECTOR $X_t$
<b>%EQNSETCOEFFS (eqn, b)</b>	Resets the coefficient vector $\beta$ for the equation.
<b>%EQNPRJ (eqn, t)</b>	Returns the value $X_t\beta$ given the current set of coefficients $\beta$

With any of these, you can use a 0 in the `eqn` parameter to refer to the most recent (linear) regression.

### The DENSITY instruction

The **DENSITY** instruction does kernel density estimation. Estimated density functions are often used in Bayesian techniques, particularly to display posterior densities which might not be well-described by a mean and standard deviation. The density needs to be estimated across a grid of points. For all the uses in this book, we'll use the option `GRID=AUTOMATIC`, which creates the grid based upon the range of values for the input series.

The default bandwidth for the estimator is

$$(.79IQR) N^{-1/5}$$

where  $IQR$ =interquartile range, and  $N$ =number of observations. This has certain optimality properties in larger samples, but you might find it to be too narrow, particular where the density is fairly flat. (It's targeted at a Normal). In practice, we generally find that increasing this by a factor of between 1.5 and 2.0 seems to estimates that look better when graphed. The **DENSITY** and **SCATTER** instructions used in this chapter were generated by the (Kernel) Density Estimation wizard added with RATS version 7.2. The only changes made to these were to increase the bandwidth (by using the option `SMOOTHING=2.0`, which doubles the default bandwidth), and add the `FOOTER` option to the **SCATTER**.

**Example 5.1 Heteroscedastic errors with a known form**

```

open data hprice.prn
data(format=prn,org=columns) 1 546 price lot_siz bed bath stor drive $
  recroom bment gas aircond gar desireloc
*
* This is the equation we're analyzing
*
linreg(define=baseeqn) price
# constant lot_siz bed bath stor
compute nbeta=%nreg
compute bdraw=%beta
compute s2draw=%seesq
*
* Prior mean and variance. Because the variance of the prior is now
* independent of the variance of the residuals, the <<hprior>> can be
* taken by just inverting the <<vprior>> without the degrees of freedom
* adjustment.
*
compute [vector] bprior=||0.0,10.0,5000.0,10000.0,10000.0||
compute [symm]   vprior=%diag(||10000.0^2,5.0^2,2500.0^2,5000.0^2,5000.0^2||)
compute [symm]   hprior =inv(vprior)
*
* Prior for (proportional) variance.
*
compute s2prior=5000.0^2
compute nuprior=5.0
dec real hdraw
*
* Define the formula for the systematic part of the variance
*
set usq = %resids^2/%seesq-1
linreg(define=vareqn) usq
# lot_siz bed bath stor
compute nalpha=%nreg
compute zxx=%sqrt(%xdiag(%xx))
*
frml(lastreg,vector=avect) zfrml
frml varfrml = (1+zfrml(t))^2
*
* We'll start the Gibbs sampler at the OLS estimates
*
compute adraw =%zeros(nalpha,1)
set resids = %resids
*
compute nburn =5000
compute ndraws=25000
compute accept=0
compute recalc=1
*
dec series[vect] bgibbs agibbs
dec series      hgibbs
gset bgibbs 1 ndraws = %zeros(nbeta,1)
gset agibbs 1 ndraws = %zeros(nalpha,1)

```

```

set hgibbs 1 ndraws = 0.0
*
* This is the covariance matrix for the R-W increment. If the percentage
* of acceptances is off, this may need to be adjusted to compute f=some
* constant x %diag(zxx).
*
compute f=%diag(zxx)
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) "Random Walk MH"
do draw=-nburn,ndraws
  *
  * Compute cross product matrix, weighted with the current scedastic
  * function
  *
  if recalc {
    compute avect=adraw
    cmom(equation=baseeqn,spread=varfrml)
  }
  *
  * Get a draw for h, given bdraw and adraw
  *
  compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
  compute hdraw =%ranchisqr(nuprior+%nobs)/rssplus
  *
  * Draw betas given hdraw and avect
  *
  compute bdraw =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
  *
  * Do random walk \textsc{m-h} attempt on avect
  *
  compute %eqnsetcoeffs(baseeqn,bdraw)
  set resids = price-%eqnprj(baseeqn,t)
  compute avect=adraw
  sstats / %logdensity(varfrml/hdraw,resids)>>logplast
  compute avect=adraw+%ranmvnormal(f)
  sstats / %logdensity(varfrml/hdraw,resids)>>logptest
  compute alpha =exp(logptest-logplast)
  if %ranflip(alpha)
    compute adraw=avect,accept=accept+1,recalc=1
  else
    compute recalc=0

  infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")
  if draw<=0
    next
  *
  * Do the bookkeeping here.
  *
  compute bgibbs(draw)=bdraw
  compute agibbs(draw)=adraw
  compute hgibbs(draw)=hdraw
end do draw
infobox(action=remove)
*

```

```

@mcmcpostproc(ndraws=ndraws,mean=amean,stderrs=astderrs,cd=acd) agibbs
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,cd=bcd) bgibbs
*
disp "Regression Coefficients"
do i=1,nbeta
    disp %eqnreglabels(baseeqn)(i) @20 bmean(i) bstderrs(i) bcd(i)
end do i
*
disp
disp "Variance Equation Coefficients"
do i=1,nalpha
    disp %eqnreglabels(vareqn)(i) @20 amean(i) astderrs(i) acd(i)
end do i

```

## Example 5.2 Heteroscedastic errors with a unknown functional form

```

open data hprice.prn
data(format=prn,org=columns) 1 546 price lot_siz bed bath stor drive $
    recroom bment gas aircond gar desireloc
*
* This is the equation we're analyzing
*
linreg(define=baseeqn) price
# constant lot_siz bed bath stor
compute nbeta=%nreg
compute bdraw=%beta
compute s2draw=%seesq
*
* Prior mean and variance. Because the variance of the prior is now
* independent of the variance of the residuals, the <<hprior>> can be
* taken by just inverting the <<vprior>> without the degrees of freedom
* adjustment.
*
compute [vector] bprior=||0.0,10.0,5000.0,10000.0,10000.0||
compute [symm]    vprior=%diag(||10000.0^2,5.0^2,2500.0^2,5000.0^2,5000.0^2||)
compute [symm]    hprior =inv(vprior)
*
* Prior for (proportional) variance.
*
compute s2prior=5000.0^2
compute nuprior=5.0
dec real hdraw
*
* Prior for nu (degrees of freedom=2)
*
compute nulambdap=25.0
compute nudraw    =nulambdap
*
* We'll start the Gibbs sampler at the OLS estimates
*
set lambda = 1.0
set resids = %resids

```



```

compute nburn =5000
compute ndraws=30000
compute accept=0
compute recalc=1
*
dec series[vect] bgibbs
dec series      hgibbs
dec series      ngibbs
gset bgibbs 1 ndraws = %zeros(nbeta,1)
set  hgibbs 1 ndraws = 0.0
set  ngibbs 1 ndraws = 0.0
*
* This is the tuning parameter for the RW M-H draws on nu. Tried c=1 to
* start; when that came in with too low an acceptance, we changed to 0.5
*
compute c=0.5
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) "Random Walk MH"
do draw=-nburn,ndraws
  *
  * Compute cross product matrix, weighted with the current scedastic
  * function Because lambda changes each time through, we have to
  * recalculate
  *
  cmom(equation=baseeqn,spread=1.0/lambda)
  *
  * Get a draw for h, given bdraw and lambda
  *
  compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
  compute hdraw  =%ranchisqr(nuprior+%nobs)/rssplus
  *
  * Draw betas given hdraw and lambda
  *
  compute bdraw  =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
  *
  * Draw lambdas given residuals, hdraw and nudraw
  *
  compute %eqnsetcoeffs(baseeqn,bdraw)
  set resid = price-%eqnprj(baseeqn,t)
  set lambda = %ranchisqr(nudraw+1)/(resids^2+hdraw+nudraw)
  *
  * Draw nu given lambdas
  *
  compute nutest=nudraw+%ran(c)
  if nutest<=0.0
    compute alpha=0.0
  else {
    sstats / lambda-log(lambda)>>sumlambda
    compute logplast=.5*%nobs*nudraw*log(.5*nudraw)-$
      %nobs*%lngamma(.5*nudraw)-nudraw*(1.0/nulambdap+.5*sumlambda)
    compute logptest=.5*%nobs*nutest*log(.5*nutest)-$
      %nobs*%lngamma(.5*nutest)-nutest*(1.0/nulambdap+.5*sumlambda)
    compute alpha =exp(logptest-logplast)
  }
}

```

```

if %ranflip(alpha)
  compute nudraw=nutest, accept=accept+1

infobox(current=draw) %strval(100.0*accept/(draw+nburn+1), "##.#")
if draw<=0
  next
*
*   Do the bookkeeping here.
*
compute bgibbs(draw)=bdraw
compute hgibbs(draw)=hdraw
compute ngibbs(draw)=nudraw
end do draw
infobox(action=remove)
*
@mcmcpostproc(ndraws=ndraws, mean=bmean, stderrs=bstderrs, cd=bcd) bgibbs
disp "Regression Coefficients"
do i=1, %nreg
  disp %eqnreglabels(baseeqn)(i) @20 bmean(i) bstderrs(i) bcd(i)
end do i
*
stats(fractiles) ngibbs 1 ndraws
density(grid=automatic, maxgrid=100, smoothing=2.0) ngibbs 1 25000 xnu fnu
scatter(style=line, vmin=0.0, $
  footer="Figure 6.1 Posterior density for degrees of freedom")
# xnu fnu
*
* Estimates the model by ML with t distributed errors with unknown
* degrees of freedom. (The hdraw parameter is scaled up by 10^6 because
* its value is so small). Scaling isn't generally a problem for Bayesian
* methods since they don't take derivatives and don't do "convergence
* tests", both of which can be sensitive to the scale of parameters.
*
compute nudraw=4.0
frml(equation=baseeqn, vector=bdraw) basefrml
compute hdraw=hdraw*1.e+6
frml studentml = %logtdensitystd(1.e+6/hdraw, $
  price-%dot(%eqnxvector(baseeqn, t), bdraw), nudraw)
nonlin bdraw hdraw nudraw
maximize studentml

```

### Example 5.3 Linear regression with AR(1) errors

```

open data yankees.txt
calendar(a) 1903
data(format=prn, org=columns) 1903:01 1999:01 pct r avg obp slg sb era
*
* This is the equation we're analyzing
*
linreg(define=baseeqn) pct
# constant obp slg era
compute startar=%regstart()+1, endar=%regend()
*

```

```

compute nbeta=%nreg
compute bdraw=%beta
compute s2draw=%seesq
*
* Prior for rho
*
compute [vector] brhoprior=||0.0||
compute [symm] hrhoprior=%identity(1)
*
* We'll start the Gibbs sampler at the OLS estimates
*
compute nburn =5000
compute ndraws=25000
compute accept=0
compute recalc=1
*
dec series[vect] bgibbs rgibbs
dec series          hgibbs
gset bgibbs 1 ndraws = %zeros(nbeta,1)
gset rgibbs 1 ndraws = %zeros(1,1)
set  hgibbs 1 ndraws = 0.0
*
set resids = %resids
dec vect rhodraw(1)
compute rhodraw(1)=0.0
*
compute c=.09
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Gibbs Sampling"
do draw=-nburn,ndraws
  *
  * Get a draw for h, given bdraw and rhodraw
  *
  sstats startar endar resids^2>>rssplus
  compute hdraw =%ranchisqr(%nobs)/rssplus
  *
  * Draw betas given hdraw and rhodraw
  *
  ar1(rho=rhodraw(1),equation=baseeqn,noprint)
  compute bdraw =%beta+%ranmvnormal(%decomp(%xx/hdraw))
  *
  compute %eqnsetcoeffs(baseeqn,bdraw)
  set rawresids = pct-%eqnprj(baseeqn,t)
  *
  cmom
  # rawresids{1} rawresids{0}
  *
  * Keep drawing until we get a draw in the stationary zone. Note that
  * if we did a major programming error, this could loop forever. When
  * debugging something like this, put a limit on the number of times
  * you do it:
  * do reps=1,100
  * compute rhodraw=...

```

```

*   if abs(rhodraw(1))<1.0
*       break
* end do reps
* if reps==100 {
*   disp "Made a programming error"
*   break
* }
*
loop {
  compute rhodraw=$
    %ranmvpostcmom(%cmom,hdraw,(1.0/c)*hrhoprior,brhoprior)
  if abs(rhodraw(1))<1.0
    break
  }
  set resid = rawresids-rhodraw(1)*rawresids{1}
  infobox(current=draw)
  if draw<=0
    next
  *
  *   Do the bookkeeping here.
  *
  compute bgibbs(draw)=bdraw
  compute hgibbs(draw)=hdraw
  compute rgibbs(draw)=rhodraw
end do draw
infobox(action=remove)
*
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,cd=bcd) bgibbs
@mcmcpostproc(ndraws=ndraws,mean=rmean,stderrs=rstderrs,cd=rcd) rgibbs
*
set rseries = rgibbs(t)(1)
density(grid=automatic,maxgrid=100,smoothing=2.0) rseries / xrho frho
*
* Compute the likelihood function for different settings of rho
* Graph against the estimated density function.
*
set likely 1 100 = 0.0
do t=1,100
  ar1(noprint,rho=xrho(t),equation=baseeqn)
  compute likely(t) = exp(%logl)
end do t
*
scatter(style=line,vmin=0.0,overlay=line,omin=0.0,$
  footer="Posterior vs Likelihood") 2
# xrho frho
# xrho likely

```

**Example 5.4 Seemingly unrelated regression**

```

open data baseball.xls
calendar(a) 1903
data(format=xls,org=columns) 1903:01 1999:01 $
  y_pct y_r y_avg y_obp y_slg y_sb y_era $
  r_pct r_r r_avg r_obp r_slg r_sb r_era
*
* This is the pair of equations we're analyzing
*
linreg(define=yankeeeqn) y_pct
# constant y_obp y_slg y_era
linreg(define=redsoxeqn) r_pct
# constant r_obp r_slg r_era
*
group baseball yankeeeqn redsoxeqn
*
@SURGibbsSetup baseball
*
sur(noprint,nosigma,model=baseball,cv=%identity(neqn),cmom)
compute bdraw=%beta
*
* Prior for the regression coefficients
*
compute [symmetric] hprior=.25*%identity(ntotal)
compute [vector] bprior=%zeros(ntotal,1)
*
* Prior for the residual precision matrix
*
compute hxxprior=%zeros(neqn,neqn)
compute nuprior =0.0
*
* We'll start the Gibbs sampler at the OLS estimates
*
compute nburn =5000
compute ndraws=25000
*
dec series[vect] bgibbs
dec series      rgibbs
gset bgibbs 1 ndraws = %zeros(ntotal,1)
set  rgibbs 1 ndraws = 0.0
*
compute wishdof=nuprior+%nobs
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) "Gibbs Sampling"
do draw=-nburn,ndraws
  *
  * Get a draw for h, given bdraw Compute covariance matrix of
  * residuals at the current beta
  *
  compute covmat=SURGibbsSigma(bdraw)+hxxprior
  *
  compute hdraw=%ranwishartf(%decomp(inv(covmat)),wishdof)
  *

```

```

@SURGibbsDataInfo hdraw hdata hbdata
*
compute hpost=hdata+hprior
compute vpost=inv(hpost)
compute bpost=vpost*(hbdata+hprior*bprior)
compute bdraw=bpost+%ranmvnormal(%decomp(vpost))
infobox(current=draw)
if draw<=0
    next
*
* Do the bookkeeping here.
*
compute bgibbs(draw)=bdraw
*
* To get the correlation, invert hdraw and convert it to a
* correlation matrix using %cvtocorr. Pull out the (1,2) element of
* that.
*
compute corrmatrix=%cvtocorr(inv(hdraw))
compute rgibbs(draw)=corrmatrix(1,2)
end do draw
infobox(action=remove)
*
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,cd=bcd) bgibbs
report(action=define)
report(atrow=2,atcol=1,fillby=col) bmean
report(atrow=2,atcol=2,fillby=col) bstderrs
report(atrow=2,atcol=3,fillby=col) bmean-1.96*bstderrs
report(atrow=2,atcol=4,fillby=col) bmean+1.96*bstderrs
report(action=format,picture="*.##")
report(action=show)
*
stats rgibbs 1 ndraws

```

## Vector Autoregressions

### 6.1 Flat Prior

Most of what we will do here applies not just to VAR's, but to any linear systems regression with identical explanatory variables in each equation. For instance, in finance it's common to have several returns regressed upon a common set of factors. If those factors are observable (or are constructed separately), we have exactly this type of model.

With identical explanatory variables, the model can be written in either the form:

$$y_t = (\mathbf{I}_M \otimes x_t) \beta + \varepsilon_t, \varepsilon_t \sim N(0, \mathbf{H}^{-1}), i.i.d.$$

or

$$y_t = \mathbf{B} x_t' + \varepsilon_t \tag{6.1}$$

where  $x_t$  is the  $k$  vector of explanatory variables,  $\otimes$  is the Kroneker product,  $\beta$  is the  $kM$  vector of coefficients for the full system and  $\mathbf{B}$  is the reshaped  $M \times k$  matrix of the same.  $\mathbf{B}$  "vec'ed" by rows gives  $\beta$ , that is, the first  $k$  elements of  $\beta$  are the first row of  $\mathbf{B}$ , the next  $k$  elements are the second row, etc. The first form will be more convenient for analyzing the  $\beta$ , while the second will be better for analyzing the precision matrix, and for computing the likelihood function. For analyzing  $\beta$ , we're best off keeping the likelihood for a data point  $t$  in the form:

$$p(y_t | x_t, \beta, \mathbf{H}) \propto |\mathbf{H}|^{1/2} \exp \left( -\frac{1}{2} (y_t - (\mathbf{I}_M \otimes x_t) \beta)' \mathbf{H} (y_t - (\mathbf{I}_M \otimes x_t) \beta) \right)$$

When we multiply across  $t$ , we get the following as the quadratic form involving  $\beta$ :

$$\sum (y_t - (\mathbf{I}_M \otimes x_t) \beta)' \mathbf{H} (y_t - (\mathbf{I}_M \otimes x_t) \beta)$$

or

$$\sum y_t \mathbf{H} y_t' - 2\beta' \sum (\mathbf{I}_M \otimes x_t)' \mathbf{H} y_t + \beta' \sum (\mathbf{I}_M \otimes x_t)' \mathbf{H} (\mathbf{I}_M \otimes x_t) \beta$$

(Since this is a scalar, the two cross terms are equal, hence the  $2 \times$ )

One of the convenient properties of the Kroneker product is that

$$(\mathbf{A} \otimes \mathbf{B}) (\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}$$

if the dimensions work. In particular,

$$(\mathbf{I} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{I}) = \mathbf{C} \otimes \mathbf{B}$$

allowing certain types of matrices to “commute”. Shrewdly using  $\mathbf{I}_1$  matrices (that is, the constant 1), allows us to do the following:

$$(\mathbf{I}_M \otimes x_t)' \mathbf{H} (\mathbf{I}_M \otimes x_t) \Rightarrow (\mathbf{I}_M \otimes x_t)' (\mathbf{H} \otimes \mathbf{I}_1) (\mathbf{I}_M \otimes x_t) \Rightarrow \mathbf{H} \otimes x_t' x_t$$

and

$$\begin{aligned} \sum (\mathbf{I}_M \otimes x_t)' \mathbf{H} y_t &\Rightarrow \sum (\mathbf{H} \otimes x_t)' y_t \Rightarrow \\ \sum (\mathbf{H} \otimes \mathbf{I}_k) (\mathbf{I}_M \otimes x_t)' (y_t \otimes \mathbf{I}_1) &\Rightarrow \sum (\mathbf{H} \otimes \mathbf{I}_k) (y_t \otimes x_t') \end{aligned} \quad (6.2)$$

which gives us

$$\sum y_t' \mathbf{H} y_t - 2\beta' \sum (\mathbf{H} \otimes \mathbf{I}_k) (y_t \otimes x_t') + \beta' \sum (\mathbf{H} \otimes x_t' x_t) \beta$$

or (after pulling sums through)

$$\sum y_t' \mathbf{H} y_t - 2\beta' (\mathbf{H} \otimes \mathbf{I}_k) \sum (y_t \otimes x_t') + \beta' \left( \mathbf{H} \otimes \sum x_t' x_t \right) \beta$$

From the properties of multivariate Normals (Appendix C), we read off that the precision of  $\beta$  is

$$\hat{\mathbf{H}} = \mathbf{H} \otimes \sum x_t' x_t$$

and the mean is

$$\hat{\beta} = \left( \mathbf{H} \otimes \sum x_t' x_t \right)^{-1} (\mathbf{H} \otimes \mathbf{I}_k) \sum (y_t \otimes x_t') = \left( \mathbf{I}_M \otimes \sum x_t' x_t \right)^{-1} \sum (y_t \otimes x_t') \quad (6.3)$$

Since  $\sum (y_t \otimes x_t')$  is just a vec'd version of the equation by equation  $\sum x_t' y_{it}$  vectors,  $\hat{\beta}$  is just equation by equation least squares, independent of  $\mathbf{H}$ . For a general SUR model, we don't get this result because it's the properties of the Kroneker product that allow us to pull the  $\mathbf{H}$  out from between the  $x$  and  $y$  factors in (6.2).

Let's return to the alternative form of the likelihood that we used in Section 5.4, and the alternative way of writing the model, which gives us

$$p(y|X, \mathbf{B}, \mathbf{H}) \propto |\mathbf{H}|^{T/2} \exp \left( -\frac{1}{2} \text{trace} \mathbf{H} (T \times \Sigma(\mathbf{B})) \right) \quad (6.4)$$

where we define

$$\Sigma(\mathbf{B}) = \frac{1}{T} \sum (y_t - \mathbf{B}x_t') (y_t - \mathbf{B}x_t')'$$

which is the covariance matrix evaluated at the matrix of coefficients  $\mathbf{B}$ .

Part of (6.4) will be the density for  $\hat{\beta}$ . That needs an integrating constant of

$$\left| \mathbf{H} \otimes \sum x_t' x_t \right|^{1/2} = |\mathbf{H}|^{k/2} \left| \sum x_t' x_t \right|^{M/2}$$



(This is another property of Kroneker products). The  $|\sum x'_t x_t|$  is just data, so we can ignore it. We do, however, need to allow for the  $|\mathbf{H}|^{k/2}$  factor. The exponent in the kernel for  $\beta$  has to be zero at the mean  $\hat{\beta}$ , so the “leftover” exponent has to be the non-zero part at  $\hat{\beta}$ , allowing us to rewrite (6.4) as:

$$p(y|X, \mathbf{B}, \mathbf{H}) \propto |\mathbf{H}|^{(T-k)/2} \exp\left(-\frac{1}{2} \text{trace} \mathbf{H}(T \times \Sigma(\hat{\mathbf{B}}))\right) \times p(\beta|y, \mathbf{H})$$

The standard non-informative (“flat”) prior for this model takes the form

$$p(\beta, \mathbf{H}) \propto |\mathbf{H}|^{-\frac{M+1}{2}}$$

allowing us to read off the posterior for  $\mathbf{H}$  as

$$p(\mathbf{H}|y) \propto |\mathbf{H}|^{\frac{T-k-(M+1)}{2}} \exp\left(-\frac{1}{2} \text{trace} \mathbf{H}(T \times \Sigma(\hat{\mathbf{B}}))\right)$$

This is a Wishart with  $T-k$  degrees of freedom and scale matrix  $(T \times \Sigma(\hat{\mathbf{B}}))^{-1}$ . Unlike the case of the general SUR model, the scale matrix on this is a function just of the data (it depends upon  $\hat{\beta}$ , not  $\mathbf{B}$ ), so we can do direct draws from it (rather than Gibbs draws). What we end up needing for all other purposes is actually  $\Sigma \equiv \mathbf{H}^{-1}$ , so we’ll use the `%RANWISHARTI` function, , which draws an inverse Wishart.

The precision matrix for  $\beta$  is  $\hat{\mathbf{H}} = \mathbf{H} \otimes \sum x'_t x_t$ , so the covariance matrix is

$$\Sigma \otimes \left(\sum x'_t x_t\right)^{-1} \quad (6.5)$$

In order to draw from a multivariate Normal distribution with this as covariance matrix, we need a factor of this matrix. Fortunately, if we have factors of the two components:

$$\mathbf{F}_\Sigma \mathbf{F}_\Sigma' = \Sigma, \mathbf{F}_X \mathbf{F}_X' = \left(\sum x'_t x_t\right)^{-1}$$

then we can get a factor of (6.5) with

$$(\mathbf{F}_\Sigma \otimes \mathbf{F}_X) (\mathbf{F}_\Sigma \otimes \mathbf{F}_X)' = \Sigma \otimes \left(\sum x'_t x_t\right)^{-1}$$

$\mathbf{F}_X$  can be computed once, outside the simulation loop. Since there will be a new draw for  $\Sigma$  each pass, we’ll have to recompute  $\mathbf{F}_\Sigma$ , but that will generally be quite a bit smaller than  $\mathbf{F}_X$ .

Now the factor matrix  $(\mathbf{F}_\Sigma \otimes \mathbf{F}_X)$  can be quite a large matrix. For instance, a 6 variable, 12 lag (+ constant) VAR has  $M = 6$ ,  $k = 6 \times 12 + 1 = 73$ , so this will be a  $438 \times 438$  matrix. Fortunately, it has so much structure that we don’t have to generate the complete matrix. The function `%RANMVKRON(FSIGMA,FX)` generates a draw from a multivariate Normal with mean 0 and covariance matrix factored as  $\mathbf{F}_\Sigma \otimes \mathbf{F}_X$ , producing the output in the form of a  $k \times M$  matrix (the “de-vec’ed” version of the usual  $kM$  vector).

There are several ways to compute this type of a multivariate regression with RATS. For a VAR, it's usually best to use the **SYSTEM** definitions and **ESTIMATE** instruction. (The others are described at the end of the section). In our example, we're doing a two variable, eight lag + constant, system:

```
system(model=varmodel)
var gnpadjust uradjust
lags 1 to 8
det constant
end(system)
*
estimate
```

This defines the model called `VARMODEL`, consisting of two equations with dependent variables `gnpadjust` and `uradjust`. Each of the equations has eight lags of each of the variables + the constant. Not that it matters here, but each equation has the same structure: lags 1 to 8 of `gnpadjust` are the first eight coefficients, lags 1 to 8 of `uradjust` come next and the `CONSTANT` is last.

The `MODEL` data type has a large set of functions which can be used to move information from and to the model. The two most important of these are `%MODELGETCOEFFS` and `%MODELSETCOEFFS`, which get and set the coefficients for the model as a whole. See the chapter's *Tips and Tricks* (Section 6.7).

The **ESTIMATE** instruction estimates the most recently defined system. It defines the following things that we will need:

- The model itself (the equations and their coefficients)
- The covariance matrix of residuals as `%SIGMA`
- The number of observations as `%NOBS`
- The number of regressors per equation as `%NREG`
- The number of regressors in the full system as `%NREGSYSTEM`
- The (stacked) coefficient vector as `%BETASYS`
- The regression  $(\sum x_t' x_t)^{-1}$  matrix as `%XX`
- The number of equations in the system as `%NVAR`

So how do we get a draw from the posterior for  $\{\Sigma, \beta\}$ ? First, we need to draw  $\Sigma$  from its unconditional distribution. The scale matrix for the Wishart is  $(T \times \Sigma(\hat{B}))^{-1}$ . To use `%RANWISHARTI`, we need a factor of this. We can get that and the degrees of freedom from the variables above with:

```
compute fwish =%decomp(inv(%noobs*%sigma))
compute wishdof=%noobs-%nreg
```

The draw for  $\Sigma$  is then done by

```
compute sigmad =%ranwisharti(fwish,wishdof)
```

Given that, how do we draw a coefficient vector? We need the two factor matrices, for `SIGMAD` and for `%XX`.

```
compute fxx      =%decomp(%xx)
compute fsigma =%decomp(sigmad)
```

The random part of the draw for  $\beta$  can be done with:

```
compute betau      =%ranmvkron(fsigma, fxx)
```

Recall that this will be a  $k \times M$  matrix.

What about the mean? We can turn the stacked vector `%BETASYS` into the same arrangement as `BETAU` using `%VECTORECT(%BETASYS, %NREG)` or (with `RATS 7.2`) we can use `%RESHAPE(%BETASYS, %NREG, %NVAR)`. Since it gives us what we need directly, we'll instead use `%MODELGETCOEFFS`:

```
compute betaols=%modelgetcoeffs(varmodel)
```

The following puts the pieces together to get the draw for the coefficients.

```
compute betadraw=betaols+betau
```

So what do we do with this? In previous examples, we were mainly interested in the distribution of the coefficients of the model; and for systems which aren't VAR's (such as the finance factor models), that might still be the case. However, the coefficients in the VAR aren't particularly interesting directly. It's nearly pointless to publish a table of VAR coefficients: because of the way the coefficients interact with each other, it's almost impossible to tell what their individual values mean for the dynamics of the data. Instead, we use one of several ways of displaying the dynamics in a more understandable fashion: the impulse response function (IRF) or moving average representation (MAR), the forecast error variance decomposition (FEVD) or decomposition of variance, the historical decomposition, and in or out-of-sample forecasts. Those are all functions of either the  $\{\Sigma, \beta\}$  alone (for the IRF and FEVD) or an interaction between  $\{\Sigma, \beta\}$  and the data (for the historical decomposition and forecasts).

The main tool for describing the dynamics of the data is the IRF, which is computed using the instruction **IMPULSE**. This shows the response of the system when the only input is a single defined shock  $\varepsilon_t$ . **IMPULSE** takes the defined model as one of its inputs. However, we need to compute the IRF's with the newly generated coefficients. Thus we need

```
compute %modelsetcoeffs(varmodel, betadraw)
```

to stuff our coefficient draw into the model.

What types of shocks should we analyze? The same basic technique applies to any set that you use. The standard used in the **MONTEVAR.RPF** example program and the **@MONTEVAR** procedure is a full set using the Cholesky factor. For an  $M$ -variable VAR, this defines a set of  $M^2$  responses, one for each of the

$M$  variables to each of  $M$  shocks. This is done over  $H$  periods, so it's quite a bit of information ( $M^2H$  numbers for each draw). You could also work with a more limited set of shocks or with a different definition for the  $M$  shocks than the Cholesky factor. We'll do this in our example, using a Blanchard-Quah factorization.

With that much information, the only good way to convey it is graphically. The standard here would be a collection of graphs with some form of error bands. The simplest type of error band is a  $\pm$  one or two standard error. The sample mean and standard error are computed for each of those  $M^2H$  statistics and you graph the mean and the mean  $\pm$  some chosen number of standard errors. An alternative is to plot the mean together with some percentiles of the empirical density. That works better when the distribution of a response is skewed. Note well that that can take quite a while, since computing those percentiles requires sorting an  $S$ -vector for each of the sets of statistics. If  $S$  is a big number (100,000 for instance), that can easily take much longer than doing all the original number crunching.

Before we move on to a full discussion of the example, we need to introduce another concept which we will use: antithetic acceleration. This is used to improve the efficiency of the simulations: how much accuracy we can achieve for a given number of draws.

## 6.2 Antithetic Acceleration

Antithetic acceleration is a technique which can be used in many types of simulations, not just those used in Bayesian analysis. For instance, it's commonly used in computing option pricing using simulations. The idea is to pair a draw with another draw which has equal probability. By far the most common use of this is flipping the sign on the random component of a Normal. If  $X_i \sim N(\mu, \Sigma)$ , then  $\tilde{X}_i = 2\mu - X_i$  is the same distance on the other side of  $\mu$  from  $X_i$  and has the same density function. If we're trying to approximate  $Eh(X)$  over the Normal density then

$$Eh(X) \approx \sum_{i=1}^S h(X_i)$$

but also

$$Eh(X) \approx \sum_{i=1}^S h(\tilde{X}_i)$$

so

$$Eh(X) \approx \sum_{i=1}^S \frac{h(X_i) + h(\tilde{X}_i)}{2} \tag{6.6}$$

What does this accomplish? Well, if  $h$  is linear, the random parts of  $h(X_i)$  and  $h(\tilde{X}_i)$  cancel, and each term in (6.6) is just  $h(\mu)$ , which is the correct answer.

We've completely eliminated any simulation error. Of course, if we knew  $h$  was linear, we wouldn't have to simulate to get its mean. But if  $h$  is nearly linear, the two random parts *almost* cancel, and we substantially reduce the simulation error.

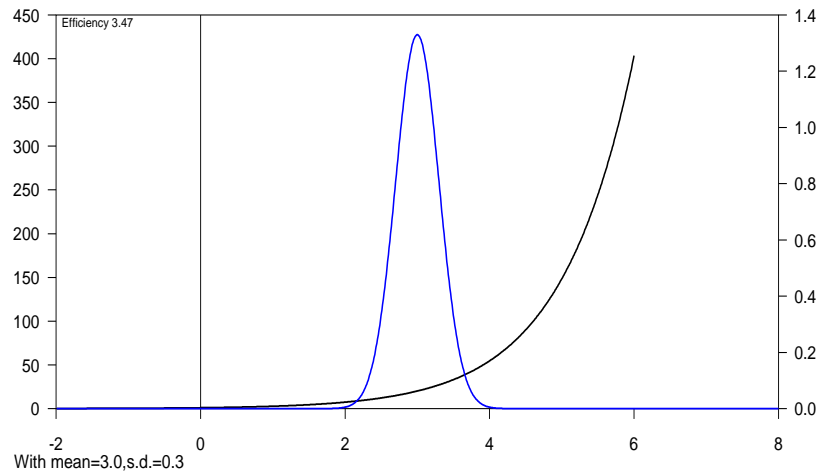
Note that this can only be used when you can do some form of direct simulation; it can't be used with any of the MCMC techniques (Gibbs sampling or M-H) since those require a new draw each step to keep the chain going.

A stylized way of doing this (so as not to break any post-processing code) is

```
do draw=1, ndraws
  if %clock(draw,2)==1 {
    draw = direct draw
  }
  else {
    draw = antithetic draw
  }
  Do bookkeeping with the value of draw
end do draw
```

This works fine if all you're doing is computing means of various functions of the draws. If, however, you want to analyze the numerical standard errors of your estimates, you have to do more. If we had independent draws, we could compute the sample mean and sample variance and use the standard  $s/\sqrt{S}$  as the standard error of the sample mean. With antithetic acceleration, however, the draws aren't independent; there will be  $\text{ndraws}/2$  pairs, where the *pairs* are independent.

We've included a program which does a parallel analysis of computing the mean of  $\exp(x)$  for a Normally distributed  $x$  for independent and antithetic samples. (That has a log-normal distribution, so we can compute the mean analytically for comparison). When the standard deviation of the Normal is small relative to the mean, the function is nearly linear over the likely values of  $x$ , so antithetic acceleration works quite well. If it's much more spread out, the function has much more curve to it, and, while antithetic acceleration still helps, it doesn't accomplish as much. The ratio of the numerical standard error for the independent draws to that for the antithetic draws (with the same number of evaluations) gives a measure of the efficiency gain. With the settings on the file, that comes out in the 3.xx range. (We could compute it exactly, but that's a simulated value). Since the numerical standard error goes down with the square root of the number of draws, that means that for this case, using the antithetic method gives us the equivalent reduction to multiplying the number of draws by roughly 10 (efficiency squared).



### 6.3 An Application: Blanchard-Quah Model

The application is from the well-known paper Blanchard & Quah (1989). This is a two equation model, with real GNP growth and unemployment as the two dependent variables, data from 1948:2 to 1987:4 (input data are from 1948:1, but one data point is lost in computing the growth rate). Allowing for eight lags, the estimation range is from 1950:2 to the end of the sample.

Rather than use a standard Cholesky factor, BQ use a factorization which imposes a long-run restriction on the response of one of the two shocks. Their “demand” shock is defined by making the long-run response of real GNP equal to zero. This is the running sum of the response of real GNP growth. That, combined with the shock being part of a factor of the covariance matrix, is enough to identify it. The “supply” shock is whatever finishes off a factorization.

In their paper, BQ use bootstrapping to get the error bands, but apparently did that incorrectly, see Sims & Zha (1999). We’ll do the same analysis, but with a correct Bayesian calculation.

RATS has a `%BQFACTOR` function which computes the BQ factorization given an input covariance matrix and *sum of lag coefficients*. If the VAR is written in the form

$$\mathbf{A}(L)y_t = \varepsilon_t \quad (6.7)$$

the  $M \times M$  matrix  $\mathbf{A}(1)$  is needed for computing the long-run responses. When we draw a new covariance matrix, the factor (and the shocks) change, and when we draw a new set of coefficients, the sum of lag coefficients changes. The **ESTIMATE** instruction produces the matrix `%VARLAGSUMS`, but that’s only the value at the original estimates. We’ll need to use the `%MODELLAGSUMS` function, which recomputes that for the current set of coefficients. To get the correct result, we need to stuff our draw into the model first, then recompute the lag sums.

In the paper, both GNP growth and the unemployment rate are adjusted before being used: GNP growth by pulling separate means out of high and low growth periods and the unemployment rate by detrending using regression, so there is quite a bit of initial code in the program to produce the inputs to the VAR. Once that's ready, we set up and run the VAR with:<sup>1</sup>

```
system(model=varmodel)
var gnpadjust uradjust
lags 1 to 8
det constant
end(system)
*
estimate
```

We can compute the Blanchard-Quah factor using

```
compute factor=%bqfactor(%sigma,%varlagsums)
{
if factor(1,2)<0.0
    compute factor=factor*%diag(||1.0,-1.0||)
}
```

The **IF** is to deal with the sign convention on the demand shock. Specifying that a shock has a zero long run response (or a zero impact response) tells us the “shape”, but not the sign. If a shock satisfies the constraint, it will also if its sign is flipped. The sign convention chosen by %BQFACTOR is buried deep in the calculation, but in most cases, the demand shock has a positive impact response on the 2<sup>nd</sup> variable. We want the impact response to be positive for GNP, not unemployment, so if the sign is wrong on GNP, we need to flip the whole column. (Sign flipping a column in a factor keeps it as a factor).

We compute the initial set of impulse responses using:

```
impulse(model=varmodel, factor=factor, steps=100, $
    results=impulses, noprint)
```

The results of the **IMPULSE** instruction are being saved in the RECT[SERIES] named **IMPULSES**. **IMPULSES(1,1)** is the series of 100 periods of response of the first series (GNPADJUST) to the first shock (“supply” comes first the way %BQFACTOR works). **IMPULSES(1,2)** is the series of responses of GNPADJUST to the second shock (“demand”).

The long-run restriction is that the sum of the responses of GNP growth to the demand shock is zero. The responses of GNP itself can be obtained by applying the **ACCUMULATE** instruction to the responses of GNP growth. We can verify that we did this correctly (a good idea before running thousands of simulations) by doing:

---

<sup>1</sup>The VAR (*Setup/Estimate*) wizard can be used to create this set of instructions.

```
acc impulses(1,2) / gnpdemand
acc impulses(1,1) / gnpsupply
print(nodates) / gnpdemand gnpsupply
```

We can see that the response to the demand shock is nicely converging to zero, while the response to the supply shock isn't.

97	0.000000033585	0.564580238007
98	0.000000025408	0.564580231726
99	0.000000018938	0.564580226622
100	0.000000013830	0.564580222302

The setup code is pretty much the same for any VAR with a flat prior:

```
compute ndraws =10000
compute nvar    =%nvar
compute fxx     =%decomp(%xx)
compute fwish   =%decomp(inv(%nobs*%sigma))
compute wishdof=%nobs-%nreg
compute betaols=%modelgetcoeffs(varmodel)
*
declare vect[rect]  %%responses(ndraws)
declare rect[series] impulses(nvar,nvar)
```

You can obviously change the number of draws. The only other thing that might change here is the dimension of `IMPULSES`: if you only want to analyze only one shock, the second subscript there will be 1 rather than `NVAR`.

This is the standard simulation loop for a VAR with a flat prior using antithetic acceleration. On the odd draws, we draw a new covariance matrix from the inverse Wishart, and compute the random part of the multivariate Normal for the coefficients. As described before, this uses the `%RANMVKRON` to do that efficiently, with the `FXX` matrix computed outside the loop. On the even draws, we use the previous covariance matrix but use the opposite sign on the random part of the multivariate Normal.



```

infobox(action=define,progress,lower=1,upper=ndraws) $
  "Monte Carlo Integration"
do draws=1,ndraws
  if %clock(draws,2)==1 {
    compute sigmad  =%ranwisharti(fwish,wishdof)
    compute fsigma  =%decomp(sigmad)
    compute betau   =%ranmvkron(fsigma,fx)
    compute betadraw=betaols+betau
  }
  else
    compute betadraw=betaols-betau
  compute %modelsetcoeffs(varmodel,betadraw)
  impulse(noprint,model=varmodel,factor=fsigma,$
    results=impulses,steps=nstep)
  *
  * Store the impulse responses
  *
  dim %%responses(draws)(nvar*nvar,nstep)
  local vector ix
  ewise %%responses(draws)(i,j)=ix=%vec(%xt(impulses,j)),ix(i)
  infobox(current=draws)
end do draws
infobox(action=remove)

```

What do we need to change in this? Everything from the top of the loop to the `%MODELSETCOEFFS` is the same regardless of what type of shocks we want. This is set up to do the Cholesky factor, while we want the BQ factor. So we need to replace the `IMPULSE` instruction with:

```

compute factor=%bqfactor(sigmad,%modellagsums(varmodel))
if factor(1,2)<0.0
  compute factor=factor*||1.0,0.0|0.0,-1.0||
  impulse(noprint,model=varmodel,factor=factor,$
    results=impulses,steps=nstep)

```

We compute the BQ factor at the current values for `SIGMAD` and the lag sums, and use that, rather than `FSIGMA`, as the value for the `FACTOR` option.

The other change is that we really want the responses of real GNP, not real GNP *growth*. We already saw above how to do that. To keep the same basic processing code, we do the `ACCUMULATE` instructions, but put the results back into the same series.

```

acc impulses(1,1) 1 nstep
acc impulses(1,2) 1 nstep

```

That's all we need to do to generate the draws. What about the post-processing? Here, we use the procedure `@MCGraphIRF`, which takes the set of responses in

the `%%responses` array and organizes them into graphs.<sup>2</sup>

First, we need better labels for the shocks: we would like to call them “Supply” and “Demand”. We also would rather use better labels than `GNPADJUST` and `URADJUST` for the two dependent variables. The following sets up the labels for the variables (`YLABEL`) and shocks (`XLABEL`):

```
dec vect[strings] xlabel(nvar) ylabel(nvar)
compute ylabel(1)="GNP"
compute ylabel(2)="UR"
compute xlabel(1)="Supply"
compute xlabel(2)="Demand"
```

The graphs are then done with:

```
@mcgraphirf(model=varmodel,percent=|.16,.84|,$
    varlabels=ylabel,shocklabels=xlabel,$
    footer="Impulse Responses with 68% Bands")
```

The `percent=|.16,.84|` gives roughly the one-standard error band, computed “robustly”. If you want wider bands (95% or roughly two standard error bands), change that to `|.025,.975|`. If you want (symmetrical) one or two standard deviation bands, use `stddev=1` or `stddev=2` rather than the percent option.

As noted before, BQ apparently did their bootstrap wrong, which is why their graphs look quite odd (in particular, in some cases the upper bound sits right on top of the response at the mean). Our results replicate (up to simulation error) those in figure 7 of Sims and Zha.

## 6.4 Structural VAR's

We just looked at a simple example of a structural VAR: the Blanchard-Quah model. A structural VAR aims to model the contemporaneous relationship among the residuals for the VAR: some type of model of the form:

$$\Sigma = G(\theta) \tag{6.8}$$

where  $G$  is a function which produces a positive-definite symmetric  $M \times M$  matrix. Because  $\Sigma$  has  $M(M+1)/2$  distinct elements, a well-designed structural VAR can have no more than that number of elements in  $\theta$ . If it has *exactly* that many, it will be just identified; if it has fewer, it will be overidentified. This is generally done as a model for a factor of  $\Sigma$ , that is,  $\Sigma = F(\theta)F(\theta)'$ , or as a model for an orthogonalizer of  $\Sigma$ :  $F(\theta)\Sigma F(\theta)' = I$ . The first of these describes the impact behavior of the (orthogonal) shocks, while the second describes how the shocks are constructed from the variables.

---

<sup>2</sup>We used the name `%%responses` and the particular way of saving the IRF in order to use this procedure.

This means that the  $\mathbf{H}$  in the VAR is a function of the underlying parameters  $\theta$ , that is,

$$\mathbf{H} = [\mathbf{G}(\theta)]^{-1}$$

The likelihood element is now:

$$p(y_t|X_t, \beta, \theta) \propto |\mathbf{G}(\theta)|^{-1/2} \times \exp\left(-\frac{1}{2}(y_t - (I_M \otimes x_t) \beta)' [\mathbf{G}(\theta)]^{-1} (y_t - (I_M \otimes x_t) \beta)\right)$$

If we track through all the calculations isolating  $\beta$  in Section 6.1, we see that conditioned on  $\theta$ , the posterior for  $\beta$  will still be Normal:

$$\beta|y, X, \theta \sim N\left(\hat{\beta}, [\mathbf{G}(\theta)]^{-1} \otimes \sum x_t' x_t\right)$$

Continuing, we see that, as before, this gives us:

$$p(y|X, \mathbf{B}, \theta) \propto |\mathbf{G}(\theta)|^{-(T-k)/2} \exp\left(-\frac{1}{2} \text{trace} [\mathbf{G}(\theta)]^{-1} (T \times \Sigma(\hat{\mathbf{B}}))\right) \times p(\beta|y, \theta) \quad (6.9)$$

The maximum likelihood estimates for  $\theta$  can be done by computing the OLS estimates  $\hat{\beta}$  and then maximizing:

$$|\mathbf{G}(\theta)|^{-T/2} \exp\left(-\frac{1}{2} \text{trace} [\mathbf{G}(\theta)]^{-1} (T \times \Sigma(\hat{\mathbf{B}}))\right) \quad (6.10)$$

The  $k$  in  $|\mathbf{G}(\theta)|^{-(T-k)/2}$  in (6.9) doesn't show up in (6.10) because (6.10) is the result of maximizing (concentrating)  $\beta$  out, while (6.9) results from *integrating*  $\beta$  out. If this is a parametric model, you can maximize or evaluate either form using the instruction **CVMODEL**, which takes as input  $\Sigma(\hat{\mathbf{B}})$  (and  $T$ ), which are the only statistics from the data which are necessary to compute (6.10).

There are some situations in which it's not obvious how to write the structural model in the form (6.8). The BQ model is such a situation. It constructs a factor of  $\Sigma$ , but it does so based upon how the shocks behave. Models like this are almost always exactly identified, which makes inference much easier. We can do exactly what we did in the previous example: draw an unrestricted  $\mathbf{H}$  (or  $\Sigma$ ) and  $\mathbf{B}$  and compute the needed factor based upon that. Note that the BQ model (and any other similar model with long-run restrictions) has a  $\mathbf{G}$  function which depends not just upon  $\theta$ , but upon  $\mathbf{B}$  (through the lag sums). For a just-identified model, that's OK. For an overidentified model, it's not. As soon as you need to do a combined inference on  $\mathbf{B}$  and  $\theta$ , you're in a very different situation, which is much more computationally intensive.

If we have a parametric  $\mathbf{G}(\theta)$  and the number of equations is four or greater, we'll probably have an overidentified model; it's just hard to write down any interesting structural model with four equations which really distinguishes the shocks from each other and which has as many as six non-zero parameters

(four parameters will be, in effect, scaling parameters). There's nothing wrong with that: the whole point is that you want an economic interpretation for your results, not a mechanical procedure like a Cholesky factor. It does, however, substantially complicate the Bayesian analysis. And this is one of those cases where standard likelihood analysis can produce rather poor results. The likelihood function (6.10) is notoriously badly behaved, so maximizing it might 1) not get you a global maximum or 2) not give you a good idea of the shape of the likelihood from the standard approximation at the maximizer.

A number of early papers attempted to handle overidentified structural models in the “obvious” way, which was to just adapt the standard VAR Monte Carlo, drawing unrestricted  $\Sigma$ , then maximizing (6.10) at the drawn  $\Sigma$  to get  $\theta$ . It's not the worst idea ever, and probably gives results which aren't too far off, but it's not a legitimate procedure. In general, you can't draw from a restricted distribution by drawing from the unrestricted distribution then restricting down; that gives you a density across the correct space, but it's not the same as the restricted density itself.

If we look at (6.9), we see that we can evaluate the likelihood for any value of  $\theta$ , and conveniently, we don't need to do an  $O(T)$  calculation to do that: we just need  $T \times \Sigma(\hat{\mathbf{B}})$ . **CVMODEL** with the option `METHOD=EVAL` can handle that. It returns the log posterior density in `%FUNCVAL`, though we'll actually need a few extra options in order to compute the marginal density from (6.9) rather than the concentrated log likelihood in (6.10).

The parametric structural models can fairly generally be written in the form

$$\mathbf{A}(\theta)u_t = \mathbf{B}(\theta)v_t$$

where the  $u_t$  are the VAR residuals ( $\varepsilon_t$  in the notation that we've been using) and  $v_t$  are the (orthogonal) structural shocks. (Generally, either  $\mathbf{A}$  or  $\mathbf{B}$  is just the identity, but we'll keep this as general as possible for now). Letting  $Ev_t v_t' = \Lambda(\theta)$ , a diagonal matrix, then, if we define  $\mathbf{C}(\theta) = \mathbf{B}(\theta)^{-1}\mathbf{A}(\theta)$ , we have

$$\mathbf{G}(\theta) = \mathbf{C}(\theta)^{-1}\Lambda(\theta)\mathbf{C}(\theta)'^{-1}$$

so

$$|\mathbf{G}(\theta)| = |\mathbf{C}(\theta)|^{-2} |\Lambda(\theta)| \quad (6.11)$$

and

$$\begin{aligned} \text{trace} [\mathbf{G}(\theta)]^{-1} (T \times \Sigma(\hat{\mathbf{B}})) &= \text{trace} \mathbf{C}(\theta)' \Lambda(\theta)^{-1} \mathbf{C}(\theta) (T \times \Sigma(\hat{\mathbf{B}})) \\ &= \text{trace} \Lambda(\theta)^{-1} \mathbf{C}(\theta) (T \times \Sigma(\hat{\mathbf{B}})) \mathbf{C}(\theta)' \end{aligned} \quad (6.12)$$

It's probably not unreasonable to assume that the  $M$  diagonal elements of  $\Lambda(\theta)$  are just  $M$  separate parameters that don't directly involve the parameters governing the  $\mathbf{A}$  and  $\mathbf{B}$  matrices. We'll partition the full parameter set into what we'll just call  $\Lambda$  (understanding that the parameters are the diagonal elements)

and just refer to the parameters in the  $\mathbf{A}$  and  $\mathbf{B}$  matrices. Using (6.11) and (6.12), the posterior for  $\{\Lambda, \theta\}$  can be written

$$p(\Lambda, \theta|y) \propto |\Lambda|^{-(T-k)/2} \exp\left(-\frac{1}{2}\text{trace}\Lambda^{-1} \times \mathbf{C}(\theta)(T \times \Sigma(\hat{\mathbf{B}}))\mathbf{C}(\theta)'\right) \\ \times |\mathbf{C}(\theta)|^{T-k} \times p(\Lambda, \theta)$$

where  $p(\Lambda, \theta)$  is the (as yet) unspecified prior. Since  $\Lambda$  is diagonal,

$$|\Lambda| = \prod_{i=1}^N \lambda_{ii} \quad \text{and} \quad \text{trace}\Lambda^{-1}\mathbf{V} = \sum_{i=1}^N \lambda_{ii}^{-1}\mathbf{V}_{ii}$$

for any matrix  $\mathbf{V}$ . Let the prior be the rather uninformative

$$p(\Lambda, \theta) = |\Lambda|^{-\delta}$$

then defining  $\mathbf{V}(\theta) = \mathbf{C}(\theta)(T \times \Sigma(\hat{\mathbf{B}}))\mathbf{C}(\theta)'$ , the posterior for  $\Lambda$  conditional on  $\theta$  is

$$p(\Lambda|y, \theta) \propto \prod_{i=1}^N \left( \lambda_{ii}^{-(T-k)/2-\delta} \exp\left(-\frac{1}{2}\lambda_{ii}^{-1}\mathbf{V}(\theta)_{ii}\right) \right) \quad (6.13)$$

By inspection, each  $\lambda_{ii}^{-1}$  (the precision of structural shock  $i$ ) is distributed as a gamma with shape parameter  $\frac{T-k}{2} + \delta + 1$  (degrees of freedom  $T-k+2\delta+2$ ) and scale parameter  $1/2\mathbf{V}_{ii}$ . Aside from the somewhat odd degrees of freedom, this makes quite a bit of sense. Under the model,  $\mathbf{C}(\theta)$  is supposed to orthogonalize  $\Sigma$ , so  $\mathbf{C}(\theta)\Sigma(\hat{\mathbf{B}})\mathbf{C}(\theta)'$  should be nearly a diagonal matrix. The diagonal elements of that are used to “center” the distribution for the draws for the  $\lambda_{ii}$ .

We can now integrate out the  $\Lambda$  to get the posterior for  $\theta$ . The  $\lambda_{ii}$  themselves are inverse gammas, so the integral is the reciprocal of the integrating constant from Appendix B.8. We can ignore the  $\Gamma(a)$ , since it doesn't depend upon  $\theta$  and we need to re-introduce the  $|\mathbf{C}(\theta)|^{T-k}$ , since it *does*, to get

$$p(\theta|y) \propto |\mathbf{C}(\theta)|^{T-k} \exp\left(-\frac{T-k+2\delta+2}{2} \sum_{i=1}^N \log(\mathbf{V}(\theta)_{ii})\right) \quad (6.14)$$

This is non-standard but readily computable. In fact, it differs from the *concentrated likelihood function* only by the  $2\delta+2$  inside the “exp”.

To evaluate (the log of) (6.14) with **CVMODEL**, you have to

1. Create the parameter set  $\theta$
2. Define the  $\mathbf{A}$  and/or  $\mathbf{B}$  functions. This is generally done with a `FRML [RECT]` that lists explicitly the form.
3. Estimate the VAR so you have the  $T$  and  $\Sigma(\hat{\mathbf{B}})$
4. Choose  $\delta$  for the prior. The standard value here is  $.5(M+1)$

As in the example with non-linear least squares, it will be convenient to define a named parameter set (`PARMSET`) so we can “poke” a test set of values into it for evaluating (6.14).

**CVMODEL** has three ways for handling the scale factors (the  $\Lambda$ ). The one described above is `DMATRIX=MARGINALIZED`. The  $\delta$  is input using the `PDF` option, and the  $k$  with the `DFC` option.

There are three methods for doing inference on  $\theta$ — two of which we’ve already seen, one which is new.

1. Random walk M-H
2. Independence chain M-H
3. Importance sampling

Importance sampling is discussed in Koop in section 4.3.3 and in the *RATS User’s Guide* in section 16.6. It’s very similar to independence chain M-H. The difference is that independence chain M-H overweights “desirable”  $\theta$  (ones where the posterior is high relative to the probability that we draw it) by staying on them for multiple sweeps, while importance sampling keeps all draws but assigns weights to them. Importance sampling has the advantage that the statistical properties are easier to determine since they’re just weighted independent random variables, while M-H draws are correlated. However, it can only be used if there isn’t a need to do Gibbs sampling. Here, we don’t need that since we have a marginal density for  $\theta$  and a conditional density for  $\Lambda|\theta$ . If we had only  $\theta|\Lambda$  and  $\Lambda|\theta$ , we would need to do Gibbs sampling and couldn’t use importance sampling.

If you can do importance sampling, you should prefer it over independence chain M-H. There may, however, be situations in which random walk M-H is preferred over importance sampling. This is when it’s hard to come up with a good importance function (the direct sampling density for  $\theta$ ). If there are regions where the posterior is high and the probability of hitting them with a draw is low, such points would get very high weights, giving you an weighted average that might be dominated by just a few points. There’s a simple measure of whether you have a problem like this:

$$\left( \sum_{i=1}^S w_i \right)^2 / \sum_{i=1}^S w_i^2$$

should ideally be a high percentage of the number of draws. If it’s a small percentage of that (below 10%), your sampler is probably not working very well. Correcting this might require fattening the tails on your candidate density. If no change like that works, then you’ll have to switch methods.

Because importance sampling creates records of draws that need weighting, you need to do slightly different calculations in the post processing. For instance, the **DENSITY** instruction requires a `WEIGHT` option, as does **STATISTICS**.

The model (and data set) are the ones used in the `CVMODEL.RPF` example from Section 7.6 of the *RATS User's Guide*.

$$\begin{bmatrix} u_u \\ u_c \\ u_r \\ u_x \\ u_m \\ u_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & \beta_{uf1} & 0 & 0 & 0 \\ \beta_{cr1} & 1 & \beta_{cf1} & 0 & 0 & 0 \\ 0 & 0 & 1 & \beta_{rf2} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \beta_{mr1} & 0 & 0 & 0 & 1 & \beta_{mn1} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{r1} \\ v_{r2} \\ v_{f1} \\ v_{f2} \\ v_{n1} \\ v_{n2} \end{bmatrix}$$

The  $u$  are the residuals and the  $v$  are the structural shocks. The dependent variables are (in order) USA GDP, Canadian GDP, Canadian interest rate, US Canada exchange rate, Canadian money stock and price level. Everything but the interest rate is in logs. The structural shocks are two “real” shocks, two “financial” shocks and two “nominal” shocks. In addition to the six parameters that appear in the matrix above, there are six scales: the variances of the six  $v$  shocks.

The basic set up is the same for all three, after reading the data, setting up and estimating the OLS VAR:

```
dec frml[rect] bfrml
nonlin(parmset=svar) uf1 cr1 cf1 rf2 mf1 mm2
frml bfrml = ||1.0,0.0,uf1,0.0,0.0,0.0|$
              cr1,1.0,cf1,0.0,0.0,0.0|$
              0.0,0.0,1.0,rf2,0.0,0.0|$
              0.0,0.0,0.0,1.0,0.0,0.0|$
              mf1,0.0,0.0,0.0,1.0,mm2|$
              0.0,0.0,0.0,0.0,0.0,1.0||
compute uf1=cr1=cf1=rf2=mf1=mm2=pm2=0.0
compute delta=3.5
cvmodel(b=bfrml,parmset=svar,dfc=ncoef,pdf=delta,$
        dmatrix=marginalized,method=bfgs) %sigma
compute nfree=%nreg
```

This model has a “B”, but no “A”. You define the `FRML[RECT]` that evaluates to the  $6 \times 6$  matrix. (You have to declare the target type for `FRML` if it’s not just a standard real-valued function). To get a set of starting (or central) values for the simulations, we first maximize the posterior density. We’ll use the estimated parameters for the center and base the covariance matrix of the candidate density on the estimated covariance matrix from this. For a model of this form, you compute the  $V$  matrix needed for drawing the  $\Lambda$  by:

```
compute b=bfrml(1)
compute vdiag =%mqformdiag(%sigma,inv(b))
```

`%MQFORMDIAG(%sigma,inv(b))` computes the diagonal (only) of  $B'^{-1}\Sigma B^{-1}$ .

### 6.4.1 Waggoner-Zha Sampler

Waggoner & Zha (2003) proposes a (very technical) Gibbs sampling procedure which applies specifically to the  $\mathbf{A}$ -form structural VAR model:

$$\mathbf{A}(\theta)\mathbf{u}_t = \mathbf{v}_t, \mathbf{v}_t \sim N(0, \mathbf{I})$$

where the  $\mathbf{A}$  matrix has restrictions that do not cross equations. Their sampler draws each row of  $\mathbf{A}$  separately. The advantage that this has over importance sampling or Metropolis-Hastings is that it doesn't require experimentation with "tuning" parameters—it's possible to draw the free coefficients in each row directly (conditional on the others).

Unlike the discussion earlier, this does *not* assume a normalization. This has both good and bad points. The good point is that it can't create a problem by choosing a normalization on a coefficient whose sign isn't determined. The bad point is that interpreting a structural VAR *without* a normalization can be difficult—does it make sense for a "price" shock to be of undetermined sign or for a "money demand" shock to not hit money?

With  $\Lambda(\theta)$  pegged to be the identity (rather than free parameters), the likelihood is proportional to

$$|\mathbf{A}(\theta)|^{-T} \exp \left( -\frac{T}{2} \text{trace } \mathbf{A}(\theta) \Sigma(\hat{\mathbf{B}}) \mathbf{A}(\theta)' \right)$$

Waggoner and Zha use a model which is a transpose of ours, so their rows in the  $\mathbf{A}$  matrix are our columns. The order of rows (or columns) in a structural VAR is arbitrary, so we will assume that we are now trying to draw the coefficients of the final row given the top rows. Write the full bottom row of  $\mathbf{A}$  in the form  $\beta' \mathbf{U}$  where  $\beta$  are  $q$  free parameters and  $(\mathbf{U})$  is a  $q \times n$  matrix. For example, if the bottom row in a five variable system is

$$\theta_{51} \quad 0 \quad 0 \quad \theta_{54} \quad \theta_{55}$$

then

$$\beta' = \begin{bmatrix} \theta_{51} & \theta_{54} & \theta_{55} \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Because there are no cross equation restrictions, the first  $n - 1$  rows of

$$\text{trace } \mathbf{A}(\theta) \Sigma(\hat{\mathbf{B}}) \mathbf{A}(\theta)'$$

don't interact with the bottom row, so it is (conditional on the first  $n - 1$  rows) a constant plus  $\beta' \mathbf{U} \Sigma(\hat{\mathbf{B}}) \mathbf{U}' \beta$ . The conditional likelihood can be simplified to

$$|\mathbf{A}(\theta)|^{-T} \exp \left( -\frac{T}{2} \beta' \mathbf{U} \Sigma(\hat{\mathbf{B}}) \mathbf{U}' \beta \right)$$



The  $QR$  decomposition can be used to write:<sup>3</sup>

$$\mathbf{A}(\theta)' = \mathbf{Q}\mathbf{R}$$

Since the first  $n - 1$  columns of  $\mathbf{A}'$  are treated as fixed, the entire  $\mathbf{Q}$  matrix is independent of the final column of  $\mathbf{A}'$  (last row of  $\mathbf{A}$ ) as are the first  $n - 1$  columns of  $\mathbf{R}$ —the only change with the parameters of the final row of  $\mathbf{A}$  will be to the final column of  $\mathbf{R}$ . Since  $\mathbf{Q}$  is orthonormal and  $\mathbf{R}$  is upper triangular,

$$|\mathbf{A}'| = \prod_i r_{ii}$$

However, as noted, the first  $n - 1$  values of  $r_{ii}$  will be considered fixed for the conditional analysis, so the likelihood simplifies further to

$$r_{nn}^{-T} \exp \left( -\frac{T}{2} \beta' \mathbf{U} \Sigma(\hat{\mathbf{B}}) \mathbf{U}' \beta \right)$$

If we use  $\tilde{\mathbf{A}}$  to denote the conditional first  $n - 1$  rows of  $\mathbf{A}$ , then

$$\mathbf{A}(\theta)' = \begin{bmatrix} \tilde{\mathbf{A}}' & \mathbf{U}'\beta \end{bmatrix} \Rightarrow \mathbf{R} = \begin{bmatrix} \mathbf{Q}'\tilde{\mathbf{A}}' & \mathbf{Q}'\mathbf{U}'\beta \end{bmatrix}$$

so  $r_{nn}$  is the element  $n$  of  $\mathbf{Q}'\mathbf{U}'\beta$ . If we take the Cholesky factor  $\mathbf{S}$  so that

$$\mathbf{S}\mathbf{S}' = \left( \mathbf{U} \Sigma(\hat{\mathbf{B}}) \mathbf{U}' \right)$$

then we can simplify the exponent by defining  $\tilde{\beta} = \mathbf{S}'\beta$ , so we get the likelihood proportional to:

$$\left( \left( \mathbf{Q}'\mathbf{U}'\mathbf{S}'^{-1}\tilde{\beta} \right)_n \right)^{-T} \exp \left( -\frac{T}{2} \tilde{\beta}'\tilde{\beta} \right)$$

The final simplification requires isolating a single parameter in what remains of the determinant factor. We can choose an orthonormal matrix  $\tilde{\mathbf{Q}}$  so that

$$\left( \mathbf{Q}'\mathbf{U}'\mathbf{S}'^{-1}\tilde{\mathbf{Q}}' \right)$$

has a final row of  $[0, \dots, 1]$ . If we then define  $\tilde{\tilde{\beta}} = \mathbf{Q}\tilde{\beta}$ , then we end up with

$$\left( \tilde{\tilde{\beta}}_n \right)^{-T} \exp \left( -\frac{T}{2} \tilde{\tilde{\beta}}'\tilde{\tilde{\beta}} \right)$$

As described in WZ, this has the first  $q - 1$  components as independent  $N(0, 1/T)$  variates and component  $q$  as the (randomly signed) square root of a gamma.

---

<sup>3</sup>Note that this is transposed so we can use the standard  $QR$ .

## 6.5 Informative Prior for Univariate Autoregressions

From the start, the “flat prior” VAR’s introduced in *Macroeconomics and Reality* have been known to forecast poorly. If they don’t forecast well, how are they useful? Economists studying the relationships among a set of variables have a different loss function than people wishing to forecast. For forecasters, that is usually some function like the MSE, perhaps for just a single series, perhaps some weighted combination of them. Whether explicit or implicit, it’s well-established that unbiased estimators are generally not best at minimizing such loss functions. For instance, if  $\hat{y}$  is an unbiased forecast of  $y$ , consider the forecast formed by taking a weighted average of  $\hat{y}$  and a constant (any constant)  $y_0$ :  $(1 - \lambda)\hat{y} + \lambda y_0$ . Then the MSE (treating  $y$  as fixed) of using this is

$$\begin{aligned} E(y - (1 - \lambda)\hat{y} - \lambda y_0)^2 &= E((1 - \lambda)(y - \hat{y}) + \lambda(y - y_0))^2 \\ &= (1 - \lambda)^2 \text{var}(\hat{y}) + \lambda^2(y - y_0)^2 \end{aligned}$$

The derivative of this at  $\lambda = 0$  is  $-2 \text{var}(\hat{y})$ , thus, for any  $y_0$ , there is some  $\lambda > 0$  such that the MSE is lower by “shrinking” the unbiased estimator towards  $y_0$ . Note that this doesn’t provide us with a workable estimator, since the “optimal” value of  $\lambda$  depends upon the true (and unknown) value of  $y$ . The “Stein” estimators for linear models with three or more regressors do provide an operational estimator that improves on the MSE of least squares. However, the point is that unbiasedness is not necessarily a desirable property for a forecasting procedure. In the case of the multivariate dynamic model, it turns out that forecasts are generally made better by shutting down much of the cross variable interaction. Needless to say, this isn’t very useful if what we’re interested in studying is precisely those cross dynamics. Thus, the VAR literature has been split almost from the start between loosely or unconstrained estimators for analyzing dynamic interactions, and biased (Bayesian) VAR’s (BVAR’s) for forecasting.

The standard prior for BVAR’s is known as the *Minnesota prior*, having been developed at the University of Minnesota and the Federal Reserve Bank of Minneapolis. While there have been slight changes and enhancements made to this over the years, most priors start with this type of structure.

If we look at a univariate autoregression

$$y_t = \alpha_1 y_{t-1} + \dots + \alpha_p y_{t-p} + \varepsilon_t \quad (6.15)$$

the scale of the  $\alpha$  coefficients is fairly clear – if the sum of the coefficients is bigger than 1 (even only slightly so), the series will be explosive, which is highly unlikely. And if the  $y$  process is rescaled, the coefficients won’t change. A common approach to fitting something like (6.15) by non-Bayesian methods is to choose  $p$  to minimize one of the information criteria, such as Akaike, Schwarz or Hannan-Quinn. The idea behind all of these is that we make the longer lags equal to zero unless the increase in the likelihood function from including

them exceeds the “cost” assigned to extra parameters imposed by the chosen criterion.

A Bayesian alternative to this is to use a longer set of lags, but to use a prior with mean of 0 and variance which decreases with lag. For the typical economic series, done in levels (for instance, interest rates) or in logs (for GDP, money, prices, exchange rates), the series is likely to have something close to a unit root, so the prior generally centers around  $\alpha_1 = 1$ , with the other lags having a mean of zero. In the Minnesota prior, the priors on the individual coefficients are independent Normal, with standard deviation taking the form (for lag  $l$ ):

$$\frac{\gamma}{l^d}$$

$\gamma$  and  $d$  are hyperparameters (or metaparameters): the parameters that govern the prior. We may want to experiment with different values for these. However, unlike the other situations that we’ve seen, we aren’t interested in looking at the posterior density for these: the posterior density would almost certainly be highest with a fairly large value of  $\gamma$ , since that would give us the OLS estimates, which are the likelihood maximizers (when we condition on the pre-sample lags of the data).

If we’re trying to build a forecasting model, a better approach would be to see how well the model seems to forecast out-of-sample with a given set of hyperparameters. We hold back part of the data to use for evaluating forecast performance, estimate with the earlier data and see how performance is affected by changes in the parameters.

Given the hyperparameters and data, we get a posterior density for the coefficients of the autoregression:

$$p(\alpha|\gamma, d, y)$$

That might be something that we can derive analytically, or it might require simulation by Gibbs sampling. We define some loss function  $l(\alpha)$  on the  $\alpha$ , typically something like the sum of squared forecast errors over the held-back part of the data. Ideally, we would choose  $\{\gamma, d\}$  to minimize

$$R(\gamma, d) = E[l(\alpha)] = \int l(\alpha)p(\alpha|\gamma, d, y)d\alpha \quad (6.16)$$

Unfortunately, that’s not an easy problem to optimize since even when  $p(\alpha|\gamma, d, y)$  is analytical, the distribution of multi-step forecasts derived from them isn’t. The expected loss can be approximated by simulation, but optimizing any function whose value is simulated is rather tricky. (For instance, if you don’t seed the random number generator, you will get different values for two calculations at the same arguments). It can also be quite time-consuming to do each function evaluation. In practice, one or both of the following steps is taken to reduce the computational burden:

1. Instead of trying to compute  $E[l(\alpha)]$ , compute  $l(E[\alpha])$ . This is a huge time saver when  $p(\alpha|\gamma, d, y)$  is analytical, but doesn’t help much when it isn’t.

2. Use a coarse grid of  $\{\gamma, d\}$  values, maybe just looking at a few fairly standard settings.

To get a feel for how this type of prior affects the lag coefficients, let's look at one of the sample data sets that comes with RATS. The example file **CANMODEL.RPF** uses the data set **OECDSAMPLE.RAT**. One of the series on that is a Canadian short interest rate. If we run an OLS autoregression on constant and 12 lags (almost certainly way too many for quarterly data, which is what this is doing), we get the following for the lags:

	Coeff	StdErr	T-Stat
CAN3MTHPCP{1}	1.19	0.11	10.58
CAN3MTHPCP{2}	-0.22	0.17	-1.26
CAN3MTHPCP{3}	0.12	0.17	0.67
CAN3MTHPCP{4}	-0.22	0.17	-1.25
CAN3MTHPCP{5}	-0.06	0.17	-0.34
CAN3MTHPCP{6}	0.32	0.16	1.94
CAN3MTHPCP{7}	-0.37	0.16	-2.22
CAN3MTHPCP{8}	0.21	0.17	1.26
CAN3MTHPCP{9}	-0.01	0.16	-0.04
CAN3MTHPCP{10}	-0.08	0.15	-0.54
CAN3MTHPCP{11}	0.12	0.14	0.89
CAN3MTHPCP{12}	-0.05	0.09	-0.52

Certainly the last year's worth of lags are almost pointless; however, the six and seven lags are fairly large and are either significant, or nearly so, at standard levels. If we do a "general to specific" lag pruning (dropping the final lag as long as it's not significant), we will shorten this to 8 lags. The RATS instruction for doing this is:

```
stwise(method=gtos,slstay=.10) can3mthpcp
# constant can3mthpcp{1 to 12}
```

The minimum AIC is also at 8 lags, while minimum Schwarz is 1 and Hannan-Quinn is 2.<sup>4</sup>

Let's see what the effect is of imposing a fairly "loose" Minnesota prior. This has  $\gamma = .2, d = 0$ ; that is, the prior on each lag coefficient has a standard deviation of .2. We'll use the same basic program as we did before for a linear regression with an independent prior. We set up the prior mean and prior precision on the coefficients with:

<sup>4</sup>If you use the Schwarz criterion with VAR's, get used to seeing 1 as the optimum lag length. An extra lag adds  $M^2$  coefficients; it's hard for the increase in the log likelihood to overcome the high penalty used in Schwarz criterion.

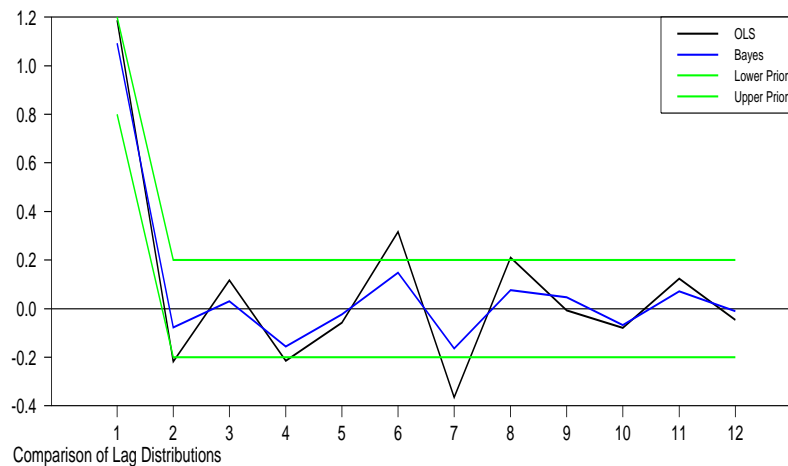
```

compute [vector] bprior=%unitv(nbeta,1)
dec vect hvector(nbeta)
ewise hvector(i)=(1./ .20)^2*%if(i<=12,i^0.0,0.0)
compute [symm] hprior=%diag(hvector)

```

The precision is equal to the reciprocal of the squared standard deviation, which is being put into `hvector`. (The precision on the `CONSTANT` is zero: a flat prior). The precision matrix is created by putting those on the diagonal. The prior mean is the unit vector with its one in the first slot (second argument). If you had a series like excess returns where zero is a more natural choice for the mean, you would just do `%zeros(nbeta,1)` in place of the `%unitv`.<sup>5</sup>

As is the case with almost any model where you can compute one cross product matrix outside the loop and do the calculations with it, the Gibbs sampler is very quick.



This shows the wildly zig-zagging OLS estimates (black line) and the smoother Bayes estimates (blue line). The green lines are the one standard deviation bounds on the prior. Note that the 6 and 7 lags are pulled well inside the one SD bounds. You might think that since the OLS estimates aren't all that unreasonable given the prior (being under two standard deviations from the mean) that they wouldn't be affected this much by such a loose prior. However, the behavior here is fairly common with (vector) autoregressions. Because of the high collinearity, the data have a fairly good idea about the values of sums of (adjacent) coefficients, but less about them individually. For instance, if we look at the sum of lags 6 and 7, we get

```

summarize
# can3mthpcp{6 7}

```

<sup>5</sup>The 1 in the `%zeros` is the second dimension; `%zeros` can create a general  $m \times n$  matrix.

Summary of Linear Combination of Coefficients			
CAN3MTHPCP	Lag(s) 6 to 7		
Value	-0.0493596	t-Statistic	-0.34696
Standard Error	0.1422635	Signif Level	0.7295446

Note that the standard error (.142) on the sum is less than it is on either coefficient individually (.163 and .165). The standard error is even lower on the sum from 6 to 12.

```
summarize
# can3mthpcp{6 to 12}
Summary of Linear Combination of Coefficients
```

CAN3MTHPCP	Lag(s) 6 to 12		
Value	0.15202224	t-Statistic	1.48112
Standard Error	0.10263999	Signif Level	0.1425520

The prior has quite an effect here by providing information that the data are lacking: defining what is a reasonable size of a coefficient. The likelihood function is fairly flat across almost any combination of lags 6 and 7 which add up to roughly zero. The prior favors those which are closer to zero, hence those estimates which maintain roughly the same sum but are quite a bit smaller in absolute value.

An autoregression is a small enough model that we can actually go about minimizing (6.16). We need to create a procedure which “wraps” around the Gibbs sampler, setting up the prior for different values for the two hyperparameters. Once we have a draw for the coefficients:

```
compute %eqnsetcoeffs(rateeq,bdraw)
*
do start=from,to-steps
  forecast(steps=steps,from=start,model=rateeq,results=fcsts)
  sstats start start+steps-1 (fcsts(1)-can3mthpcp)^2>>ssethis
  compute sse+=ssethis
end do start
```

We then do multi-step forecasts starting at each point in the evaluation period and compute the sum of squared forecast errors. Note that `RESULTS=FCSTS` creates a `VECT[SERIES]` since **FORECAST** can be used for models with more than one equation. That’s why we use `fcsts(1)` in the **SSTATS** instruction. `fcsts(1)` is the first (and only, in this case) series in `fcsts`. `ssethis` will be the sum of squared forecast errors across the `steps` forecasts just done. `sse` adds those up across draws.

As mentioned before, you can only do this type of optimization if you control the random number seed. The procedure used here has a **SEED** instruction before starting the Gibbs loop which takes the value from the **SEED** option. Note that a different **SEED** means a different expected value for each setting of the two parameters, which means a different optimum. If things go well, the results won’t be too sensitive to this.

```

nonlin decay tight
compute decay=.25,tight=.05
declare real sse
*
find(trace) min sse
  *
  * Reject out-of-range values. (Set the value to NA and skip the
  * remaining calculation.)
  *
  if decay<0.0.or.tight<=0.0 {
    compute sse=%na
    next
  }
  @ForecastLoss(from=fstart,to=fend,steps=nsteps,$
    tight=tight,decay=decay,seed=21439) sse
end find

```

The optimized value for `TIGHT` ends up being quite small, indicating that for this series, the best forecasting autoregression is basically a random walk.

1.	DECAY	0.2247858542
2.	TIGHT	0.0026886316

## 6.6 VAR with Informative Prior (BVAR)

When applied to a univariate autoregression, the standard deviations in the Minnesota prior take the simple form:

$$\gamma l^{-d}$$

There are just two hyperparameters, and the prior isn't affected by the scale of the variable. A VAR will be more complicated. We need to adjust for the scales of the variables; if, for instance, interest rates are in percentages, the coefficients will be a factor of 100 smaller than if they're in decimals. We also need to incorporate into the prior the notion that "own" lags of the typical variable have most of the explanatory power.

To adjust for scale, the Minnesota prior starts with an (OLS) univariate autoregression (including any deterministic components) on each of the dependent variables. The standard errors of estimate of these regressions (called  $s_i$  for equation  $i$ ) are used to rescale the standard deviation of the prior for a coefficient on a lag of variable  $j$  in equation  $i$  by  $s_i/s_j$ . While somewhat arbitrary, this has stood the test of time. It handles issues of scale correctly (you'll get exactly the same results if you rescale variables), it doesn't add any extra hyperparameters to the structure and it seems to work fine in practice. The univariate autoregressions are used rather than the OLS VAR because it's quite possible that an unrestricted VAR may have almost no (or even negative) degrees of freedom if there are many endogenous variables.

In the RATS manual, the standard error function takes the general form:

$$S(i, j, l) = \frac{\gamma^{l-d} f(i, j) s_i}{s_j}$$

$f(i, j)$  is the relative tightness of variable  $j$  in equation  $i$ . This could be a general  $M \times M$  function, but the standard here is to make

$$f(i, i) = 1; f(i, j) = w \text{ if } i \neq j$$

where  $w$  is a new hyperparameter. The closer this is to zero, the more the lags of “other” variables are shut down. (The means of the priors on all these other variables are zero).

We now have a SUR model with an informative prior. Because the underlying model is linear, we can do a single cross product matrix to get all the statistics required to compute the likelihood. There’s one obstacle standing in the way of working with the model using standard sampling techniques: the possibility of the size overwhelming the capacity of the computer.

With a flat prior, the VAR is easy to handle because the (potentially) huge covariance matrix for the coefficients factors into smaller parts:

$$\Sigma \otimes \left( \sum x'_t x_t \right)^{-1}$$

The number of calculations in a matrix inversion or Cholesky factorization go up with the cube of the matrix size. Reducing an  $Mk \times Mk$  inversion to  $M \times M$  and  $k \times k$  reduces the number of calculations by roughly  $M^3$  (assuming  $k \gg M$ ). For a six variable model, that’s 216. Although the precision matrix coming from the data is still the very nicely formed:

$$\hat{\mathbf{H}} = \mathbf{H} \otimes \sum x'_t x_t$$

the precision matrix from the prior ( $\mathbf{H}$ ) is a general diagonal matrix and we need to compute

$$\left( \hat{\mathbf{H}} + \mathbf{H} \right)^{-1} \tag{6.17}$$

There is no simple way to compute that other than taking the full inverse. In a small enough system, that’s certainly not unreasonable, but it *is* for the much larger VAR’s that are often used in macroeconomic forecasting.

An alternative is to treat  $\mathbf{H}$  as a diagonal matrix. Since the priors are independent across equations, this makes (6.17) a matrix that’s non-zero only on the blocks on the diagonal, so it can be handled equation by equation. We wouldn’t expect that to produce that much of a difference compared with doing the whole system; after all, the maximizing point for the likelihood is OLS equation by equation and the priors are independent. If we’re interested in the model solely for forecasting (not analyzing an SVAR), it’s the lag coefficients that really matter.



Any of the standard forms of Minnesota prior can be implemented in RATS by adding the `SPECIFY` instruction to the system definition. When you estimate the system using **ESTIMATE**, it does the calculation of the  $s_i$  scale factors as described above. Note that this makes the prior somewhat sensitive to the data range over which you estimate the system. The point estimates are computed using what's known as *mixed estimation*. This is basically the natural conjugate prior done somewhat informally. The formal natural conjugate prior for a single equation would have the precision from the likelihood as

$$h \sum x'_t x_t$$

and the precision from the prior as

$$h(\mathbf{H}^*)$$

where  $\mathbf{H}^*$  is the prior precision matrix adjusted for multiplication by  $h$ . In example 2-1, that adjustment was to divide by the prior mean for  $h$ . In mixed estimation, it's usually done by multiplying by the variance of an OLS estimate of the equation (same as dividing by the precision); in RATS, that multiplier is  $s_i^2$ . Note that the precision is the reciprocal of the standard deviation squared, so if we write the precision of variable  $j$  in equation  $i$  at lag  $l$  as

$$H(i, j, l) = \frac{l^{2d} s_j^2}{\gamma^2 f(i, j)^2 s_i^2}$$

then the  $\mathbf{H}^*$  used for mixed estimation just cancels out the  $s_i^2$  in the denominator.

Thirty years ago, when VAR's were first being introduced, there was no serious option for analyzing a BVAR more formally. Even the simple Monte Carlo integration with flat priors could take an hour for 1000 draws. Now, it's possible to handle medium-sized models with full systems estimation. Since the Minnesota prior is still widely used, it would be helpful to be able to strip the setup out of the **ESTIMATE** instruction. There are several procedures which can do that: `@BVARBuildPrior` and `@BVARBuildPriorMN`. `@BVARBuildPriorMN` calls `@BVARBuildPrior` many times to construct the prior. In our example, we build the prior with:

```
@BVARBuildPriorMN(model=varmodel,tight=.1,other=.5) $
  hbpriors hpriors
  @BVARFinishPrior hbpriors hpriors bprior hprior
```

The `@BVARBuild` routines generate the precision and the precision x the mean. (That's the easiest form to generate). `@BVARFinishPrior` converts this information back into precision and mean. Note that `@BVARBuildPrior` can be used to add *dummy observation* elements to a prior.

The example program generates out-of-sample simulations. This allows a full accounting for all the uncertainty in forecasts: both the uncertainty regarding

the coefficients (handled by Gibbs sampling) and the shocks during the forecast period.

```
compute %modelsetcoeffs (varmodel, bdraw)
simulate (model=varmodel, cv=sigmad, results=simresults, steps=nstep)
do i=1, nvar
    set forecast(i) fstart fend = forecast(i)+simresults(i)
    set forestderr(i) fstart fend = forestderr(i)+simresults(i)**2
end do i
```

The **SIMULATE** instruction does the forecasts with randomly drawn shocks. We can't really do this with the equation-at-a-time methods because this depends upon the draw for the covariance matrix. (We can get point forecasts, but not the "cloud"). The `forecast` and `forecastderrs` series accumulate the sum and sum of squares of the forecasts. Outside the loop, these are converted into the mean and standard errors by standard calculations:

```
do i=1, nvar
    set forecast(i) fstart fend = forecast(i)/ndraws
    set forestderr(i) fstart fend = $
        sqrt(forestderr(i)/ndraws-forecast(i)**2)
end do i
```

## 6.7 RATS Tips and Tricks

### The function %CLOCK

`%clock(draw, base)` is like a "mod" function, but gives values from 1 to base, so `%clock(draw, 2)` will be 1 for odd values of `draw` and 2 for even values.

### The %MODEL function family

A **MODEL** is an object which organizes a collection of linear equations or formulas. The family of functions with names beginning with **%MODEL** take information out of and put it into a **MODEL**. These are all included in the *Model* category of the *Functions* wizard.

The most important of these apply to the case where the model is a set of linear equations like a VAR. For those, you can get and reset the coefficients of all equations at one time:

**%modelgetcoeffs(model)** returns a matrix of coefficients, with each column giving the coefficients in one of the equations.

**%modelsetcoeffs(model, b)** resets the coefficients of all equations in the model. The input matrix `b` can either have the same form as the one returned by `%modelgetcoeffs` (one column per equation) or can be a single "vec'ed" coefficient vector, with equation stacked on top of equation.

`%modellagsums(model)` returns the matrix  $A(1)$  for the model written in the form (6.7) using the current values of the coefficients.

`%modellabel(model,n)` returns the label of the dependent variable for the  $n^{th}$  equation in the model. This can be used to construct graph labels (for instance) in a flexible way.

## Estimating Multivariate Regressions

Besides **ESTIMATE**, the other instruction for computing a multivariate regression with identical explanatory variables is **SWEEP**. For the model we're using in this section, you would use:

```
sweep(coeffs=betaols)
# gnpadjust uradjust
# gnpadjust{1 to 8} uradjust{1 to 8} constant
```

However, this produces in `%XX` the full matrix  $\Sigma(\hat{\beta}) \otimes (\sum x_t' x_t)^{-1}$ , not the separate components which we need for Bayesian inference. (**SWEEP** has quite a few other options for doing grouped analysis, which is why it produces the “grand” covariance matrix).

You can also use the function `%SWEEP` applied to a properly set up cross product matrix.

```
cmom
# gnpadjust{1 to 8} uradjust{1 to 8} constant gnpadjust uradjust
```

or (if we've set up the `MODEL` using the **SYSTEM** instructions)

```
cmom(model=varmodel)
```

We'll actually need this type of cross product matrix for other types of work with a VAR; it's the one that we used in the SUR analysis for avoiding computing the residuals at each step – explanatory variables first, then dependent variables. If you do

```
compute s=%sweeptop(%cmom,17)
```

you can extract the following:

$(\sum x_t' x_t)^{-1}$	is the top $17 \times 17$ submatrix: <code>%XSUBMAT(S,1,17,1,17)</code>
$T \times \Sigma(\hat{\beta})$	is the bottom right $2 \times 2$ matrix: <code>%XSUBMAT(S,18,19,18,19)</code>
$\hat{B}$	is the top right $17 \times 2$ corner: <code>%XSUBMAT(S,1,17,18,19)</code>

So the one matrix `S` has all the pieces that we need. The method shown before is more convenient for the specific case of the VAR because we can use the defined `MODEL` type to handle much of the calculation of the functions of interest.

**Example 6.1 Antithetic Acceleration**

```

*
* A small value of sigma (relative to the mean) makes the function
* nearly linear over the range of probable draws, so we would expect a
* high level of efficiency. If it's much larger, the function has much
* more curve to it, and the efficiency goes down.
*
compute mean =3.0
compute sigma=0.3
*
compute ndraws = 10000
*
set hdraw 1 ndraws = 0.0
set hdrawa 1 ndraws = 0.0
*
compute nse =0.0
compute nsea=0.0
*
do draw=1,ndraws
  if %clock(draw,2)==1 {
    *
    * Compute and save the "random" part of the Normal draw
    *
    compute u=%ran(sigma)
    *
    * Take the positive addition
    *
    compute x=mean+u
    compute antix=x
  }
  else {
    *
    * Use the last value of u, and subtract it from the mean
    *
    compute antix=mean-u
    *
    * Computes the sum of squares of the averages of consecutive terms
    *
    compute nsea=nsea+(.5*(exp(antix)+exp(x)))^2
    *
    * This is only necessary because we're doing the comparison with
    * independent sampling.
    *
    compute x=mean+%ran(sigma)
  }
  compute hdrawa(draw)=exp(antix)
*
* These are only necessary for the comparison. These save the
* independent draws and computes the sum of squares of the
* independent terms.
*
compute hdraw(draw)=exp(x)
compute nse=nse+exp(x)^2

```

```

end do draw
*
* Compute the sample means
*
sstats(mean) 1 ndraws hdraw>>mu hdrawa>>mua
*
* Compute the numerical standard errors.
* NSE is for the full set of independent draws.
* NSEA is for the pairs of antithetic draws. Note that the number of
* draws on which this is based is .5*ndraws
*
compute nse =sqrt((nse/ndraws-mu^2)/ndraws)
compute nsea=sqrt((nsea/(.5*ndraws)-mua^2)/(.5*ndraws))
disp "Independent Sample" @20 mu @35 nse
disp "Antithetic Sample" @20 mua @35 nsea
disp "True mean" @20 exp(mean)*exp(sigma^2/2.0)
*
* Compute the relative efficiency
*
compute efficiency=nse/nsea
*
* Draw the function being analyzed (exp(x)) and the density from which
* we're drawing on a single graph, but with different scales since
* there's no real connection between the values of the two. (This is
* done with SCATTER using the OVERLAY option).
*
set xt 1 1000 = -2.0+.01*t
set fx 1 1000 = exp(%logdensity(sigma^2,xt-mean))
set ex 1 1000 = exp(xt)
*
* The SPGRAPH and SPGRAPH(DONE) put a "wrapper" around the graph, so we
* can add the text with the efficiency information to the top left.
*
compute footer=$
    "With mean="+%strval(mean,"*.##")+", s.d.="+%strval(sigma,"*.##")
spgraph
scatter(style=line,overlay=line,omin=0.0,footer=footer) 2
# xt ex 1 800
# xt fx
grtext(position=upleft) "Efficiency "+%strval(efficiency,"*.###")
spgraph(done)

```

**Example 6.2 VAR with flat prior**

```

cal(q) 1948
open data bqdata.xls
data(format=xls,org=cols) 1948:1 1987:4 gnp gd87 lhmur
*
* Generate log real GNP from the gnp and the price deflator
*
set loggnp = log(gnp)
set loggd = log(gd87)
set logrgnp = loggnp-loggd+log(100)
set dlogrgnp = 100.0*(logrgnp-logrgnp{1})
*
* BQ de-mean the gnp growth data with separate values for two subsamples.
*
set dummy1 = t<=1973:4
set dummy2 = t>1973:4
*
linreg dlogrgnp
# dummy1 dummy2
set gnpadjust = %resids
prj removed
*
* The unemployment rate is detrended
*
filter(remove=trend) lhmur / uradjust
*
*****
system(model=varmodel)
var gnpadjust uradjust
lags 1 to 8
det constant
end(system)
*
estimate
*
* This computes the BQ factor at the OLS values. This is constructed so
* the <<second>> shock is the constrained ("demand") one. Since the zero
* restriction doesn't fix the sign, we make sure that the demand shock
* has a positive impact on GNP.
*
compute factor=%bqfactor(%sigma,%varlagsums)
{
if factor(1,2)<0.0
compute factor=factor*%diag(||1.0,-1.0||)
}
*
impulse(model=varmodel,factor=factor,steps=100,results=impulses,noprint)
*
* Check that we have the demand shock defined correctly. The value of
* the response of real GNP to the demand shock should be close to zero
* at long horizons.
*
acc impulses(1,2) / gnpdemand

```

```

acc impulses(1,1) / gnpsupply
print(nodates) / gnpdemand gnpsupply
*****
compute ndraws =10000
compute nstep =40
compute nvar =%nvar
compute fxx =%decomp(%xx)
compute fwish =%decomp(inv(%nobs*%sigma))
compute wishdof=%nobs-%nreg
compute betaols=%modelgetcoeffs(varmodel)
*
declare vect[rect] %%responses(ndraws)
declare rect[series] impulses(nvar,nvar)
*
infobox(action=define,progress,lower=1,upper=ndraws) $
  "Monte Carlo Integration"
do draws = 1,ndraws
  if %clock(draws,2)==1 {
    compute sigmad =%ranwisharti(fwish,wishdof)
    compute fsigma =%decomp(sigmad)
    compute betau =%ranmvkron(fsigma,fxx)
    compute betadraw=betaols+betau
  }
  else
    compute betadraw=betaols-betau
    compute %modelsetcoeffs(varmodel,betadraw)
    compute factor=%bqfactor(sigmad,%modellagsums(varmodel))
    *
    if factor(1,2)<0.0
      compute factor=factor*||1.0,0.0|0.0,-1.0||
    impulse(noprint,model=varmodel,factor=factor,$
      results=impulses,steps=nstep)
    acc impulses(1,1) 1 nstep
    acc impulses(1,2) 1 nstep
    *
    * Store the impulse responses
    *
    dim %%responses(draws) (nvar*nvar,nstep)
    local vector ix
    ewise %%responses(draws) (i,j)=ix=%vec(%xt(impulses,j)),ix(i)
    infobox(current=draws)
  end do draws
infobox(action=remove)
*
dec vect[strings] xlabel(nvar) ylabel(nvar)
compute ylabel(1)="GNP"
compute ylabel(2)="UR"
compute xlabel(1)="Supply"
compute xlabel(2)="Demand"
*
@mcgraphirf(model=varmodel,percent=||.16,.84||,$
  varlabels=ylabel,shocklabels=xlabel,$
  footer="Impulse Responses with 68% Bands")

```

**Example 6.3 Structural VAR: Importance sampling**

```

open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp canexpgdpchs canexpgdpds $
    canmls canusxsr usaexpgdpch
*
set logcangdp = log(canexpgdpchs)
set logcandefl = log(canexpgdpds)
set logcanml = log(canmls)
set logusagdp = log(usaexpgdpch)
set logexrate = log(canusxsr)
*
system(model=canmodel)
variables logusagdp logcangdp can3mthpcp logexrate logcanml logcandefl
lags 1 to 4
det constant
end(system)
*
estimate(noprint)
compute ncoef=%nreg
compute nvar=%nvar
*
dec frml[rect] bfrml
nonlin(parmset=svar) uf1 cr1 cf1 rf2 mf1 mm2
frml bfrml = ||1.0,0.0,uf1,0.0,0.0,0.0|$
           cr1,1.0,cf1,0.0,0.0,0.0|$
           0.0,0.0,1.0,rf2,0.0,0.0|$
           0.0,0.0,0.0,1.0,0.0,0.0|$
           mf1,0.0,0.0,0.0,1.0,mm2|$
           0.0,0.0,0.0,0.0,0.0,1.0||
compute uf1=cr1=cf1=rf2=mf1=mm2=pm2=0.0
*
* Compute the maximum of the log of the marginal posterior density for
* the B coefficients with a prior of the form  $|D|^{**(-\delta)}$ . Delta
* should be at least  $(nvar+1)/2.0$  to ensure an integrable posterior
* distribution.
*
compute delta=3.5
cvmodel(b=bfrml,parmset=svar,dfc=ncoef,pdf=delta,$
    dmatrix=marginalized,method=bfgs) %sigma
compute nfree=%nreg
*
* thetamean is the maximizing vector of coefficients.
*
compute [vector] thetamean=%parmspeek(svar)
*
* fxx is a factor of the (estimated) inverse Hessian at the final
* estimates.
*
compute fxx=%decomp(%xx)
*
* scladjust is used to prevent overflows when computing the weight
* function

```



```

*
compute scladjust=%funcval
*
* nu is the degrees of freedom for the multivariate Student used in
* drawing A's
*
compute nu=10.0
declare vect tdraw(nfree)
*
compute ndraws=10000
*
declare vect lambdadraw(nvar) vdiag(nvar)
*
dec series[vect] timport limport
gset timport 1 ndraws = %zeros(nfree,1)
gset limport 1 ndraws = %zeros(nvar,1)
*
set weights 1 ndraws = 0.0
*
* sumwt and sumwt2 will hold the sum of the weights and the sum of
* squared weights
*
compute sumwt=0.0
compute sumwt2=0.0
infobox(action=define,progress,lower=1,upper=ndraws) $
  "Monte Carlo Integration"
do draw=1,ndraws
  *
  * Draw a new theta centered at thetamean from a multivariate t-density
  *
  compute thetadraw=thetamean+%ranmvt(fxx,nu)
  compute logqdensity=%ranlogkernel()
  *
  * Evaluate the model there
  *
  compute %parmspoke(svar,thetadraw)
  cvmodel(parmset=svar,dfc=ncoef,pdf=delta,dmatrix=marginalized,$
    method=evaluate,b=bfrml) %sigma
  compute logpdensity=%funcval
  *
  * Compute the weight value by exp'ing the difference between the two
  * densities, with scale adjustment term to prevent overflow.
  *
  compute weight =exp(logpdensity-scladjust-logqdensity)
  compute weights(draw)=weight
  compute sumwt =sumwt+weight
  compute sumwt2=sumwt2+weight^2
  *
  * Conditioned on theta, make a draw for lambda.
  *
  compute b=bfrml(1)
  compute vdiag =%mqformdiag(%sigma,inv(b))
  ewise lambdadraw(i)=$
    (%nobs/2.0)*vdiag(i)/%rangamma(.5*(%nobs-ncoef)+delta+1)

```

```

*
* Save the theta's and lambda's
*
compute timport(draw)=thetadraw
compute limport(draw)=lambdadraw
infobox(current=draw)
end do draws
infobox(action=remove)
*
* The efficacy of importance sampling depends upon function being
* estimated, but the following is a simple estimate of the number of
* effective draws.
*
compute effsize=100.0*sumwt**2/(ndraws*sumwt2)
disp "Effective sample size (percent)" * .## effsize
*
* Normalize the weights to sum to 1.0
*
set weights 1 ndraws = weights/sumwt
*
* Extract the record for the 2nd coefficient in theta. This is the
* loading from the first real shock (the one we would most identify at
* being the US real shock) on Canadian GDP.
*
set can_on_r1 1 ndraws = timport(t) (2)
*
* Compute the weighted first and second moments, convert into mean and
* standard deviation.
*
sstats 1 ndraws weights*can_on_r1>>mean weights*can_on_r1^2>>mom2
disp "Mean, std dev on cr1" mean sqrt(mom2-mean^2)
*
* Compute an estimate of the density for that coefficient. We need to
* use the weights option. Note that this has a much broader spread of
* value than the MH estimates. This is because the "undesirable" draws
* are included (just with a very small weight) rather than being
* rejected.
*
density(grid=automatic,maxgrid=100,smoothing=2.0,weight=weights) $
  can_on_r1 1 ndraws xrl frl
scatter(style=line,vmin=0.0,footer="Loading of Real 1 onto Can GDP")
# xrl frl
*
* Compute an estimate of the density for the standard deviation for the
* first real shock.
*
set rlse 1 ndraws = sqrt(limport(t) (1))
density(grid=automatic,maxgrid=100,smoothing=2.0,weight=weights) $
  rlse / xrlse frlse
scatter(style=line,vmin=0.0,footer="standard error of real 1 shock")
# xrlse frlse

```

**Example 6.4 Structural VAR: Random Walk MH**

```

open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp canexpgdpchs canexpgdpds $
    canmls canusxsr usaexpgdpch
*
set logcangdp = log(canexpgdpchs)
set logcandefl = log(canexpgdpds)
set logcanml = log(canmls)
set logusagdp = log(usaexpgdpch)
set logexrate = log(canusxsr)
*
system(model=canmodel)
variables logusagdp logcangdp can3mthpcp logexrate logcanml logcandefl
lags 1 to 4
det constant
end(system)
*
estimate(noprint)
compute ncoef=%nreg
compute nvar =%nvar
*
dec frml[rect] bfrml
nonlin(parmset=svar) uf1 cr1 cf1 rf2 mf1 mm2
frml bfrml = ||1.0,0.0,uf1,0.0,0.0,0.0|$
           cr1,1.0,cf1,0.0,0.0,0.0|$
           0.0,0.0,1.0,rf2,0.0,0.0|$
           0.0,0.0,0.0,1.0,0.0,0.0|$
           mf1,0.0,0.0,0.0,1.0,mm2|$
           0.0,0.0,0.0,0.0,0.0,1.0||
compute uf1=cr1=cf1=rf2=mf1=mm2=pm2=0.0
*
* Compute the maximum of the log of the marginal posterior density for
* the B coefficients with a prior of the form  $|D|^{**(-\delta)}$ . Delta
* should be at least  $(nvar+1)/2.0$  to ensure an integrable posterior
* distribution.
*
compute delta=3.5
cvmodel(b=bfrml,parmset=svar,dfc=ncoef,pdf=delta,$
    dmatrix=marginalized,method=bfgs) %sigma
compute nfree=%nreg
*
* Start the chain at the maximizer
*
compute [vector] thetadraw=%parmspeek(svar)
compute logplast=%funcval
*
* This is the covariance matrix for the RW increment. We might have to
* change the scaling on this to achieve a reasonable acceptance
* probability.
*
compute f=0.5*%diag(%sqrt(%xdiag(%xx)))
*

```

```

compute nburn =2500
compute ndraws=10000
compute accept=0
*
declare vect lambdadraw(nvar) vdiag(nvar)
*
dec series[vect] tgibbs lgibbs
gset tgibbs 1 ndraws = %zeros(nfree,1)
gset lgibbs 1 ndraws = %zeros(nvar,1)
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Walk MH"
do draw=-nburn,ndraws
  *
  * Draw a new theta based at previous value
  *
  compute theta=thetadraw+%ranmvnormal(f)
  *
  * Evaluate the model there
  *
  compute %parmspoke(svar,theta)
  cvmodel(parmset=svar,dfc=ncoef,pdf=delta,$
    dmatrix=marginalized,method=evaluate,b=bfrml) %sigma
  compute logptest=%funcval
  *
  * Compute the acceptance probability
  *
  compute alpha =exp(logptest-logplast)
  if %ranflip(alpha)
    compute thetadraw=theta,logplast=logptest,accept=accept+1
  *
  infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")
  if draw<=0
    next
  *
  * Conditioned on theta, make a draw for lambda. (We don't need to do
  * this until we're saving results, since theta is being drawn
  * unconditionally).
  *
  compute %parmspoke(svar,thetadraw)
  compute b=bfrml(1)
  compute vdiag =%mqformdiag(%sigma,inv(b))
  ewise lambdadraw(i)=$
    (%nobs/2.0)*vdiag(i)/%rangamma(.5*(%nobs-ncoef)+delta+1)
  *
  * Save the theta's and lambda's
  *
  compute tgibbs(draw)=thetadraw
  compute lgibbs(draw)=lambdadraw
end do draw
infobox(action=remove)
*
@mcmcpostproc(ndraws=ndraws,mean=tmean,stderrs=tstderrs,cd=tcd) tgibbs
*

```

```

* Compute an estimate of the density for the 2nd coefficient in theta.
* This is the loading from the first real shock (the one we would most
* identify at being the US real shock) on Canadian GDP.
*
set can_on_r1 1 ndraws = tgibbs(t) (2)
density(grid=automatic,maxgrid=100,smoothing=2.0) can_on_r1 1 ndraws xrl fr1
scatter(style=line,vmin=0.0,footer="Loading of Real 1 onto Can GDP")
# xrl fr1
*
* Compute an estimate of the density for the standard deviation for the
* first real shock.
*
set r1se 1 ndraws = sqrt(lgibbs(t) (1))
density(grid=automatic,maxgrid=100,smoothing=2.0) r1se / xrlse fr1se
scatter(style=line,vmin=0.0,footer="standard error of real 1 shock")
# xrlse fr1se

```

## Example 6.5 Structural VAR: Independence Chain MH

```

open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp canexpgdpchs canexpgdpds $
    canmls canusxsr usaexpgdpch
*
set logcangdp = log(canexpgdpchs)
set logcandefl = log(canexpgdpds)
set logcanml = log(canmls)
set logusagdp = log(usaexpgdpch)
set logexrate = log(canusxsr)
*
system(model=canmodel)
variables logusagdp logcangdp can3mthpcp logexrate logcanml logcandefl
lags 1 to 4
det constant
end(system)
*
estimate(noprint)
compute ncoef=%nreg
compute nvar=%nvar
*
dec frml[rect] bfrml
nonlin(parmset=svar) uf1 cr1 cf1 rf2 mf1 mm2
frml bfrml = ||1.0,0.0,uf1,0.0,0.0,0.0|$
              cr1,1.0,cf1,0.0,0.0,0.0|$
              0.0,0.0,1.0,rf2,0.0,0.0|$
              0.0,0.0,0.0,1.0,0.0,0.0|$
              mf1,0.0,0.0,0.0,1.0,mm2|$
              0.0,0.0,0.0,0.0,0.0,1.0||
compute uf1=cr1=cf1=rf2=mf1=mm2=pm2=0.0
*
* Compute the maximum of the log of the marginal posterior density for
* the B coefficients with a prior of the form  $|D|^{**(-\delta)}$ . Delta
* should be at least (nvar+1)/2.0 to ensure an integrable posterior

```

```

* distribution.
*
compute delta=3.5
cvmodel(b=bfrml,parmset=svar,dfc=ncoef,pdf=delta,$
    dmatrix=marginalized,method=bfgs) %sigma
compute nfree=%nreg
*
* Start the chain at the maximizer
*
compute [vector] thetadraw=%parmspeek(svar)
compute logplast=%funcval
compute logqlast=0.0
*
* fxx is a factor of the (estimated) inverse Hessian at the final
* estimates. We might need to scale this if the acceptance probability
* is too small.
*
compute thetamean=thetadraw
compute fxx=%decomp(%xx)
*
* nuxx is the degrees of freedom for the multivariate Student used in
* drawing theta's This is also a "tuning" parameter for the MH process.
* We might need to adjust it.
*
compute nuxx=10.0
*
compute nburn =2500
compute ndraws=10000
compute accept=0
*
declare vect lambdadraw(nvar) vdiag(nvar)
*
dec series[vect] tgibbs lgibbs
gset tgibbs 1 ndraws = %zeros(nfree,1)
gset lgibbs 1 ndraws = %zeros(nvar,1)
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
    "Independence MH"
do draw=-nburn,ndraws
    *
    * Draw a new theta centered at thetamean from a multivariate t-density
    *
    compute theta=thetamean+%ranmvt(fxx,nuxx)
    compute logqtest=%ranlogkernel()
    *
    * Evaluate the model there
    *
    compute %parmspoke(svar,theta)
    cvmodel(parmset=svar,dfc=ncoef,pdf=delta,dmatrix=marginalized,$
        method=evaluate,b=bfrml) %sigma
    compute logptest=%funcval
    *
    * Compute the acceptance probability
    *

```

```

compute alpha =exp(logptest-logplast-logqtest+logqlast)
if %ranflip(alpha)
    compute thetadraw=theta,logplast=logptest,$
        logqlast=logqtest,accept=accept+1
*
infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")
if draw<=0
    next
*
* Conditioned on theta, make a draw for lambda. (We don't need to do
* this until we're saving results, since theta is being drawn
* unconditionally).
*
compute %parmspoke(svar,thetadraw)
compute b=bfrml(1)
compute vdiag =%mqformdiag(%sigma,inv(b))
ewise lambdadraw(i)=$
    (%nobs/2.0)*vdiag(i)/%rangamma(.5*(%nobs-ncoef)+delta+1)
*
* Save the theta's and lambda's
*
compute tgibbs(draw)=thetadraw
compute lgibbs(draw)=lambdadraw
end do draw
infobox(action=remove)
*
@mcmcpostproc(ndraws=ndraws,mean=tmean,stderrs=tstderrs,cd=tcd) tgibbs
*
* Compute an estimate of the density for the 2nd coefficient in theta.
* This is the loading from the first real shock (the one we would most
* identify at being the US real shock) on Canadian GDP.
*
set can_on_r1 1 ndraws = tgibbs(t)(2)
density(grid=automatic,maxgrid=100,smoothing=2.0) can_on_r1 1 ndraws xrl fr1
scatter(style=line,vmin=0.0,footer="Loading of Real 1 onto Can GDP")
# xrl fr1
*
* Compute an estimate of the density for the standard deviation for the
* first real shock.
*
set rlse 1 ndraws = sqrt(lgibbs(t)(1))
density(grid=automatic,maxgrid=100,smoothing=2.0) rlse / xrlse frlse
scatter(style=line,vmin=0.0,footer="standard error of real 1 shock")
# xrlse frlse

```

**Example 6.6 Univariate autoregression with prior**

```

open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp
*
* General to specific is one of the less stringent lag "pruning"
* techniques. In this case, it cuts 12 lags down to 8.
*
stwise(method=gtos,slstay=.10) can3mthpcp
# constant can3mthpcp{1 to 12}
*
* Standard set up for Gibbs sampler with independent prior
*
linreg(define=rateeq) can3mthpcp
# can3mthpcp{1 to 12} constant
*
* Save the lag coefficients to a series for later graphing
*
set bols 1 12 = %beta(t)
*
cmom(lastreg)
compute tobs=%nobs
compute nbeta=%nreg
*
* Prior for h
*
compute s2prior=1.0
compute nuprior=5.0
*
* Prior for coefficients. This has a mean of 1 on the first lag, and 0 on
* everything else. The precision matrix is diagonal. The standard errors
* are .20 on all the lag coefficients. This is a fairly "loose" prior.
* The prior precision on the constant is 0 (flat prior).
*
compute [vector] bprior=%unitv(nbeta,1)
dec vect hvector(nbeta)
ewise hvector(i)=(1./ .20)^2*%if(i<=12,i^0.0,0.0)
compute [symp] hprior=%diag(hvector)
*
* Start sampler at OLS estimates
*
compute bdraw=%beta
compute hdraw=1.0/%seesq
*
compute nburn =1000
compute ndraws=25000
*
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %zeros(nbeta,1)
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
"Gibbs Sampler"
do draw=-nburn,ndraws

```



```

*
* Get a draw for h, given bdraw
*
compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
compute hdraw  =%ranchisqr(nuprior+tobs)/rssplus
*
* Draw betas given hdraw
*
compute bdraw  =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
infobox(current=draw)
if draw<=0
  next
*
* Do the bookkeeping here.
*
compute bgibbs(draw)=bdraw
end do draw
infobox(action=remove)
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs) bgibbs
*
* Save the posterior means of the lag coefficients
*
set bnodecay 1 12 = bmean(t)
*
* These are the upper and lower 1 std deviation bounds for prior
*
set upper 1 12 = bprior(t)+1.0/sqrt(hvector(t))
set lower 1 12 = bprior(t)-1.0/sqrt(hvector(t))
*
* The "number" option is used when you want to label a graph with a
* numeric sequence rather than dates.
*
graph(number=1,footer="Comparison of Lag Distributions",key=upright,$
  klabels=||"OLS","Bayes","Lower Prior","Upper Prior"||) 4
# bols
# bnodecay
# upper / 3
# lower / 3
*
* Looks at sum of lags 6 and 7. (SUMMARIZE applies to the last
* regression, which despite all the calculations in the Gibbs sampler,
* is the original one.)
*
summarize
# can3mthpcp{6 7}
*
* Sums from 6 to end
*
summarize
# can3mthpcp{6 to 12}

```

## Example 6.7 Univariate Autoregression: Out-of-sample forecast performance

```

open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp
*
* Number of lags
*
compute nlags =8
*
* Start of evaluation period; end of evaluation period; number of
* forecast steps
*
compute fstart=2002:1
compute fend =2006:4
compute nsteps=8
*
* Standard set up for Gibbs sampler with independent prior. This is
* estimated over the period prior to the hold-back.
*
linreg(define=rateeq) can3mthpcp * fstart-1
# can3mthpcp{1 to nlags} constant
*
cmom(lastreg) * fstart-1
*
* We need to save these into our own variables
*
compute tobs=%nobs
compute nbeta=%nreg
compute betaols=%beta
compute seesqols=%seesq
*
* Prior for h
*
compute s2prior=1.0
compute nuprior=5.0
*****
procedure ForecastLoss sse
type real *sse
*
option real    decay    .5
option real    tight    .2
*
option integer nburn     500
option integer ndraws    2500
option integer seed
*
option integer steps     8
option integer from
option integer to
*
local vector bdraw

```

```

local vector hvector
local vector bprior
local symm    hprior
local real    hdraw rssplus ssethis
local integer draw start i
*
if .not.%defined(sse) {
    disp "###@ForecastLoss(options)  SSE"
    return
}
if .not.%defined(from).or..not.%defined(to) {
    disp "###@ForecastLoss (FROM=start,TO=end,other options) "
    return
}
*
* Prior for coefficients. This has a mean of 1 on the first lag, and 0 on
* everything else. The precision matrix is diagonal. The standard errors
* are <<tight>>/lag^<<decay>> so the precision is the squared reciprocal
* of that.
*
compute [vector] bprior=%unitv(nbeta,1)
dim hvector(nbeta)
ewise hvector(i)=(1./tight)^2*if(i<=nlags,i^(2*decay),0.0)
compute [symm] hprior=%diag(hvector)
*
compute bdraw=betaols
compute hdraw=1.0/seesqols
*
compute sse=0.0
seed seed
do draw=-nburn,ndraws
    *
    * Get a draw for h, given bdraw
    *
    compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
    compute hdraw  =%ranchisqr(nuprior+tobs)/rssplus
    *
    * Draw betas given hdraw
    *
    compute bdraw  =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
    if draw<=0
        next
    *
    * Stuff the drawn coefficients into the equation
    *
    compute %eqnsetcoeffs(rateeq,bdraw)
    *
    * Do a set of <<steps>> forecasts beginning with <<from>> and
    * continuing until <<steps>> periods before the end of the hold-back.
    * Compute the sum of squared forecast errors and add it to the
    * running sum. Note that this treats each horizon (from 1 to steps)
    * equally. Since the longer range forecasts are likely to have
    * greater errors (and greater uncertainty), this will tend to favor
    * settings which deliver better long-range forecasts.

```

```

*
* If we wanted this to be a truer "out-of-sample" forecasting
* evaluation, we would add a new data point to the cross product
* matrix before moving to the next horizon. While doable, that would
* increase the calculation time by a factor of <<steps>>
*
do start=from,to-steps
  forecast(steps=steps,from=start,model=rateeq,results=fcsts)
  sstats start start+steps-1 (fcsts(1)-can3mthpcp)^2>>ssethis
  compute sse+=ssethis
end do start
end do draw
*
* Divide the sum of squared errors by the number of forecasts
*
compute sse=sse/(ndraws*(to-steps-from+1))
end
*****
*
* We're optimizing over decay and tight.
*
nonlin decay tight
compute decay=.25,tight=.05
declare real sse
*
find(trace) min sse
*
* Reject out-of-range values. (Set the value to NA and skip the
* remaining calculation.)
*
if decay<0.0.or.tight<=0.0 {
  compute sse=%na
  next
}
@ForecastLoss(from=fstart,to=fend,steps=nsteps,$
  tight=tight,decay=decay,seed=27338) sse
end find

```

## Example 6.8 Bayesian VAR: Gibbs sampling

```

compute lags=4           ;*Number of lags
compute nstep=16         ;*Number of response steps
compute nburn=500        ;*Number of burn-in draws
compute ndraws=2500      ;*Number of keeper draws
*
open data haversample.rat
cal(q) 1959
data(format=rats) 1959:1 2006:4 ftb3 gdph ih cbhm
*
set loggdp = log(gdph)
set loginv = log(ih)
set logc   = log(cbhm)
*

```

```

* These are the controlling parameters for the symmetric "Minnesota"
* prior - the own lag tightness and the relative tightness on the other
* lags.
*
compute tight=.1
compute other=.5
*
* First, estimate the system with a prior via mixed estimation, to see if
* we come up with something similar via Gibbs sampling.
*
system(model=varmodel)
variables loggdp loginv logc ftb3
lags 1 to lags
specify(type=symmetric,tight=tight) other
det constant
end(system)
*****
*
* Estimate with a prior
*
estimate
compute nvar=%nvar
*
@BVARBuildPriorMN(model=varmodel,tight=.1,other=.5) hbpriors hpriors
@BVARFinishPrior hbpriors hpriors bprior hprior
*
compute sigmad=%sigma
cmom(model=varmodel)
*
dec vect[series] forecast(nvar)
dec vect[series] forestderr(nvar)
*
* Range to forecast
*
compute fstart=%regend()+1
compute fend =fstart+nstep-1
*
do i=1,nvar
    set forecast(i) fstart fend = 0.0
    set forestderr(i) fstart fend = 0.0
end do i
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) "Gibbs Sampler"
do draw=-nburn,ndraws
    infobox(current=draw)
    *
    * Draw b given sigma
    *
    compute bdraw =%ranmvkroncmom(%cmom,inv(sigmad),hprior,bprior)
    compute rssmat=%sigmacmom(%cmom,bdraw)
    *
    * Draw sigma given b
    *
    compute sigmad=%ranwisharti(%decomp(inv(rssmat)),%nobs)

```

```

if draw<=0
  next

  compute %modelsetcoeffs(varmodel,bdraw)
  simulate(model=varmodel,cv=sigmad,results=simresults,steps=nstep)
  do i=1,nvar
    set forecast(i) fstart fend = forecast(i)+simresults(i)
    set forestderr(i) fstart fend = forestderr(i)+simresults(i)**2
  end do i
end do draw
infobox(action=remove)
*
do i=1,nvar
  set forecast(i) fstart fend = forecast(i)/ndraws
  set forestderr(i) fstart fend = sqrt(forestderr(i)/ndraws-forecast(i)**2)
end do i
*
do i=1,nvar
  set lower fstart fend = forecast(i)-2.0*forestderr(i)
  set upper fstart fend = forecast(i)+2.0*forestderr(i)
  graph(header="Forecasts of "+%1(%modeldepvars(varmodel)(i))) 4
  # forecast(i) fstart fend
  # lower fstart fend 2
  # upper fstart fend 2
  # %modeldepvars(varmodel)(i) fstart-12 fstart-1
end do i

```

## Cross Section and Panel Data

### 7.1 Panel Data

The panel data model with individual effects takes the form:

$$y_{it} = \alpha_i + X_{it}\beta + u_{it}, i = 1, \dots, N; t = 1, \dots, T \quad (7.1)$$

This has slope coefficients that are shared by all individuals, but separate intercepts. Possible, but less common, is the inclusion of “time effects”, that is

$$y_{it} = \alpha_i + \lambda_t + X_{it}\beta + u_{it}, i = 1, \dots, N; t = 1, \dots, T$$

In conventional econometrics, there are two commonly used methods for estimating (7.1): *fixed effects* and *random effects*. Fixed effects estimates (7.1) as it’s written, with separate intercepts for each individual. This can be done cheaply by removing individual means from the data:

$$y_{it} - y_{i\bullet} = (X_{it} - X_{i\bullet})\beta + (u_{it} - u_{i\bullet}) \quad (7.2)$$

estimating  $\beta$  by OLS on (7.2). The  $\alpha_i$  can be backed out after the estimation, but most of the time, they’re considered to be nuisance parameters. In the common case of “big  $N$ , small  $T$ ”, their estimates are quite imprecise. It is also possible to do fixed effects by estimating (7.1) directly. This is known as LSDV (for Least Squares, Dummy Variable). When  $N$  is small (less than 50), it doesn’t take much time on a modern computer, and has the advantage that it directly estimates all the coefficients (in case the  $\alpha_i$  are of interest) and provides a full covariance matrix, including the individual effects.

Random effects moves the individual effect into the error term, generally in the form:

$$y_{it} = \alpha + X_{it}\beta + \varepsilon_i + \eta_{it} \quad (7.3)$$

By moving it into the error term, we add the assumption that  $EX_{it}\varepsilon_i = 0$ . This has the advantage of allowing for regressors that are time invariant ( $X_{it} = X_{is}$  for all  $i, s$  and  $t$ ). Such regressors are wiped out in computing  $(X_{it} - X_{i\bullet})$  in (7.2). It has the disadvantage that the assumption  $EX_{it}\varepsilon_i = 0$  might be unrealistic for some regressors. The most commonly studied example of this is a “returns to schooling” analysis, where one of the regressors is level of education. The  $\varepsilon_i$  is likely to include an unobservable component of “ability”, which we would expect to be correlated with education.

These two models can be estimated using conventional methods with the **PREG** (Panel REGression) instruction.

```
preg(method=fixed) invest
# value cap indivs
preg(method=random) invest
# value cap indivs constant
```

Random effects requires that the error variance be split between the individual effect  $\varepsilon_i$  and the unsystematic error  $\eta_{it}$ . There are various “feasible” methods that can be used to compute the component variances. The one chosen by RATS starts with the fixed effects estimators.

### Fixed Effects, Bayesian Methods

Just as there are two ways to compute fixed effects estimators using conventional techniques, there are two ways to compute them using Bayesian methods. The simpler of the two is the LSDV approach. (7.1) is a linear regression, which we can analyze using the methods developed for linear regressions. There isn’t much new about this, other than dealing with a large number of generated regressors (the dummy variables):

```
compute nindiv=10
dec vect[series] indivs(nindiv)
do i=1,nindiv
  set indivs(i) = %indiv(t)==i
end do i
linreg invest
# value cap indivs
```

This creates `indivs` as a collection of 10 series, each of which is a dummy for one of the individuals. (`%indiv(t)` is a function which maps to the number for an individual in a panel data set). Note that in the **LINREG** instruction, you can just use `indivs` to get the entire list.

```
compute [symm] hprior=%diag(||0.0,0.0||~%fill(1,nindiv,1.0/50.0^2))
```

`%fill` is a function which creates a matrix (here of dimensions `1 x nindiv`) with a constant value (here `1.0/50.0^2`) in each of its slots. The `~` is a matrix concatenation operator, so this creates the vector `[0,0,1/50^2,...,1/50^2]`, then makes `hprior` a diagonal matrix with those values down the diagonal.

The second method (which isn’t discussed in Koop) is to rearrange 7.1 to the form:

$$y_{it} - \alpha_i = X_{it}\beta + u_{it} \quad (7.4)$$

Given the  $\alpha$ ;  $\beta$  can be estimated by a low-dimension regression, and given  $\beta$ , the  $\alpha$  can be estimated as simple means across an individual. Thus, we have a Gibbs sampler setup, where we also need to include the precision of the error. Again, we will highlight the differences with examples earlier:



First, we define a new series `yadjust`, and create an equation with that as the dependent variable. We'll replace `yadjust` with the left side of (7.4) when we need it to estimate  $\beta$ .

```
set yadjust = invest
linreg(define=eqn) yadjust
# value cap
```

Inside the loop, `yadjust` is reset with

```
set yadjust = invest-adraw(%indiv(t))
cmom(equation=eqn)
```

With that done, the  $\beta$  and  $h$  are drawn using standard methods. Given those draws, the  $\alpha$  are drawn in a loop which uses the sum of  $y_{it} - X_{it}\beta$  across  $t$  as the sample statistic in estimating  $\alpha_i$ .

```
compute %eqnsetcoeffs(eqn,bdraw)
do i=1,nindiv
  sstats(smpl=%indiv(t)==i) / invest-%eqnprj(eqn,t)>>sumepsi
  compute hmean=hdraw*%nobs+haprior
  compute adraw(i)=(sumepsi*hdraw+haprior*aprior)/hmean+$
    %ran(sqrt(1.0/hmean))
end do i
```

While you might think that this would be more efficient than LSDV, it isn't, at least until the number of individuals is quite large. The LSDV analysis is able to do a single cross product matrix outside the loop. It has no  $O(NT)$  calculations inside the loop, while the second method has several. The one time-consuming calculation inside the loop is inverting a matrix with dimension (roughly)  $N$ . Inversion of an  $N \times N$  is an  $O(N^3)$  operation, so you can see that which is more efficient will depend upon how large  $N$  is relative to  $T$ .

## Random Effects, Bayesian Methods

The calculations for random effects are quite similar to the second method for calculating fixed effects. The difference is that the  $\alpha_i$  aren't drawn independently of each other, but come from a common "pool". Thus, while the  $\alpha_i$  are different, they can't be too different. As in classical econometrics, the two methods converge when the variance of the pool goes to infinity.

We will make a slight adjustment to Koop's procedure in order to keep it more similar to conventional random effects. In equation (7.3), it's standard practice to include the intercept, since the effects (as errors) are assumed to be mean zero. Thus, we will include an intercept in the main regression, and make the pool mean zero. So we need a gamma prior for the precision of the pool:

```
compute s2vprior=50.0^2
compute nuvprior=20
```

As you can see, this is much more informative than our typical variance prior. It really has to be in order to be of much use.<sup>1</sup> Almost everything else is the same as the previous example, except that we need the extra step of drawing the precision for the pool. This is the calculation at the bottom of Koop, p 154. In this code, %NORMSQR (ADRAW) computes the sum of squares of the elements of ADRAW.

```
compute nuvpost=nindiv+nuvprior
compute s2vpost=(%normsqr(adraw)+s2vprior*nuvprior)/nuvpost
compute hadraw=%ranchisqr(nuvpost)/(s2vpost*nuvpost)
```

### Random Coefficients Models

This is the big brother of the random effects model. Instead of the slopes being fixed and the intercept changing, the entire coefficient vector changes from individual to individual. However, they (the coefficient vectors) are constrained by being drawn from a common pool. (If there were no such constraint, there would be little point in estimating using panel data; we could just analyze each separately).

In order to implement this, we first will need separate cross product matrices for each individual:<sup>2</sup>

```
dec vect[symm] cmoms(nindiv)
do i=1,nindiv
  cmom(equation=eqn, smp1=%indiv(t)==i, matrix=cmoms(i))
end do i
```

Governing the common pool is a mean, which has prior mean and precision, and, a covariance matrix.<sup>3</sup> The appropriate prior on a covariance matrix is an inverse Wishart. We use an uninformative prior on the mean, and an informative one on the covariance matrix. (Again, if that is diffuse, there is almost no difference between this and estimating each individual separately).

```
compute [vector] bpoolprior=%zeros(1,%nreg)
compute [symm] hpoolprior=%zeros(%nreg,%nreg)
compute [symm] vpoolprior=%diag(||25.0^2,0.01,0.01||)
compute nupoolprior=20
```

We need to generate draws for the following (in some order):

1. The mean of the pool.
2. The precision of the pool.

<sup>1</sup>Conventional random effects estimators rely on data-determined estimates for the component variances to get proper behavior.

<sup>2</sup>A VECT[SYMM] is a VECTOR, each element of which is a SYMMETRIC matrix.

<sup>3</sup>In our implementation of the random effects model, these were zero, and the mean was shifted into the constant in the regression specification.

3. The individual coefficient vectors
4. The precision of the residuals

We will start everything at the OLS estimates.

```
dec vector[vector] bdraw(nindiv)
do i=1,nindiv
  compute bdraw(i)=%beta
end do i
compute bpooldraw=%beta
compute hpooldraw=%zeros(%nreg,%nreg)
*
compute s2draw =%seesq
```

The draw for  $h$  is similar to what we've seen many times, except that we now have to sum across individuals, since the coefficients are different for each:

```
compute rssplus=nuprior*s2prior
do i=1,nindiv
  compute rssplus+=%rsscmom(cmoms(i),bdraw(i))
end do i
compute hdraw=%ranchisqr(nuprior+tobs)/rssplus
```

Now we need to draw the individual coefficient vectors. This again, is the same calculation with `%ranmvpostcmom` that we've seen before many times, just in a loop. While we are drawing these, we are getting the sum and the sum of outer products of these for use in the next step.

```
compute vpoolpost=vpoolprior*nupoolprior
compute bsum=%zeros(%nreg,1)
do i=1,nindiv
  compute bdraw(i)=%ranmvpostcmom(cmoms(i),hdraw,hpooldraw,bpooldraw)
  compute vpoolpost=vpoolpost+%outerxx(bdraw(i)-bpooldraw)
  compute bsum=bsum+bdraw(i)
end do i
```

We next draw for the precision of the pool. Because we need the precision (not the variance) for the next step, we use the direct Wishart draw (`%RANWISHARTF`) rather than the inverse Wishart (`%RANWISHARTI`). This takes as its first parameter a factor of the inverse of the degrees of freedom times the target "mean" for the covariance matrix. We just left `vpoolpost` in a raw sum form (rather than dividing through by the degrees of freedom) since we would have to just undo that scaling now anyway.

```
compute nupoolpost=nupoolprior+nindiv
compute hpooldraw=%ranwishartf(%decomp(inv(vpoolpost)),nupoolpost)
```

and finally the mean of the pool:

```
compute bpooldraw=%ranmvpost(nindiv*hpooldraw,bsum/nindiv,$
    hpoolprior,bpoolprior)
```

For convenience in doing all this programming, we organized the individual coefficient vectors into a `VECT[VECT]`. However, our post-processing routines will like it better if we combine them into a single vector. So the bookkeeping includes:

```
ewise bmatrix(i,j)=bdraw(j)(i)
compute bigibbs(draw)=%vec(bmatrix)
```

This copies the information into a standard matrix, then saves it in vec form. In the post-processing, we use the standard procedure, then later re-arrange that for output:

```
@mcmcpostproc(ndraws=ndraws,mean=bimean,stderrs=bistderrs) bigibbs
do j=1,nindiv
    compute bmean      =%xcol(%reshape(bimean,%nreg,nindiv),j)
    ...
```

`%RESHAPE` basically undoes the `%vec(...)` operation, and the `%XCOL` pulls out column `j`. This is the simplest way to pull the information out of a stacked vector.

## 7.2 Probit and Tobit Models

Both of these have extremely well-behaved (log) likelihood functions – the probit (and logit) have globally concave log likelihoods, and the tobit has a globally concave likelihood in a particular reparameterization. If we were interested only in estimating the models with an informative prior, both would work quite well with random draws with importance sampling.

The advantage of the Gibbs sampling procedure that we discuss here is that

1. It can be used in more complicated bivariate or multivariate models where the likelihood function isn't as well behaved (and might not even be computable in a closed-form) and
2. It can be used to get more accurate estimates of the predictions for individual data points, since those are highly non-linear functions of the data.

### Tobit Model

The tobit model takes the form

$$y_i^* = x_i' \beta + \varepsilon_i, \varepsilon_i \sim N(0, \sigma^2) \text{ i.i.d.} \quad (7.5)$$

where  $y_i^*$  is an underlying latent variable, not necessarily observed. What is observed is

$$y_i = \max(y_i^*, 0)$$

Such a series is said to be censored below at 0. A common example is hours worked – you observe positive hours for anyone who works, but zero for anyone who doesn't. The model (7.5) would have some observations that are nearly at the point of choosing to work ( $y_i^*$  is just barely negative) and some where the individual is quite far from choosing to work. From the observed data, we can't distinguish between the two, since they both will report zero.

It's also possible to have the observed series truncated above, so that you see something like

$$y_i = \min(y_i^*, U)$$

An example of that is, for instance, a series of unemployment durations where anyone who is still unemployed at the end of the sample will not have an observed value. The censoring points can be fixed numbers, or they can vary by individual. Adjusting the calculations for top or variable truncation is straightforward.

The tobit model is estimated using RATS with the instruction **LDV**, with the options **CENSOR** and **LOWER** or **UPPER**. In the example we will be using, this is

```
ldv(censor=lower, lower=0.0) hours
# nwifeinc educ exper expersq age kidslt6 kidsge6 constant
```

**CENSOR=LOWER/UPPER/BOTH/NONE** selects the form of censoring, and the **LOWER** and **UPPER** options set the censoring limits.

If we could observe  $y_i^*$ , we could just estimate (7.5) using the methods discussed early on for linear models. We can't observe  $y_i^*$ , but, given  $\beta$  and  $\sigma^2$ , we do know how it's distributed. This suggests that we can do Gibbs sampling, treating the  $y_i^*$  as auxiliary parameters: at each sweep, patching over  $y_i^*$  for the censored observations, then updating  $\beta$  and  $\sigma^2$  using standard linear model techniques.

If  $y_i = 0$ , then we know that  $y_i^* = x_i'\beta + \varepsilon_i \leq 0$ . This is a Normal with mean  $x_i'\beta$  and variance  $\sigma^2$ , truncated above at 0. You can draw this with RATS using the function `%rantruncate(mean, sd, lower, upper)`, where **mean** and **sd** are the mean and standard deviation of the underlying Normal, **lower** is the lower truncation limit and **upper** is the upper truncation limit. Here, we have no lower limit, so we'll use the missing value (**%NA**) for that.

If you look at the code for the tobit example, you'll see that we start early on with

```
set ystar = hours
linreg(define=eqn) ystar
# nwifeinc educ exper expersq age kidslt6 kidsge6 constant
```

We don't use hours directly in defining this because we're going to be using the created series inside the loop. As before, it's more convenient to use the precision rather than  $\sigma^2$ . Given `bdraw` (the current coefficients) and `hdraw` (the current precision), the draw for `ystar` is done with

```
compute %eqnsetcoeffs(eqn,bdraw)
set ystar = %if(hours>0.0, hours, $
    %rantruncate(%eqnprj(eqn,t), ranse, %na, 0.0))
```

If `hours>0`, we just use `hours`; if it's zero, we draw from a Normal with mean `%eqnprj(eqn,t)`, standard deviation `ranse`, not truncated below (lower is `%na`) and truncated above at 0.0.

Given the draw for `ystar`, everything else goes through as with a linear model. However, because the data have changed, we have to recompute the cross product matrix on each sweep:

```
cmom(equation=eqn)
compute bdraw=%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
compute hdraw =%ranchisqr(nuprior+%nobs)/rssplus
```

### Probit Model

The probit model is estimated in RATS using the instruction `DDV`. In our example, this is done with

```
ddv(dist=probit) union
# potexp exp2 married partt grade constant
```

The latent variable model for the probit is

$$y_i^* = x_i' \beta + \varepsilon_i, \varepsilon_i \sim N(0, 1) \text{ i.i.d.} \quad (7.6)$$

$$y_i = \begin{cases} 1 & \text{if } y_i^* \geq 0 \\ 0 & \text{if } y_i^* < 0 \end{cases} \quad (7.7)$$

The variance is normalized to 1 because scaling the  $\varepsilon$ 's and  $\beta$  by the same value leaves the signs in (7.6) unchanged, and all we can observe are the signs (unlike the situation in the tobit where we had some data points which had hard values for  $y$ ).

The Gibbs sampler for the probit is similar to the tobit except that we don't have to estimate the precision, and we have to patch over `ystar` for all the observations. Here the `ystar` is bottom truncated for the data points where the dependent variable is 1 and (as above) top truncated for the ones where the dependent variable is 0. That is, if  $y_i = 1$ , then  $y_i^* = x_i' \beta + \varepsilon_i \geq 0$ , which is a bottom truncation.

```
compute %eqnsetcoeffs(eqn,bdraw)
set ystar = z=%eqnprj(eqn,t), $
    %if(union,%rantruncate(z,1.0,0.0,%na),%rantruncate(z,1.0,%na,0.0))
```

You might ask why, in this case, we do the calculation of the *index*  $x_t\beta$  before the %IF, while in the tobit we did it as part of the one %IF branch. Here, we need the same value regardless of which branch comes up. In the tobit, we needed it only in the second branch. Since this is a calculation that comes up for every entry on every sweep, it's good to avoid it if possible. On an %IF, RATS only does the calculations it needs in the branch that's taken; the other branch is ignored.

## 7.3 RATS Tips and Tricks

### Matrix Reshaping Functions

RATS has three configurations of matrices:

- **VECTOR** is a one-dimensional array, whose elements are addressed with a single subscript.
- **RECTANGULAR** is a general two-dimensional array, whose elements are addressed with a pair of subscripts.
- **SYMMETRIC** is a symmetric two-dimensional array, with only the part on and below the diagonal stored. Its elements are also addressed with a pair of subscripts. Note that you can use subscripts in either order; it's just that (i,j) and (j,i) map to the same physical location.

In some cases, the same set of values might be best organized as a **VECTOR** for some purposes and as a **RECTANGULAR** or **SYMMETRIC** for others. There are several functions which can be used to move information from one “shape” to another. Note that none of these change the input matrix; they just make copies of the data in a different shape.

**%VEC(a)** stacks a **RECTANGULAR** or **SYMMETRIC** into a **VECTOR**. When applied to a **RECTANGULAR**, it stacks the columns on top of one another. For the **SYMMETRIC**, it takes the lower triangle by rows, so, for instance, the first three elements will be a(1,1), a(2,1), a(2,2).

**%VECTORECT(v, n)** is the inverse of **%VEC** for a **RECTANGULAR**. n is the number of rows in the matrix being created; the elements of v are put in order into the columns.

**%VECTOSYMM(v, n)** is the inverse of **%VEC** for a **SYMMETRIC**. n is the number of rows in the matrix being created; the elements of v are put into the lower triangle by rows as described above.

**%RESHAPE(a, rows, cols)** is a general function for reshaping **VECTOR** and **RECTANGULAR**. rows and cols are the number of rows and columns in the matrix being formed.



## Example 7.1 Panel data: LSDV

```

open data grunfeld.xls
calendar(panelobs=20,a) 1935
all 10//1954:01
data(format=xls,org=columns) 1//1935:01 10//1954:01 $
    firm year invest value cap
*
* Create the dummies for the individuals
*
compute nindiv=10
dec vect[series] indivs(nindiv)
do i=1,nindiv
    set indivs(i) = %indiv(t)==i
end do i
*
* Estimate by OLS
*
linreg invest
# value cap indivs
*
* Prior for the regression equation variance.
*
compute s2prior=100.0^2
compute nuprior=5.0
*
* Prior mean and precision for the coefficients. This has an
* non-informative prior on the two slope coefficients and a prior with
* standard deviation 50 on the individual effects.
*
compute [vector] bprior=%zeros(nindiv+2,1)
compute [symm]   hprior=%diag(||0.0,0.0||~%fill(1,nindiv,1.0/50.0^2))
*
* Compute the cross product matrix from the regression. This will
* include the dependent variable as the final row. The cross product
* matrix has all the sufficient statistics for the likelihood (except
* the number of observations, which is %nobs).
*
cmom(lastreg)
*
* Start the Gibbs sampler at the OLS estimates.
*
compute bdraw =%beta
compute s2draw=%seesq
*
compute nburn =1000
compute ndraws=100000
*
dec series[vect] bgibbs
dec series      hgibbs
gset bgibbs 1 ndraws = %zeros(%nreg,1)
set  hgibbs 1 ndraws = 0.0
*
compute savage=0.0

```

```

do draw=-nburn, ndraws
  *
  * Draw residual precision conditional on previous beta
  *
  compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
  compute hdraw  =%ranchisqr(nuprior+%nobs)/rssplus
  *
  * Draw betas given hdraw
  *
  compute bdraw  =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
  if draw<=0
    next
  *
  * Do the bookkeeping here.
  *
  compute bgibbs(draw)=bdraw
  compute hgibbs(draw)=hdraw
end do draw
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,cd=bcd) bgibbs
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" "Std Error" "CD"
do i=1,%nreg
  report(row=new,atcol=1) %eqnreglabels(0)(i) bmean(i) bstderrs(i) bcd(i)
end do i
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.##")
report(action=show)

```

## Example 7.2 Panel data: Fixed Effects

```

open data grunfeld.xls
calendar(panelobs=20,a) 1935
all 10//1954:01
data(format=xls,org=columns) 1//1935:01 10//1954:01 $
  firm year invest value cap
*
compute nindiv=10
*
* We define the equation in terms of <<yadjust>> (which is just the
* original dependent variable at first) since we need to replace the
* dependent variable with a mean-corrected version later on.
*
set yadjust = invest
linreg(define=eqn) yadjust
# value cap
*
* Prior for equation variance.
*
compute s2prior=100.0^2
compute nuprior=5.0
*
* Prior mean and precision for the regression coefficients
*

```

```

compute [vector] bprior=%zeros(1,%nreg)
compute [symm]   hprior=%zeros(%nreg,%nreg)
*
* Prior mean and precision for the individual effects
*
compute aprior =0.0
compute haprior=1.0/50.0^2
*
* Start with OLS (zeros for the alphas)
*
compute [vector] adraw  =%zeros(nindiv,1)
compute [vector] bdraw  =%xsubvec(%beta,1,%nreg)
compute s2draw =%seesq
compute hadraw =0.0
*
* This needs a high burn-in because the intercept in the regression is
* highly correlated with the individual effects, but is drawn
* separately. This also demonstrates code to allow for sub-sampling -
* saving only some of the draws. nsub is the sampling frequency. Here,
* we keep all the draws. If you make nsub=5, 5 times as many draws will
* be done, but it will still save just ndraws of them.
*
compute nburn =10000
compute ndraws=10000
compute nsub  =1
*
dec series[vect] bgibbs
dec series[vect] agibbs
dec series      hgibbs
*
gset bgibbs 1 ndraws = %zeros(%nreg,1)
gset agibbs 1 ndraws = %zeros(nindiv,1)
set  hgibbs 1 ndraws = 0.0
*
infobox(action=define,progress,lower=-nburn,upper=ndraws*nsub) $
  "Fixed Effects-Gibbs Sampling"
do draw=-nburn,nsub*ndraws
  *
  * Compute y's adjusted for the current individual intercepts
  *
  set yadjust = invest-adraw(%indiv(t))
  cmom(equation=eqn)
  *
  * Draw residual precision conditional on previous beta
  *
  compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
  compute hdraw  =%ranchisqr(nuprior+%nobs)/rssplus
  *
  * Draw betas given hdraw
  *
  compute bdraw  =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
  *
  * Draws alphas given betas, h and ha
  *

```

```

compute %eqnsetcoeffs (eqn,bdraw)
do i=1,nindiv
  sstats (smp1=%indiv(t)==i) / invest-%eqnprj (eqn,t)>>sumepsi
  compute hmean=hdraw*%nobs+haprior
  compute adraw(i)=$
    (sumepsi*hdraw+haprior*aprior)/hmean+%ran(sqrt(1.0/hmean))
end do i
infobox(current=draw)
if draw<=0.or.%clock(draw,nsub)<>nsub
  next
*
* Do the bookkeeping here.
*
compute bgibbs(draw/nsub)=bdraw
compute hgibbs(draw/nsub)=hdraw
compute agibbs(draw/nsub)=adraw
end do draw
infobox(action=remove)
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,$
  cd=bcd,nse=bnse) bgibbs
@mcmcpostproc(ndraws=ndraws,mean=amean,stderrs=astderrs,$
  cd=acd,nse=anse) agibbs
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" $
  "Std Error" "CD"
do i=1,%nreg
  report(row=new,atcol=1) %eqnreglabels(0)(i) bmean(i) $
    bstderrs(i) bcd(i)
end do i
report(row=new)
do i=1,nindiv
  report(row=new,atcol=1) "Indiv-"+i amean(i) astderrs(i) acd(i)
end do i
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.##")
report(action=show)

```

### Example 7.3 Panel data: Random Effects (hierarchical prior)

```

open data grunfeld.xls
calendar(panelobs=20,a) 1935
all 10//1954:01
data(format=xls,org=columns) 1//1935:01 10//1954:01 $
  firm year invest value cap
*
compute nindiv=10
*
* We define the equation in terms of <<yadjust>> (which is just the
* original dependent variable at first) since we need to replace the
* dependent variable with a mean-corrected version later on.
*
set yadjust = invest
linreg(define=eqn) yadjust

```

```

# constant value cap
*
* Prior for equation variance.
*
compute s2prior=100.0^2
compute nuprior=5.0
*
* Prior mean and precision for the regression coefficients
*
compute [vector] bprior=%zeros(1,%nreg)
compute [symm]   hprior=%zeros(%nreg,%nreg)
*
* Prior for the precision of the common pool for the individual
* intercepts
*
compute s2vprior=50.0^2
compute nuvprior=20
*
* Start with OLS (zeros for the alphas)
*
compute [vector] adraw  =%zeros(nindiv,1)
compute [vector] bdraw  =%xsubvec(%beta,1,%nreg)
compute s2draw =%seesq
compute hadraw =0.0
*
* This needs a high burn-in because the intercept in the regression is
* highly correlated with the individual effects, but is drawn separately.
*
compute nburn =10000
compute ndraws=10000
*
dec series[vect] bgibbs
dec series[vect] agibbs
dec series      hgibbs
*
gset bgibbs 1 ndraws = %zeros(%nreg,1)
gset agibbs 1 ndraws = %zeros(nindiv,1)
set  hgibbs 1 ndraws = 0.0
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Effects-Gibbs Sampling"
do draw=-nburn,ndraws
  *
  * Compute y's adjusted for the current individual intercepts
  *
  set yadjust = invest-adraw(%indiv(t))
  cmom(equation=eqn)
  *
  * Draw residual precision conditional on previous beta
  *
  compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
  compute hdraw  =%ranchisqr(nuprior+%nobs)/rssplus
  *
  * Draw betas given hdraw

```

```

*
compute bdraw  =%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
*
* Draws alphas given betas, h and ha
*
do i=1,nindiv
  sstats(smpl=%indiv(t)==i) / $
  invest-%dot(%eqnxvector(eqn,t),bdraw)>>sumepsi
  compute hmean=hdraw*%nobs+hadraw
  compute adraw(i)=sumepsi*hdraw/hmean+%ran(sqrt(1.0/hmean))
end do i
*
* Draw the precision for the alpha pool given the alphas.
* %normsqr(adraw) computes the sum of squares of the alphas.
*
compute nuvpost=nindiv+nuvprior
compute s2vpost=(%normsqr(adraw)+s2vprior*nuvprior)/nuvpost
compute hadraw=%ranchisqr(nuvpost)/(s2vpost*nuvpost)
infobox(current=draw)
if draw<=0
  next
*
* Do the bookkeeping here.
*
compute bgibbs(draw)=bdraw
compute hgibbs(draw)=hdraw
compute agibbs(draw)=adraw
end do draw
infobox(action=remove)
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,cd=bcd) bgibbs
@mcmcpostproc(ndraws=ndraws,mean=amean,stderrs=astderrs,cd=acd) agibbs
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" $
  "Std Error" "CD"
do i=1,%nreg
  report(row=new,atcol=1) %eqnreglabels(0)(i) bmean(i) $
    bstderrs(i) bcd(i)
end do i
report(row=new)
do i=1,nindiv
  report(row=new,atcol=1) "Indiv-"+i amean(i) astderrs(i) acd(i)
end do i
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.###")
report(action=show)

```

## Example 7.4 Panel data: Random coefficients model

```

open data grunfeld.xls
calendar(panelobs=20,a) 1935
all 10//1954:01
data(format=xls,org=columns) 1//1935:01 10//1954:01 $
    firm year invest value cap
*
* The "classical" analog of this is the "Swamy" random
* coefficients model
*
@swamy invest
# constant value cap
*
compute nindiv=10
*
linreg(define=eqn) invest
# constant value cap
compute tobs=%nobs
*
* Prior for variance.
*
compute s2prior=100.0^2
compute nuprior=5.0
*
* Compute separate cross product matrices for each individual
*
dec vect[symm] cmoms(nindiv)
do i=1,nindiv
    cmom(equation=eqn, smpl=%indiv(t)==i, matrix=cmoms(i))
end do i
*
* Prior mean and precision for the mean of the pooled density. This is
* uninformative.
*
compute [vector] bpoolprior=%zeros(1,%nreg)
compute [symm] hpoolprior=%zeros(%nreg,%nreg)
*
* Parameters for the (inverse) Wishart prior on the variance of the pool
* from which the individual coefficients are drawn. This is specified as
* the "mean" of the distribution, similar to what we've done with
* univariate inverse gammas.
*
compute [symm] vpoolprior=%diag(||25.0^2,0.01,0.01||)
compute nupoolprior=20
*
* Create a separate coefficient vector for each individual. Initialize
* it to the pooled OLS estimate which is where we'll start the Gibbs
* sampler.
*
dec vector[vector] bdraw(nindiv)
do i=1,nindiv
    compute bdraw(i)=%beta
end do i

```

```

compute bpooldraw=%beta
compute hpooldraw=%zeros(%nreg,%nreg)
*
compute s2draw=%seesq
*
compute nburn=1000
compute ndraws=10000
*
dec series[vect] bgibbs
dec series[vect] bigibbs
dec series      hgibbs
*
dec rect bmatrix(%nreg,nindiv)
gset bgibbs 1 ndraws = %zeros(%nreg,1)
gset bigibbs 1 ndraws = %zeros(nindiv*%nreg,1)
set  hgibbs 1 ndraws = 0.0
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Random Coefficients-Gibbs Sampling"
do draw=-nburn,ndraws
  *
  * Draw residual precision conditional on previous betas
  *
  compute rssplus=nuprior*s2prior
  do i=1,nindiv
    compute rssplus+=%rsscmom(cmoms(i),bdraw(i))
  end do i
  compute hdraw=%ranchisqr(nuprior+tobs)/rssplus
  *
  * Draw betas conditional on h and the pooled mean and precision.
  * While we are doing this, also compute the sum of outer products of
  * the difference between bdraw and the pooled beta. That will be
  * needed in the next step.
  *
  compute vpoolpost=vpoolprior*nupoolprior
  compute bsum=%zeros(%nreg,1)
  do i=1,nindiv
    compute bdraw(i)=%ranmvpostcmom(cmoms(i),hdraw,hpooldraw,bpooldraw)
    compute vpoolpost=vpoolpost+%outerxx(bdraw(i)-bpooldraw)
    compute bsum=bsum+bdraw(i)
  end do i
  *
  * Draw hpooldraw given bdraw and bpool (as summarized in vbpost)
  *
  compute nupoolpost=nupoolprior+nindiv
  compute hpooldraw=%ranwishartf(%decomp(inv(vpoolpost)),nupoolpost)
  *
  * Draw bpooldraw given bdraw, hpooldraw
  *
  compute bpooldraw=$
    %ranmvpost(nindiv*hpooldraw,bsum/nindiv,hpoolprior,bpoolprior)
  infobox(current=draw)
  if draw<=0
    next

```



```

*
* Do the bookkeeping here.
*
ewise bmatrix(i,j)=bdraw(j)(i)
compute bgibbs(draw) =bpooldraw
compute bigibbs(draw)=%vec(bmatrix)
compute hgibbs(draw) =hdraw
end do draw
infobox(action=remove)
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,cd=bcd) bgibbs
@mcmcpostproc(ndraws=ndraws,mean=bimean,stderrs=bstderrs) bigibbs
*
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" "Std Error" "CD"
do i=1,%nreg
  report(row=new,atcol=1) %eqnreglabels(0)(i) bmean(i) $
    bstderrs(i) bcd(i)
end do i
*
do j=1,nindiv
  compute bmean =%xcol(%reshape(bimean,%nreg,nindiv),j)
  compute bstderrs=%xcol(%reshape(bstderrs,%nreg,nindiv),j)
  report(row=new)
  report(row=new,atcol=1,span) "For individual "+j
  do i=1,%nreg
    report(row=new,atcol=1) %eqnreglabels(0)(i) bmean(i) bstderrs(i)
  end do i
end do j
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.##")
report(action=show)

```

## Example 7.5 Tobit model

```

open data mroz.raw
data(format=free,org=columns) 1 753 inlf hours kidslt6 kidsge6 age $
  educ wage repwage hushrs husage huseduc huswage faminc mtr $
  motheduc fatheduc unem city exper nwifeinc lwage expersq
*
set expersq = exper**2
*
* A tobit model is estimated by ML using the instruction LDV
*
ldv(censor=lower,lower=0.0) hours
# nwifeinc educ exper expersq age kidslt6 kidsge6 constant
*
* Estimate by OLS.
*
set ystar = hours
linreg(define=eqn) ystar
# nwifeinc educ exper expersq age kidslt6 kidsge6 constant
*
* Start Gibbs sampler at OLS estimates

```

```

*
compute bdraw=%beta
compute hdraw=1.0/%sigmasq
*
* Non-informative prior on the regression coefficients
*
compute [vector] bprior=%zeros(%nreg,1)
compute [symm]   hprior=%zeros(%nreg,%nreg)
*
* Weakly informative prior for variance.
*
compute s2prior=200.0^2
compute nuprior=5.0
*
compute nburn =2500
compute ndraws=10000
*
dec series[vect] bgibbs
dec series hgibbs
gset bgibbs 1 ndraws = %zeros(%nreg,1)
set  hgibbs 1 ndraws = 0.0
*
infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Tobit Model-Gibbs Sampling"
do draw=-nburn,ndraws
  *
  * Patch the dependent variable for the censored observations. A
  * censored observation has  $Xb + u \leq 0$ , so  $y \sim N(Xb, \sigma^2)$  with no
  * bottom truncation, top truncated at 0. This is drawn with
  * %rantruncate( $Xb$ ,  $\sigma$ , %na, 0)
  *
  compute ranse=1.0/sqrt(hdraw)
  compute %eqnsetcoeffs(eqn,bdraw)
  set ystar = %if(hours>0.0,hours,$
    %rantruncate(%eqnprj(eqn,t),ranse,%na,0.0))
  *
  * Compute draws for the coefficients and for h
  *
  cmom(equation=eqn)
  compute bdraw=%ranmvpostcmom(%cmom,hdraw,hprior,bprior)
  compute rssplus=nuprior*s2prior+%rsscmom(%cmom,bdraw)
  compute hdraw =%ranchisqr(nuprior+%nobs)/rssplus
  *
  infobox(current=draw)
  if draw<=0
    next
  *
  * Save the beta's and h's
  *
  compute bgibbs(draw)=bdraw
  compute hgibbs(draw)=hdraw
end do draw
infobox(action=remove)
*

```

```
@mcmcpostproc (ndraws=ndraws, mean=bmean, stderrs=bstderrs, cd=bcd) bgibbs
report (action=define)
report (atrow=1, atcol=1, align=center) "Variable" "Coeff" "Std Error" "CD"
do i=1,%nreg
    report (row=new, atcol=1) %eqnreglabels(0) (i) bmean(i) $
        bstderrs(i) bcd(i)
end do i
report (action=format, atcol=2, tocol=3, picture="*.###")
report (action=format, atcol=4, picture="*.##")
report (action=show)
```

## Example 7.6 Probit model

```
open data cps88.asc
data (format=prn, org=columns) 1 1000 age exp2 grade ind1 married $
    lnwage occl partt potexp union weight high
*
* ML estimates are done with the DDV instruction
*
ddv(dist=probit) union
# potexp exp2 married partt grade constant
*
* Linear probability model (OLS)
*
set ystar = union
linreg(define=eqn) ystar
# potexp exp2 married partt grade constant
*
* Start Gibbs sampler at LPM estimates
*
compute bdraw=%beta
*
* Non-informative prior on the regression coefficients
*
compute [vector] bprior=%zeros(%nreg,1)
compute [symm] hprior=%zeros(%nreg,%nreg)
*
compute nburn =500
compute ndraws=2000
*
dec series[vect] bgibbs
gset bgibbs 1 ndraws = %zeros(%nreg,1)
*
infobox(action=define, progress, lower=-nburn, upper=ndraws) $
    "Probit Model-Gibbs Sampling"
do draw=-nburn,ndraws
    * Patch the dependent variable.
    *
    * %rantruncate(z,1.0,0.0,%na) draws a truncated deviate from a Normal
    * with mean z and std dev 1, truncated below at 0 and not truncated
    * above.
    *
    * %rantruncate(z,1.0,%na,0) is the same, but not truncated below and
```

```

* truncated above at 0.
*
compute %eqnsetcoeffs(eqn,bdraw)
set ystar = z=%eqnprj(eqn,t), $
    %if(union,%rantruncate(z,1.0,0.0,%na),%rantruncate(z,1.0,%na,0.0))
*
* Compute draws for the coefficients
*
cmom(equation=eqn)
compute bdraw=%ranmvpostcmom(%cmom,1.0,hprior,bprior)
*
infobox(current=draw)
if draw<=0
    next
compute bgibbs(draw)=bdraw
end do draw
infobox(action=remove)
@mcmcpostproc(ndraws=ndraws,mean=bmean,stderrs=bstderrs,cd=bcd) bgibbs
*
* With an uninformative prior, there really isn't much difference
* between the ML estimates and the Gibbs estimates. In fact, if you have
* an informative prior and all you're interested in are the coefficient
* estimates, you're probably better off doing importance sampling using
* the asymptotic distribution from probit as the importance function.
* (Probits have a very well-behaved likelihood).
*
report(action=define)
report(atrow=1,atcol=1,align=center) "Variable" "Coeff" "Std Error" "CD"
do i=1,%nreg
    report(row=new,atcol=1) %eqnreglabels(0)(i) bmean(i) $
        bstderrs(i) bcd(i)
end do i
report(action=format,atcol=2,tocol=3,picture="*.###")
report(action=format,atcol=4,picture="*.##")
report(action=show)

```

## State Space Models

We'll stick here with the notation used in the RATS manual, since it maps directly into the names of the options on the **DLM** instruction. Note that Koop uses a different timing scheme on the state variable, the same choice made by, for instance, Durbin & Koopman (2012). The scheme used in RATS is the same as used in, for instance, Hamilton (1994).

$$\mathbf{X}_t = \mathbf{A}_t \mathbf{X}_{t-1} + \mathbf{Z}_t + \mathbf{F}_t \mathbf{w}_t \quad (8.1)$$

$$\mathbf{Y}_t = \mathbf{c}_t' \mathbf{X}_t + \mathbf{v}_t \quad (8.2)$$

The  $\mathbf{X}$ 's are the unobservable state variables. The  $\mathbf{Y}$ 's are the observable data. 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 allows for there to be fewer shocks to the states than there are states (which is generally the case). The  $\mathbf{v}$ 's are measurement errors. The  $\mathbf{w}$ 's and  $\mathbf{v}$ 's are assumed to be mean zero, Normally distributed, independent across time and (generally) independent of each other at time  $t$  as well.

Extracting estimates of the  $\mathbf{X}$ 's given the  $\mathbf{Y}$ 's is, itself, an inference process which uses Bayesian updating methods. If all the matrices ( $\mathbf{A}$ ,  $\mathbf{Z}$ ,  $\mathbf{F}$ ,  $\mathbf{c}$ ) are known and we also know the covariance matrices of  $\mathbf{w}$  and  $\mathbf{v}$ , we still have a problem trying to infer  $\mathbf{X}_1$  from  $\mathbf{Y}_1$ — according to (8.1),  $\mathbf{X}_1$  depends upon  $\mathbf{X}_0$ , which we don't have. The solution to this is to choose a prior for  $\mathbf{X}_0$ . For convenience, this is generally chosen to be multivariate Normal with mean  $\mathbf{X}_{0|0}$  and covariance matrix  $\Sigma_{0|0}$ , independent of the future  $\mathbf{w}$ 's and  $\mathbf{v}$ 's. ( $\mathbf{X}_{t|s}$  for any  $t$  and  $s$  is the expected value of  $\mathbf{X}_t$  given information through  $s$ ). If, as is generally the case,  $\mathbf{Z}$  is zero, then we can apply (8.1) to get

$$\mathbf{X}_{1|0} = \mathbf{A}_1 \mathbf{X}_{0|0} \quad (8.3)$$

$$\Sigma_{1|0} = \mathbf{A}_1 \Sigma_{0|0} \mathbf{A}_1' + \mathbf{F}_1 \mathbf{M}_1 \mathbf{F}_1' \quad (8.4)$$

where we'll use the notation  $\mathbf{M}_t = \text{var}(\mathbf{w}_t)$  and  $\mathbf{N}_t = \text{var}(\mathbf{v}_t)$ . Applying (8.2) means that

$$\begin{bmatrix} \mathbf{X}_1 \\ \mathbf{Y}_1 \end{bmatrix} \sim N \left( \begin{bmatrix} \mathbf{X}_{1|0} \\ \mathbf{c}_1' \mathbf{X}_{1|0} \end{bmatrix}, \begin{bmatrix} \Sigma_{1|0} & \Sigma_{1|0} \mathbf{c}_1 \\ \mathbf{c}_1' \Sigma_{1|0} & \mathbf{c}_1' \Sigma_{1|0} \mathbf{c}_1 + \mathbf{N}_1 \end{bmatrix} \right) \quad (8.5)$$

The Kalman filter is the application of standard inference for the conditional distribution in a multivariate Normal (see Appendix C): computing the distri-

bution of  $\mathbf{X}_1|\mathbf{Y}_1$ , we get

$$\mathbf{X}_{1|1} = \mathbf{X}_{1|0} + \Sigma_{1|0}\mathbf{c}_1 (\mathbf{c}_1'\Sigma_{1|0}\mathbf{c}_1 + \mathbf{N}_1)^{-1} (\mathbf{Y}_1 - \mathbf{c}_1'\mathbf{X}_{1|0}) \quad (8.6)$$

$$\Sigma_{1|1} = \Sigma_{1|0} - \Sigma_{1|0}\mathbf{c}_1 (\mathbf{c}_1'\Sigma_{1|0}\mathbf{c}_1 + \mathbf{N}_1)^{-1} \mathbf{c}_1'\Sigma_{1|0} \quad (8.7)$$

From (8.5), we also have that

$$\mathbf{Y}_1 \sim N(\mathbf{c}_1'\mathbf{X}_{1|0}, \mathbf{c}_1'\Sigma_{1|0}\mathbf{c}_1 + \mathbf{N}_1) \quad (8.8)$$

We can now repeat (8.3), (8.4), (8.6), (8.7) and (8.8) updating all the subscripts by 1. Note, however, that the densities in (8.5) and (8.8) are now (at entry 2) conditional on having observed  $\mathbf{Y}_1$ . Repeating this through the entire data set gives us distributions for  $\mathbf{X}_{t|t}$  and for  $\mathbf{Y}_t|\mathbf{Y}_{t-1}, \dots, \mathbf{Y}_1$ . The latter allows us to compute the likelihood of the model by the standard time-series trick of factoring by sequential conditionals:

$$p(\mathbf{Y}_1, \dots, \mathbf{Y}_T) = p(\mathbf{Y}_1)p(\mathbf{Y}_2|\mathbf{Y}_1)p(\mathbf{Y}_3|\mathbf{Y}_1, \mathbf{Y}_2) \cdots p(\mathbf{Y}_T|\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_{T-1}) \quad (8.9)$$

So for evaluating the likelihood, we just need the fairly simple Kalman filter. If we have free parameters in the matrices governing the processes, we can estimate them by classical methods using maximum likelihood, or by Bayesian methods using importance sampling or Metropolis-Hastings. (They will rarely have any nice unconditional distribution).

State-space models are analyzed using RATS with the instruction **DLM**. A state-space model has quite a few components; these are input into **DLM** using a set of options which have names that generally match those used in equations (8.1) and (8.2). One of the more basic models, which is used by itself and is also a component in more complicated models, is the local trend model, shown on page 194 in Koop. In our timing scheme, that's

$$\begin{aligned} \begin{bmatrix} \mu_t \\ \tau_t \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_{t-1} \\ \tau_{t-1} \end{bmatrix} + \begin{bmatrix} \xi_t \\ \zeta_t \end{bmatrix} \\ y_t &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \mu_t \\ \tau_t \end{bmatrix} + \varepsilon_t \end{aligned} \quad (8.10)$$

$\tau_t$  is the local trend rate,  $\mu_t$  is the local trend formed by adding its last value to the last value of the rate. So the observable is the local trend plus the measurement error  $\varepsilon_t$ .

One thing to note about the model as written is that it has one observable, but is trying to decompose that into three independent shocks. That often proves to be too many, and, in practice, it is hard to separate  $\xi_t$  from  $\varepsilon_t$ . So we'll work with the simpler model with

$$\begin{bmatrix} \mu_t \\ \tau_t \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_{t-1} \\ \tau_{t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \zeta_t$$

From these, we can read off

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{F} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

which are fixed matrices. What we *don't* necessarily know are the variances of the components  $\varepsilon_t$  and  $\zeta_t$ . In the design of **DLM**, the variance of the measurement errors (here  $\varepsilon_t$ ) is input using the option **SV**, which takes a real value or parameter (or a **SYMMETRIC** matrix if the dimension of the observable is bigger than one). And the variance of the shock in the transition equation is input using **SW**, similarly a real or a **SYMMETRIC** depending upon the dimension. We'll have two free parameters: one for each of these variances.

We also have the question of the pre-sample mean and variance:  $\mathbf{X}_{0|0}$  and  $\Sigma_{0|0}$ . This model has two states and two unit roots in the transition equation, so there is no stationary distribution. RATS has the special option **EXACT** to deal with this. This uses non-standard matrix calculations (appendix D) to allow for a non-informative prior for the states.<sup>1</sup> How this works in this case is that the first two observations are enough to figure out those two initial states. In the likelihood function (8.9), the  $p(\mathbf{Y}_1)$  and  $p(\mathbf{Y}_2|\mathbf{Y}_1)$  aren't included because the predictive variance is still infinite. Seeing two data points resolves the 2-vector of diffuse states, similar to the way that two observations in a regression model is the minimum required to get estimates if you have two free parameters with diffuse priors.

One important thing to note about the Kalman filter is that the estimates of the means  $\mathbf{X}_{t|s}$  aren't affected if all variances are scaled by a constant, that is, the point estimates of the states depend only on *relative* variances. This is exploited in, for instance, the Hodrick-Prescott filter, which uses this precise model with the variance ratio  $\sigma_\varepsilon^2/\sigma_\zeta^2$  pegged to 1600 (for quarterly data). Particularly when you have just one observable, it is often convenient to analyze the model with all variances (precisions) parameterized as proportions to the variance (precision) of the measurement error.

In almost all state-space models, there are several parameters which give component variances. At a minimum, you have to have as many shocks as observable series (otherwise the model predicts an identity connecting the series), but most models (such as this one) have more shocks than observables. This produces a situation where some of the shocks can have zero variance; in fact, it's possible for the maximum likelihood estimate of a variance to come in negative if it's parameterized as just an unconstrained real value. (That's the main reason why the  $\zeta_t$  is often dropped from this model). This can happen because all that matters for computing the likelihood is that the variance in (8.8) be positive definite. Since that's the sum of two terms, if one of those is big enough, the other could fail to be positive definite and still give a computable likelihood.

---

<sup>1</sup>"Exact" refers to the fact that this does an exact calculation, rather than approximating it with very large but finite variances.

With a prior, we can easily prevent the negative values, but we need to allow for the possibility that the value could be quite small. A gamma is an obvious choice for a prior, but it sometimes isn't convenient, particularly when there are other parameters that aren't variances. You can also parameterize a variance in log form, where a value of  $-\infty$  represents a zero variance. When a state-space model is estimated by maximum likelihood, you will sometimes see the variance parameterized in standard deviation form, that is, using the  $\sigma$  as the parameter. In Bayesian methods, there's no real advantage in that, and it's actually not a good idea anyway if  $\sigma = 0$  is a possibility since the derivative of  $\sigma^2$  with respect to  $\sigma$  goes to zero as you approach  $\sigma = 0$ .

There is often a need for information beyond what can be obtained with the Kalman filter. For instance, in many cases, what's of particular interest is one of the states. Perhaps the state space model is designed to extract an estimate of potential GDP from a GDP series with cycles; or a "state of the business cycle" from several observables, or to pull an estimate of seasonally adjusted data from data with seasonality. The Kalman filter only gives us the estimates at time  $t$  given data through  $t$ ; the early values are thus likely to be very imprecise. The Kalman smoother takes the information produced by the Kalman filter and produces the sequence  $X_{t|T}$  (and also  $\Sigma_{t|T}$ ), that is, our estimates of the states given all the data. The derivation of the Kalman smoother is beyond the scope, but can be found in (for instance) Hamilton.<sup>2</sup>

The Kalman smoother gives us better information about the states, but in many cases, we need more than that. It's easy to generate an unconditional draw for the states because of the sequential independence. We just draw  $X_0$  from  $N(X_{0|0}, \Sigma_{0|0})$ , draw  $w_1$  from  $N(0, M_1)$  and use (8.1) to get  $X_1 = A_1 X_0 + F_1 w_1$ . We can repeat to get a whole record. That, however, isn't very interesting since the generated states will have no relationship to the observed data.<sup>3</sup> Of greater interest is a draw for the states *conditional on the data*. The Kalman smoother gives us the mean and covariance matrix for each state individually, but that isn't enough to allow us to do draws, since conditional on the data, the states are highly correlated. Conditional simulation is even more complicated than Kalman smoothing. Koop describes an algorithm from De Jong & Shepard (1995); RATS uses an alternative from Durbin and Koopman. These produce the same types of results —Durbin and Koopman's is simpler to implement.

In the instruction **DLM**, the type of analysis is chosen by the **TYPE** option. The default for this is **TYPE=FILTER**; the alternatives are **TYPE=SMOOTH** (Kalman smoothing), **TYPE=SIMULATE** (unconditional simulation) and **TYPE=CSIMULATE** (conditional simulation).

<sup>2</sup>Durbin and Koopman's discussion is more useful if you're trying to program the Kalman smoother itself, but is very hard to follow if you're just trying to understand it.

<sup>3</sup>The one real use of the unconditional simulations is to generate out-of-sample forecasts.



As our first example, let's look at the HP filter. This uses the model described in (8.10) with the ratio of the variances pegged at 1600. So there's one free parameter: the variance of the measurement error  $\sigma_\varepsilon^2$ . The HP filtered series is the extracted Kalman smoothed  $\mu_t$  (the first state).<sup>4</sup> While the mean is independent of the value of  $\sigma_\varepsilon^2$ , the variance in the estimate of  $\mu_t$  isn't. If we treat the states as an additional set of parameters, we have a simple way to do inference on it: draw (in some fashion) a value for  $\sigma_\varepsilon^2$ , then use the conditional simulation to draw  $\mu_t$  conditional on the data and on  $\sigma_\varepsilon^2$ .

We'll do the inference on  $\log \sigma_\varepsilon^2$  by independence MH. We'll use maximum likelihood estimates to provide the proposal density for this. We set up the model and estimate it with:

```
compute a=||1.0,1.0|0.0,1.0||
compute f=||0.0|1.0||
compute c=||1.0|0.0||
nonlin lsv
compute lsv=-2.0
dlm(a=a,c=c,f=f,y=lgdp,sv=exp(lsv),sw=(exp(lsv)/1600.0),var=known,$
    exact,method=bfgs) 1959:1 2006:4
```

On the **DLM** instruction, the **A**, **C**, **F** and **Y** options are used to provide those matrices (or formulas for them). The formula for the measurement error is put in using the **SV** option. Since our free parameter is  $\log \sigma_\varepsilon^2$ , we need to **exp** that to get the variance itself. Similarly, the variance for the one error component in the state equation is  $\sigma_\varepsilon^2/1600$ , so we put the formula in for that in terms of the **LSV** parameter. **VAR=KNOWN** (which is the default anyway) indicates that all the variances are given by the options; an alternative is **VAR=CONCENTRATE**, which makes them all relative to one standardized variance. As mentioned above, **EXACT** is used to handle the pre-sample states as fully diffuse.

This is generally a very well-behaved optimization, so we can just use the standard asymptotic distribution as the proposal density for independence MH. (We get about 98% acceptances, which is what you would like to see with independence MH). This is the pre-simulation setup:

```
*
* Pull out coefficient and variance for use in simulations
*
compute lsvmean=lsv, lsvsxx=sqrt(%xx(1,1))
*
* Keep track of last values for MH sampling
*
compute logplast=%logl, logqlast=0.0, lsvdraw=lsvmean
```

and this is inside the loop:

---

<sup>4</sup>Note that you can compute that with RATS with the instruction **FILTER (TYPE=HP)**. We're planning to go beyond the simple calculation of the filtered series.

```

*
* Draw lsv from the Normal approximation
*
compute lsv=lsvmean+lsvsxx*%ran(1.0)
compute logqtest=%ranlogkernel()
*
* Evaluate the likelihood function at the drawn value
*
dlm(a=a,c=c,f=f,y=lgdp,sv=exp(lsv),sw=(exp(lsv)/1600.0),$,
    exact) 1959:1 2006:4
compute logptest=%funcval
*
compute alpha =exp(logptest-logplast+logqlast-logqtest)

```

The **DLM** without the **METHOD** option just runs the Kalman filter with the model as input, producing %**LOGL** as the log likelihood.

To get the simulations for the filtered data, we do another **DLM** instruction, this time with the **TYPE=CSIM** option:

```

dlm(a=a,c=c,f=f,y=lgdp,sv=exp(lsv),sw=(exp(lsv)/1600.0),var=known,$
    exact,type=csim) 1959:1 2006:4 xstates
set first = first+xstates(t)(1)
set second = second+xstates(t)(1)^2

```

The third parameter on the **DLM** instruction saves the states in a **SERIES[VECT]** (a time series, each element of which is a **VECTOR**). When used in a **SET** instruction, **XSTATES(T)** is the vector of estimated or simulated states at entry **T**. So **XSTATES(T)(1)** is the first element of the state vector at  $t$ , that is,  $\mu_t$ . The **FIRST** and **SECOND** series sum up the simulated value and value squared of this. The sum and sum of squares are sufficient statistics to let us compute the mean and standard deviation. (It's possible to save the whole simulated history in order to do percentiles, but the programming is harder).

The post-processing is straightforward. We graph only a small part of the range, since over a long range the series all land on top of each other.

```

set first = first/ndraws
set second = sqrt(second/ndraws-first^2)
set lower = first-2.0*second
set upper = first+2.0*second
*
graph(footer="Error bands for HP filter") 4
# lgdp 2000:1 *
# first 2000:1 *
# lower 2000:1 * 3
# upper 2000:1 * 3

```

You might ask why, in the HP filter, the second variance is pegged by ratio to the first rather than being estimated separately. With this particular model,

the likelihood is highest (for almost any series) for a ratio which is much lower than the standard 1600. With a lower ratio, the trend series tracks the series itself more closely. That's not what is wanted, however. The desired trend series is supposed to be fairly "stiff"; the 1600 value is a very (infinitely) tight prior on the ratio that achieves the desired level of stiffness.

Conditional simulation also makes possible a Gibbs sampling approach to estimating the hyperparameters of the state space model.<sup>5</sup> In addition to the simulated states, you can also get the simulated errors; these are obtained using the `WHAT` (read `w-hat`) and `VHAT` options. Given these, the hyperparameters can be estimated as if they were just parameters on sample statistics. The formulas for this are worked out in Koop on page 197.

Our application is to the local level model, applied to an example from Durbin and Koopman.

$$\begin{aligned}\alpha_t &= \alpha_{t-1} + u_t \\ y_t &= \alpha_t + \varepsilon_t\end{aligned}$$

This has two hyperparameters:  $h$  is the precision of  $\varepsilon_t$ , and  $\eta$  is the relative variance. These are both given very diffuse gamma priors.

```
compute nuh=1.0
compute s2h=100.0^2
compute hdraw=nuh/s2h
*
compute nueta=1.0
compute s2eta=.1
compute etadraw=s2eta/nueta
```

Given values for the two hyperparameters, we can simulate the shocks with

```
dlim(type=csim,exact,y=nile,c=1.0,sv=1.0/hdraw,$
sw=etadraw/hdraw,var=known,what=what,vhat=vhat) / xstates
```

By default,  $A$  is the identity matrix (or 1, in this case) and so is  $F$ , which explains why this has so few inputs. Remember that  $h$  is the precision, so we need to divide by it to convert to the input variances.

Drawing  $\lambda$  given the simulated  $u_t$  (the `what` series) is a standard calculation, except that since it's a relative variance, not precision, the calculation is inverted so the chi-squared draw goes in the denominator. The `SSTATS` instruction computes the needed sum of squares (for `vhat` as well, which we'll also need). `what` and `vhat` are both `SERIES[VECT]`, which explains the `(t)(1)` subscripting.

```
sstats / vhat(t)(1)^2>>sumvsq what(t)(1)^2>>sumwsq
compute etadraw=(hdraw*sumwsq+nueta*s2eta)/%ranchisqr(%nobs+nueta)
```

---

<sup>5</sup>The hyperparameters are the parameters like the variances that govern the process; the "parameters" here are the states.

The draw for  $h$  given  $\eta$  and the draws for the shocks are also fairly standard. The only tricky part to this is that we have two sources of information on  $h$ , directly from the `what` series (for which  $h$  is the precision), indirectly from the `what` series, since their precision is being modeled as relative to  $h$ . That explains the `%nobs*2` in the degrees of freedom.

```
compute hdraw=$
  %ranchisqr(nuh+%nobs*2.0)/(nuh*s2h+sumvsq+sumwsq/etadraw)
```

The final example is Koop's empirical illustration 8.3.2. The results on this seem a bit off, though the technique is perfectly reasonable. This uses state space techniques to analyze a regression model (here an autoregression) with time varying parameters, with those parameters varying according to a random walk. The one major change between this and the models above is that the **C** matrix is now changing from entry to entry: the model takes the form:

$$\begin{aligned}\beta_t &= \beta_{t-1} + w_t \\ y_t &= X_t \beta_t + v_t\end{aligned}$$

so the multiplier on the  $\beta_t$  in the observation equation is  $X_t$ . For a simple model like this, you can do this most easily by defining a **FRML** which evaluates to a **VECTOR**.

```
dec frml[vect] cf
frml cf = ||1.0,dliprod{1}||
```

and use the option **C=CF** on **DLM**. If you were trying to apply this more generally, it would be better to define an **EQUATION** describing the regression model:

```
equation ipeq
# constant dliprod{1}
```

Then you could use the option **C=%EQNXVECTOR(ipeq,t)** on the **DLM** instruction.

This model has three hyperparameters, again the error precision  $h$ , and the relative variances of the shocks to each of the regression coefficients. The calculations are quite similar to the previous example. We just need to loop over the draws for the relative variances (same calculation as above, just in a loop):

```
do i=1,nlambda
  sstats 2 * what(t)(i)^2>>sumwsq
  compute lambdadraw(i)=(hdraw*sumwsq+nulambda*s2lambda)/$
    %ranchisqr(%nobs+nulambda)
end do i
```

and the calculation for  $h$  is now also quite similar, it's just that we now have `nlambda+1` separate sets of observations on  $h$ , since each of the shocks to the coefficients has a precision proportional to  $h$ .

```

compute hmatrix=inv(%diag(lambdadraw))
sstats 2 * vhat(t) (1)^2>>sumvsq %qform(hmatrix,what(t))>>sumwsq
compute hdraw=%ranchisqr(nuprior+%nobs*(nlambda+1))/$(
(nuprior*s2prior+sumvsq+sumwsq)

```

The `%qform(hmatrix,what)` computes the sums of squares of the components of `what` divided by the corresponding value of `lambdadraw`.

As mentioned in the notes in the program, the prior on the time variation is too loose to be of much practical value. This is a case where a very informative prior proves to be quite useful.

## Example 8.1 State-space model: Independence MH

```

cal(q) 1959:1
open data haversample.rat
data(format=rats) 1959:1 2006:4 gdp
log gdp / lgdp
*
compute a=||1.0,1.0|0.0,1.0||
compute f=||0.0|1.0||
compute c=||1.0|0.0||
*
* Estimate the observation equation variance as exp(lsv)
*
nonlin lsv
compute lsv=-2.0
dlm(a=a,c=c,f=f,y=lgdp,sv=exp(lsv),sw=(exp(lsv)/1600.0),var=known,$
exact,method=bfgs) 1959:1 2006:4
*
* Pull out coefficient and variance for use in simulations
*
compute lsvmean=lsv,lsvsxx=sqrt(%xx(1,1))
*
* Keep track of last values for MH sampling
*
compute logplast=%logl,logqlast=0.0,lsvdrow=lsvmean
*
* This does independence MH
*
compute nburn =100
compute ndraws=5000
*
compute accept=0
set hgibbs 1 ndraws = 0.0
*
* These are used for the first moment and second moments of the
* smoothed variables.
*
set first = 0.0
set second = 0.0
*

```

```

infobox(action=define,progress,lower=-nburn,upper=ndraws) $
  "Independence MH"
do draw=-nburn,ndraws
  *
  * Draw lsv from the Normal approximation
  *
  compute lsv=lsvmean+lsvsxx*%ran(1.0)
  compute logqtest=%ranlogkernel()
  *
  * Evaluate the likelihood function at the drawn value
  *
  dlm(a=a,c=c,f=f,y=lgdp,sv=exp(lsv),sw=(exp(lsv)/1600.0),var=known,$
    exact) 1959:1 2006:4
  compute logptest=%funcval
  *
  compute alpha =exp(logptest-logplast+logqlast-logqtest)
  if %ranflip(alpha)
    compute logqlast=logqtest,logplast=logptest,$
      accept=accept+1,lsvdrow=lsv
  infobox(current=draw) %strval(100.0*accept/(draw+nburn+1),"##.##")
  if draw<=0
    next
  *
  * Do the bookkeeping here.
  *
  dlm(a=a,c=c,f=f,y=lgdp,sv=exp(lsv),sw=(exp(lsv)/1600.0),var=known,$
    exact,type=csim) 1959:1 2006:4 xstates
  set first = first+xstates(t)(1)
  set second = second+xstates(t)(1)^2
  compute hgibbs(draw)=lsvdrow
end do draw
infobox(action=remove)
*
set first = first/ndraws
set second = sqrt(second/ndraws-first^2)
set lower = first-2.0*second
set upper = first+2.0*second
*
graph(footer="Error bands for HP filter") 4
# lgdp 2000:1 *
# first 2000:1 *
# lower 2000:1 * 3
# upper 2000:1 * 3

```

## Example 8.2 State Space Model: Gibbs sampling

```

*
* This model is quite a bit simpler with RATS than is described in the
* book because RATS has the "exact" option for dealing with the unit
* root in the state equation, so you don't have to difference the model
* and worry about initial conditions.
*
open data nile.dat
calendar 1871
data(format=free,org=columns,skips=1) 1871:1 1970:1 nile
*
* This has two hyperparameters. h is the precision in the observation
* equation and eta is the relative variance in the drift in level.
*
compute nuh=1.0
compute s2h=100.0^2
compute hdraw=nuh/s2h
*
compute nueta=1.0
compute s2eta=.1
compute etadraw=s2eta/nueta
*
* This does Kalman smoothing at the mean values for the prior
*
dlm(exact,type=smooth,y=nile,c=1.0,$
    sv=1.0/hdraw,sw=etadraw/hdraw,var=known) / xstates
*
* This extracts the fitted value for the local level.
*
set fitted = xstates(t)(1)
graph 2
# nile
# fitted
*
compute nburn =100
compute ndraws=2000
*
dec series      etagibbs
dec series      hgibbs
set etagibbs    1 ndraws = 0.0
set hgibbs      1 ndraws = 0.0
*
* These will keep track of the sum and the sum of squares of the
* fitted series.
*
set fitsum      = 0.0
set fitsumsq    = 0.0
*
* This does maximum likelihood estimates for h and eta
*
nonlin(parmset=dlmparms) h eta
compute h=hdraw,eta=etadraw
dlm(exact,parmset=dlmparms,method=bfgs,y=nile,c=1.0,$

```

```

    sv=1.0/h, sw=eta/h, var=known)
*
infobox(action=define, progress, lower=-nburn, upper=ndraws) $
  "Gibbs Sampler"
do draw=-nburn, ndraws
  *
  * The key to analyzing the state space model using the Gibbs sampler
  * is DLM with type=csim. This does a draw from the joint distribution
  * of the measurement error and state disturbances subject to hitting
  * the observed data. The <<what>> and <<vhat>> have the simulated
  * values for those disturbances.
  *
  dlm(type=csim, exact, y=nile, c=1.0, var=known, $
    sv=1.0/hdraw, sw=etadraw/hdraw, what=what, vhat=vhat) / xstates
  *
  * Draw eta given h and the simulated w's. Because eta is a relative
  * *variance* (rather than precision), the %ranchisqr is in the
  * denominator
  *
  sstats / vhat(t)(1)^2>>sumvsq what(t)(1)^2>>sumwsq
  compute etadraw=(hdraw*sumwsq+nueta*s2eta)/%ranchisqr(%nobs+nueta)
  *
  * Draw h given eta and the simulated w's and v's
  *
  compute hdraw=$
    %ranchisqr(nuh+%nobs*2.0) / (nuh*s2h+sumvsq+sumwsq/etadraw)
  infobox(current=draw)
  if draw<=0
    next
  *
  * Do the bookkeeping here.
  *
  compute etagibbs(draw)=etadraw
  compute hgibbs(draw)=hdraw
  set fitsum = fitsum+xstates(t)(1)
  set fitsumsq = fitsumsq+xstates(t)(1)^2
end do draw
infobox(action=remove)
*
* Show the mean and upper and lower one standard deviation bounds for
* the fitted value.
*
set fitsum = fitsum/ndraws
set fitsumsq = fitsumsq/ndraws-fitsum^2
set lower = fitsum-sqrt(fitsumsq)
set upper = fitsum+sqrt(fitsumsq)
*
graph(footer="Fitted Values for Nile Data") 4
# nile
# fitsum
# lower / 3
# upper / 3

```



### Example 8.3 State space model: Time-varying coefficients

```

open data dliprod.dat
calendar(a) 1701
data(format=free,org=columns) 1701:01 1992:01 dliprod
set dliprod = dliprod*100.0
*
* The model is a one-lag autoregression with the two coefficients
* varying with independent random walks. This basic analysis would work
* with any such regression with time-varying parameters.
*
* The observation equation is the current set of states (coefficients) x
* (1, IP(t-1)). This creates the "C" vector in  $y(t)=C(t)'X(t)+v(t)$ . Since
* it depends upon time (through the IP(t-1)), it needs to be a
* FRML[VECT].
*
dec frml[vect] cf
frml cf = ||1.0,dliprod{1}||
*
* There are three "hyperparameters" governing the process: the precision
* of measurement equation error (h) and the relative variances of the
* shocks to the two coefficients in the AR(1) model. In addition, the
* model will generate simulated paths for the coefficients themselves.
* In many applications, those might be the subject of interest.
*
* Prior and initial value for h
*
compute hdraw=1.0
compute nuprior=1.0
compute s2prior=1.0
*
* Prior and initial values for lambda. We'll start at lambda=0 (no time
* variation) since we really don't know what might be appropriate.
*
compute nlambdas=2
compute [vector] lambdadraw=%zeros(nlambdas,1)
compute nulambdas=1
compute s2lambdas=1.0
*
compute nburn =1000
compute ndraws=20000
*
dec series[vect] lgibbs
dec series      hgibbs
gset lgibbs 1 ndraws = %zeros(nlambdas,1)
set  hgibbs 1 ndraws = 0.0
*
* This estimates the meta parameters using maximum likelihood
*
compute h=hdraw,lambda=lambdadraw
nonlin(parmset=dlmparms) h lambda
compute h=1.0,lambda=%fill(nlambdas,1,s2lambdas)
dlm(method=bfgs,type=smooth,parmset=dlmparms,exact,c=cf,y=dliprod,$
    sw=(1.0/h)*%diag(lambda),sv=1.0/h,var=known,vhat=vhat) 2 * alpha

```

```

*
* This will be the point estimate of the evolution of the lag
* coefficient (second component of the state vector).
*
set a1 2 * = alpha(t) (2)
graph(footer="Time varying AR coefficient")
# a1 2 *
*
* These will be used to hold the sum and sum of squared of the
* simulations of the autoregressive coefficient
*
set a1    = 0.0
set alsq = 0.0

infobox(action=define,progress,lower=-nburn,upper=ndraws) "Gibbs Sampler"
do draw=-nburn,ndraws
  *
  * The key to analyzing the state space model using the Gibbs sampler
  * is DLM with type=csim. This does a draw from the joint distribution
  * of the measurement error and state disturbances subject to hitting
  * the observed data. The <<what>> and <<vhat>> have the simulated
  * values for those disturbances.
  *
  dlm(type=csim,exact,c=cf,y=dliprod,sw=(1.0/hdraw)*%diag(lambdadraw), $
      sv=1.0/hdraw,what=what,vhat=vhat,var=known) 2 * alpha
  *
  * Add in the value and squared values for the lag coefficient
  *
  set a1 2 * = a1 +alpha(t) (2)
  set alsq 2 * = alsq+alpha(t) (2)^2
  *
  * Given the draws for w's and current h, draw lambdas. Because
  * lambdas are relative variances*, not precisions, the %ranchisqr is
  * in the denominator
  *
  do i=1,nlambda
    sstats 2 * what(t) (i)^2>>sumwsq
    compute lambdadraw(i)=$
      (hdraw*sumwsq+nulambda*s2lambda)/%ranchisqr(%nobs+nulambda)
  end do i
  *
  * Given the draws for v's, w's and lambda, draw h
  *
  compute hmatrix=inv(%diag(lambdadraw))
  sstats 2 * vhat(t) (1)^2>>sumvsq %qform(hmatrix,what(t))>>sumwsq
  compute hdraw=%ranchisqr(nuprior+%nobs*(nlambda+1)) / $
    (nuprior*s2prior+sumvsq+sumwsq)
  infobox(current=draw)
  if draw<=0
    next
  *
  * Do the bookkeeping here. The graphs in figure 8.3 are almost
  * certainly lambda/h or its square root (probably the latter).
  * lambda itself is not particularly interesting - lambda/h is the

```

```

* variance in the shock to the coefficients.
*
compute lgibbs(draw)=lambdadraw/hdraw
compute hgibbs(draw)=hdraw
end do draw
infobox(action=remove)
@mcmcpostproc(ndraws=ndraws,mean=lmean,stderrs=lsterrs,$
  cd=lcd,nse=lnse) lgibbs
*
* Compute and graph the mean and upper and lower 1-std error bounds on
* the histories for the lag coefficient. As frequently happens with
* models like this, the highest likelihood is achieved with
* hyperparameters which allow too much time variation.
*
set a1      = a1/ndraws
set alsq    = sqrt(alsq/ndraws-a1^2)
set lower   = a1-alsq
set upper   = a1+alsq
graph 3
# a1
# lower / 2
# upper / 2
*
* Graph the posterior densities. The closest match is the square root of
* lambda/h, which would be the standard deviations of the evolution. As
* mentioned above, this is way too high to be a useful model. Allowing
* the AR coefficient to move over a range this size from period to
* period isn't reasonable. You can get more sensible results with a much
* tighter prior on lambda, say nulambda=20 with s2lambda of 1.0/the
* number of observations. Since the variance of a T-period random walk
* is the variance of the increment x T, a more reasonable prior for the
* variance on the increment would be 1/T, which would say that wandering
* over a range on the order of 1 for the full sample isn't unreasonable.
*
set lambda0 1 ndraws = sqrt(lgibbs(t)(1))
set lambda1 1 ndraws = sqrt(lgibbs(t)(2))
spgraph(footer="Figure 8.3. Posterior densities for lambdas",vfields=2)
density(grid=automatic,maxgrid=100) lambda0 / x0 f0
scatter(style=line,vmin=0.0)
# x0 f0
density(grid=automatic,maxgrid=100) lambda1 / x1 f1
scatter(style=line,vmin=0.0)
# x1 f1
spgraph(done)

```

### General Information on RATS Instructions

---

#### Integrating Constants

All density functions in RATS and all reported log likelihoods from statistical instructions include all integrating constants.

#### Evaluating Likelihoods

The following non-linear estimation instructions have `METHOD=EVALUATE` options, which just do a single function evaluation at the current values for the parameters. All define the variable `%LOGL`.

<b>BOXJENK</b>	INITIAL option for setting parameters
<b>CVMODEL</b>	
<b>DLM</b>	called <code>METHOD=SOLVE</code>
<b>GARCH</b>	use INITIAL option for setting parameters
<b>MAXIMIZE</b>	
<b>NLLS</b>	

#### `%RANLOGKERNEL()` function

After doing any single set of random draws, you can retrieve the (log) of the kernel for the density for that set of draws using `%RANLOGKERNEL()`. The kernel are the factors in the density that include the values being drawn, omitting any multiplicative factors that don't include those.

```
compute xbeta =xbeta0+%ranmvnormal(fbeta)
compute gx    =%ranlogkernel()
```

## Probability Distributions

### B.1 Uniform

<b>Parameters</b>	Lower ( $a$ ) and upper ( $b$ ) bounds
<b>Kernel</b>	1
<b>Support</b>	$[a, b]$
<b>Mean</b>	$\frac{a + b}{2}$
<b>Variance</b>	$\frac{(b - a)^2}{12}$
<b>Main uses</b>	priors and approximate posteriors for parameters that have a limited range.
<b>Density function</b>	trivial
<b>Moment Matching</b>	<code>%UniformParms(mean, sd)</code> (external function) returns the 2-vector of parameters $(a, b)$ for a uniform with the given mean and standard deviation.
<b>Random Draws</b>	<code>%UNIFORM(a, b)</code> draws one or more (depending upon the target) independent $U(a, b)$ .
<b>Notes</b>	$U(0, 1)$ is a special case of the beta distribution (beta parameters are 1 and 1).

## B.2 Univariate Normal

<b>Parameters</b>	Mean ( $\mu$ ), Variance ( $\sigma^2$ )
<b>Kernel</b>	$\sigma^{-1} \exp \left( -\frac{(x - \mu)^2}{2\sigma^2} \right)$
<b>Support</b>	$(-\infty, \infty)$
<b>Mean</b>	$\mu$
<b>Variance</b>	$\sigma^2$
<b>Main uses</b>	Distribution of error terms in univariate processes. Asymptotic distributions. Prior, exact and approximate posteriors for parameters with unlimited ranges.
<b>Density Function</b>	<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 $\log f(x \mu, \sigma^2)$ .
<b>CDF</b>	<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).
<b>Inverse CDF</b>	<code>%INVNORMAL(p)</code> is the standard Normal inverse CDF.
<b>Random Draws</b>	<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 $N(0, 1)$ .

### B.3 Univariate Student ( $t$ )

<b>Parameters</b>	Mean ( $\mu$ ), Variance of underlying Normal ( $\sigma^2$ ) or of the distribution itself ( $s^2$ ), Degrees of freedom ( $\nu$ )
<b>Kernel</b>	$\left(1 + (x - \mu)^2 / (\sigma^2 \nu)\right)^{-(\nu+1)/2} \text{ or } \left(1 + (x - \mu)^2 / (s^2(\nu - 2))\right)^{-(\nu+1)/2}$
<b>Support</b>	$(-\infty, \infty)$
<b>Mean</b>	$\mu$
<b>Variance</b>	$\sigma^2 \nu / (\nu - 2)$ or $s^2$
<b>Main uses</b>	Small sample distributions of univariate statistics. Fatter-tailed alternative to Normal for error processes. Prior, exact and approximate posteriors for parameters with unlimited ranges.
<b>Density Function</b>	<p><code>%TDENSITY(x, nu)</code> is the (non-logged) density function for a standard (<math>\mu = 0, \sigma^2 = 1</math>) <math>t</math>.</p> <p><code>%LOGTDENSITY(ssquared, u, nu)</code> is the log density based upon the <math>s^2</math> parameterization.</p> <p>Use <code>%LOGTDENSITY(ssquared, x-mu, nu)</code> to compute <math>\log f(x \mu, s^2, \nu)</math> and <code>%LOGTDENSITYSTD(sigmasq, x-mu, nu)</code> to compute <math>\log f(x \mu, \sigma^2, \nu)</math>.</p>
<b>CDF</b>	<p><code>%TCDF(x, nu)</code> is the CDF for a standard <math>t</math>.</p> <p><code>%TTEST(x, nu)</code> is the two-tailed probability of a standard <math>t</math></p>
<b>Inverse CDF</b>	<p><code>%INVTCDF(p, nu)</code> is the inverse CDF for a standard <math>t</math>.</p> <p><code>%INVTTEST(p, nu)</code> is the critical value for a two-tailed standard <math>t</math> test.</p>
<b>Random Draws</b>	<code>%RANT(nu)</code> draws one or more (depending upon the target) standard $t$ 's with independent numerators and a <i>common</i> denominator. To get a draw from a $t$ density with variance <code>ssquared</code> and <code>nu</code> degrees of freedom, use <code>%RANT(nu)*sqrt(ssquared*(nu-2.)/nu)</code> .
<b>Notes</b>	With $\nu = 1$ , this is a Cauchy (no mean or variance); with $\nu \leq 2$ , the variance doesn't exist. $\nu \rightarrow \infty$ tends towards a Normal.

## B.4 Beta distribution

<b>Parameters</b>	2 (called $\alpha$ and $\beta$ below)
<b>Kernel</b>	$x^{\alpha-1} (1-x)^{\beta-1}$
<b>Support</b>	$[0, 1]$ . If $\alpha > 1$ , the density is 0 at $x = 0$ ; if $\beta > 1$ , the density is 0 at $x = 1$ .
<b>Mean</b>	$\frac{\alpha}{\alpha + \beta}$
<b>Variance</b>	$\frac{\alpha\beta}{(\alpha + \beta)^2 (\alpha + \beta + 1)}$
<b>Main uses</b>	priors and approximate posteriors for parameters that measure fractions, or probabilities, or autoregressive coefficients (when a negative value is unreasonable).
<b>Density function</b>	<code>%LOGBETADENSITY (x, a, b)</code>
<b>Random Draws</b>	<code>%RANBETA (a, b)</code> draws one or more (depending on the target) independent Betas.
<b>Moment Matching</b>	<code>%BetaParms(mean,sd)</code> (external function) returns the 2-vector of parameters for a beta with the given mean and standard deviation.
<b>Extensions</b>	If $x \sim \text{Beta}(\alpha, \beta)$ then $ax+b$ is distributed on $[a, a+b]$



## B.5 Chi-Squared Distribution

<b>Parameters</b>	Degrees of freedom ( $\nu$ ).
<b>Kernel</b>	$x^{(\nu-2)/2} \exp(-x/2)$
<b>Range</b>	$[0, \infty)$
<b>Mean</b>	$\nu$
<b>Variance</b>	$2\nu$
<b>Main uses</b>	Asymptotic distribution. Prior, exact and approximate posterior for the precision (reciprocal of variance) of residuals or other shocks in a model.
<b>Density function</b>	<code>%CHISQRDENSITY(x, nu)</code> is the (non-logged) density with <code>nu</code> degrees of freedom at <code>x</code> .
<b>Tail Probability</b>	<code>%CHISQR(x, nu)</code> returns the probability that a chi-squared with <code>nu</code> degrees of freedom exceeds <code>x</code> .
<b>Inverse Tail Probability</b>	<code>%INVCHISQR(p, nu)</code> returns the critical value for probability $p$ .
<b>Random Draws</b>	<code>%RANCHISQR(nu)</code> draws one or more (depending upon the target) independent chi-squareds with <code>NU</code> degrees of freedom.

## B.6 (Scaled) Inverse Chi-Squared Distribution

<b>Parameters</b>	Degrees of freedom ( $\nu$ ) and scale ( $\tau^2$ ). 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 $\tau^2$ for large degrees of freedom.)
<b>Kernel</b>	$x^{-(a+1)} \exp(-bx^{-1})$
<b>Integrating Constant</b>	$b^a / \Gamma(a)$
<b>Support</b>	$[0, \infty)$
<b>Mean</b>	$\frac{\nu\tau^2}{\nu-2}$ if $\nu > 2$
<b>Variance</b>	$\frac{2\nu^2\tau^4}{(\nu-2)^2(\nu-4)}$ if $\nu > 4$
<b>Main uses</b>	Prior, exact and approximate posterior for the variance of residuals or other shocks in a model. The closely-related inverse gamma (Appendix B.8) can be used for that as well, but the scaled inverse chi-squared tends to be more intuitive.
<b>DensityFunction</b>	<code>%LOGINVCHISQRDENSITY(x, nu, tausq)</code> . Added with RATS 9.1.
<b>Moment Matching</b>	<code>%InvChisqrParms(mean, sd)</code> (external function) returns the 2-vector of parameters ( $(\nu, \tau^2)$ 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 $\nu$ which gives an infinite variance.
<b>Random draws</b>	You can use <code>(nu*tausq)/%ranchisqr(nu)</code> . Note that you divide by the random chi-squared.

## B.7 Gamma Distribution

<b>Parameters</b>	shape ( $a$ ) and scale ( $b$ ), alternatively, degrees of freedom ( $\nu$ ) and mean ( $\mu$ ). 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 $\nu$ degrees of freedom is a special case with $\mu = \nu$ .
<b>Kernel</b>	$x^{a-1} \exp\left(-\frac{x}{b}\right)$ or $x^{(\nu/2)-1} \exp\left(-\frac{x\nu}{2\mu}\right)$
<b>Support</b>	$[0, \infty)$
<b>Mean</b>	$ba$ or $\mu$
<b>Variance</b>	$b^2a$ or $\frac{2\mu^2}{\nu}$
<b>Main uses</b>	Prior, exact and approximate posterior for the precision (reciprocal of variance) of residuals or other shocks in a model
<b>Density function</b>	<code>%LOGGAMMADENSITY (x, a, b)</code> . For the $\{\nu, \mu\}$ parameterization, use <code>%LOGGAMMADENSITY (x, .5*nu, 2.0*mu/nu)</code>
<b>Random Draws</b>	<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> .
<b>Moment Matching</b>	<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.

## B.8 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 B.6 with parameters  $\nu$  (degrees of freedom) and  $\tau^2$  (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

$$\frac{b}{(a-1)} \text{ if } a > 1$$

### Variance

$$\frac{b^2}{(a-1)^2(a-2)} \text{ if } a > 2$$

### 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

`%LOGINVGAMMADENSITY(x, a, b)`. Added with RATS 9.1.

### Moment Matching

`%InvGammaParms(mean, sd)` (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 `sd` 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 `b/%rangamma(a)`. A draw from a scaled inverse chi-squared is typically done with `nu*tausqr/%ranchisqr`

## B.9 Bernoulli Distribution

<b>Parameters</b>	probability of success ( $p$ )
<b>Kernel</b>	$p^x(1 - p)^{1-x}$
<b>Range</b>	0 or 1
<b>Mean</b>	$p$
<b>Variance</b>	$p(1 - p)$
<b>Main uses</b>	Realizations of 0-1 valued variables (usually latent states).
<b>Random Draws</b>	<p><code>%RANBRANCH(  p1, p2  )</code> draws one or more independent trials with (integer) values 1 or 2. The probability of 1 is <math>\frac{p1}{p1 + p2}</math> and the probability of 2 is <math>\frac{p2}{p1 + p2}</math>. (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>

## B.10 Multivariate Normal

<b>Parameters</b>	Mean ( $\mu$ ), Covariance matrix ( $\Sigma$ ) or precision ( $H$ )
<b>Kernel</b>	$ \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)$
<b>Support</b>	$\mathbb{R}^n$
<b>Mean</b>	$\mu$
<b>Variance</b>	$\Sigma$ or $H^{-1}$
<b>Main uses</b>	Distribution of multivariate error processes. Asymptotic distributions. Prior, exact and approximate posteriors for a collection of parameters with unlimited ranges.
<b>Density Function</b>	<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).
<b>Distribution Function</b>	<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.
<b>Random Draws</b>	<p><code>%RANMAT(m,n)</code> draws a matrix of independent <math>N(0, 1)</math>.  <code>%RANMVNORMAL(F)</code> draws an <math>n</math>-vector from a <math>N(0, FF')</math>, where <math>F</math> 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 <math>N(\mu, \Sigma)</math>, 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.</p>

## B.11 Multivariate Student (*t*)

<b>Parameters</b>	Mean ( $\mu$ ), covariance matrix of the underlying multivariate Normal ( $\Sigma$ ) or of the distribution itself ( $S$ ), Degrees of freedom ( $\nu$ )
<b>Kernel</b>	$(1 + (x - \mu)' \Sigma^{-1} (x - \mu) / \nu)^{-(\nu+n)/2}$ or $(1 + (x - \mu)' S^{-1} (x - \mu) / (\nu - 2))^{-(\nu+n)/2}$
<b>Support</b>	$\mathbb{R}^n$
<b>Mean</b>	$\mu$
<b>Variance</b>	$\Sigma \frac{\nu}{(\nu - 2)}$ or $S$
<b>Main uses</b>	Prior, exact and approximate posteriors for sets of parameters with unlimited ranges.
<b>Density Function</b>	<p><code>%LOGTDENSITY(s, u, nu)</code> is the log density based upon the <math>S</math> parameterization.</p> <p>Use <code>%LOGTDENSITY(s, x-mu, nu)</code> to compute <math>\log f(x \mu, S, \nu)</math> and <code>%LOGTDENSITYSTD(sigma, u, nu)</code> to compute <math>\log f(x \mu, \Sigma, \nu)</math>.<sup>1</sup> The same functions work for both univariate and multivariate distributions.</p>
<b>Random Draws</b>	<p><code>%RANT(nu)</code> or <code>%RANMVT(fsigma, nu)</code> <code>%RANT(nu)</code> draws one or more (depending upon the target) standard <math>t</math>'s with independent numerators and a common denominator. <code>%RANMVT(fsigma, nu)</code> draws a multivariate <math>t</math> with <code>fsigma</code> as a “square root” of the <math>\Sigma</math> matrix. You would typically compute <code>fsigma</code> as <code>%DECOMP(sigma)</code>. The function is parameterized this way in case you need to do many draws with a single <code>sigma</code>, as you can do the decomposition in advance.</p> <p>To draw a multivariate <math>t</math> with covariance matrix <math>S</math> and <code>nu</code> degrees of freedom, use <code>%RANMVT(%DECOMP(s*((nu-2.)/nu)))</code>.</p>

<sup>1</sup>`%LOGTDENSITYSTD` and `%RANMVT` were added with RATS 7.3. Before that, use `%LOGTDENSITY(sigma*nu/(nu-2), x-mu, nu)` and `mu+%decomp(sigma)*(u=%rant(nu))`

## B.12 Wishart Distribution

<b>Parameters</b>	Scaling $\mathbf{A}$ (symmetric $n \times n$ matrix) and degrees of freedom ( $\nu$ ). This only has a proper density if $\nu > n - 1$ and $\mathbf{A}$ is positive definite.
<b>Kernel</b>	$\exp\left(-\frac{1}{2}\text{trace}(\mathbf{A}^{-1}\mathbf{X})\right)  \mathbf{X} ^{\frac{1}{2}(\nu-n-1)}$
<b>Support</b>	Positive definite symmetric matrices
<b>Mean</b>	$\nu\mathbf{A}$
<b>Main uses</b>	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 B.13) since that would be the distribution of the covariance matrix itself.
<b>Random Draws</b>	<p><code>%RANWISHART (n, nu)</code> draws a single <math>n \times n</math> Wishart matrix with <math>\mathbf{A} = \mathbf{I}</math> and degrees of freedom <math>\nu</math>.</p> <p><code>%RANWISHARTF (F, nu)</code> draws a single <math>n \times n</math> Wishart matrix with <math>\mathbf{A} = \mathbf{F}\mathbf{F}'</math> and degrees of freedom <math>\nu</math>. <math>\mathbf{F}</math> can be any factor of <math>\mathbf{A}</math>, but would typically be computed as the Cholesky factor using <code>%DECOMP</code>.</p>



## B.13 Inverse Wishart Distribution

<b>Parameters</b>	Scaling $\Psi$ (symmetric $n \times n$ matrix) and degrees of freedom ( $\nu$ ). This only has a proper density if $\nu > n - 1$ and $\Psi$ is positive definite.
<b>Kernel</b>	$\exp\left(-\frac{1}{2}\text{trace}(\Psi\mathbf{X}^{-1})\right)  \mathbf{X} ^{-\frac{1}{2}(\nu+n+1)}$
<b>Support</b>	Positive definite symmetric matrices
<b>Mean</b>	$\frac{1}{\nu-n-1}\Psi$
<b>Main uses</b>	Prior, exact and approximate posterior for the covariance matrix of residuals in a multivariate regression with Gaussian residuals.
<b>Diffuse versions</b>	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.
<b>Combining Densities</b>	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 $\mathbf{X} \sim IW(\Psi_1 + \Psi_2, \nu_1 + \nu_2 + n + 1)$ .
<b>Random Draws</b>	<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 $\nu$ . 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> .
<b>Notes</b>	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 $\Psi$ 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  $\Sigma$ , call this  $\Sigma_0$ . The corresponding value of  $\Psi$  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  $\Psi$  matrix which is  $T\hat{\Sigma} + \alpha\Sigma_0$ , thus (roughly) sample and non-sample information on  $\Sigma$  weighted by the number of actual and “dummy” observations.

## Properties of Multivariate Normals

The density function for a jointly Normal random  $n$ -vector  $\mathbf{x}$  with mean  $\mu$  and covariance matrix  $\Sigma$  is

$$(2\pi)^{-n/2} |\Sigma|^{-1/2} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu)' \Sigma^{-1} (\mathbf{x} - \mu) \right)$$

Manipulations with this can quickly get very messy. Fortunately, the exponent in this (which is just a scalar – the result of the quadratic form) contains enough information that we can just read from it  $\mu$  and  $\Sigma$ . In particular, if we can determine that the kernel of the density of  $\mathbf{x}$  (the part of the density which includes all occurrences of  $\mathbf{x}$ ) takes the form

$$\exp \left( -\frac{1}{2} Q(\mathbf{x}) \right) \quad , \text{where} \quad Q(\mathbf{x}) = \mathbf{x}' \mathbf{A} \mathbf{x} - 2\mathbf{x}' \mathbf{b} + c$$

then, by completing the square, we can turn this into

$$Q(\mathbf{x}) = (\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})' \mathbf{A} (\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}) + (c - \mathbf{b}' \mathbf{A}^{-1} \mathbf{b})$$

Thus  $\Sigma = \mathbf{A}^{-1}$  and  $\mu = \mathbf{A}^{-1}\mathbf{b}$ . The final part of this,  $(c - \mathbf{b}' \mathbf{A}^{-1} \mathbf{b})$ , doesn't involve  $\mathbf{x}$  and will just wash into the constant of integration for the Normal. We might need to retain it for analyzing other parameters (such as the residual variance), but it has no effect on the conditional distribution of  $\mathbf{x}$  itself.

One standard manipulation of multivariate Normals comes in the Bayesian technique of combining a prior and a likelihood to produce a posterior density. In the standard Normal linear model, the data have density

$$f(\mathbf{Y}|\beta) \sim N(\mathbf{X}\beta, \sigma^2 \mathbf{I})$$

The (log) kernel of the density is

$$-\frac{1}{2\sigma^2} (\mathbf{Y} - \mathbf{X}\beta)' (\mathbf{Y} - \mathbf{X}\beta) = -\frac{1}{2} (\beta' (\sigma^{-2} \mathbf{X}' \mathbf{X}) \beta - 2\beta' \sigma^{-2} \mathbf{X}' \mathbf{Y} + \sigma^{-2} \mathbf{Y}' \mathbf{Y})$$

Looking at this as a function of  $\beta$ , we read off

$$\beta|\mathbf{Y} \sim N \left( (\mathbf{X}' \mathbf{X})^{-1} \mathbf{X}' \mathbf{Y}, \sigma^2 (\mathbf{X}' \mathbf{X})^{-1} \right) = N \left( \hat{\beta}, \sigma^2 (\mathbf{X}' \mathbf{X})^{-1} \right)$$

Now, suppose that, in addition to the data, we have the prior

$$\beta \sim N(\beta^*, \mathbf{H}^{-1})$$

and we write  $\hat{\mathbf{H}} = \sigma^{-2} (\mathbf{X}'\mathbf{X})$  (the inverse of the least squares covariance matrix). If we multiply the densities, the only parts that include  $\beta$  will be the two quadratic parts, which we can add in log form to get

$$Q(\beta) = (\beta - \hat{\beta})' \hat{\mathbf{H}} (\beta - \hat{\beta}) + (\beta - \beta^*)' \mathbf{H} (\beta - \beta^*) \quad (\text{C.1})$$

Expanding this gives us

$$Q(\beta) = \beta' (\hat{\mathbf{H}} + \mathbf{H}) \beta - 2\beta' (\hat{\mathbf{H}}\hat{\beta} + \mathbf{H}\beta^*) + \dots$$

where the extra terms don't involve  $\beta$ . Thus, the posterior for  $\beta$  is

$$\beta \sim N \left( (\hat{\mathbf{H}} + \mathbf{H})^{-1} (\hat{\mathbf{H}}\hat{\beta} + \mathbf{H}\beta^*), (\hat{\mathbf{H}} + \mathbf{H})^{-1} \right)$$

Notice that the posterior mean is a matrix weighted average of the two input means, where the weights are the inverses of the variances. The inverse of the variance (of a Normal) is known as the *precision*. Notice that precision is additive: the precision of the posterior is the sum of the precisions of the data information and prior.

If we need to keep track of the extra terms, there's a relatively simple way to evaluate this. If we write the posterior mean and precision as:

$$\bar{\beta} = (\hat{\mathbf{H}} + \mathbf{H})^{-1} (\hat{\mathbf{H}}\hat{\beta} + \mathbf{H}\beta^*), \bar{\mathbf{H}} = \hat{\mathbf{H}} + \mathbf{H}$$

then we have

$$Q(\beta) = (\beta - \bar{\beta})' \bar{\mathbf{H}} (\beta - \bar{\beta}) + Q(\bar{\beta}) \quad (\text{C.2})$$

The first term in (C.2) has value 0 at  $\bar{\beta}$ , so using this and (C.1) gives

$$Q(\bar{\beta}) = (\bar{\beta} - \hat{\beta})' \hat{\mathbf{H}} (\bar{\beta} - \hat{\beta}) + (\bar{\beta} - \beta^*)' \mathbf{H} (\bar{\beta} - \beta^*)$$

As another example, consider the partitioned process

$$\begin{bmatrix} \mathbf{Y}_1 \\ \mathbf{Y}_2 \end{bmatrix} \sim N \left( \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma'_{12} & \Sigma_{22} \end{bmatrix} \right)$$

The  $Q$  function takes the form

$$\begin{bmatrix} \mathbf{Y}_1 - \mu_1 \\ \mathbf{Y}_2 - \mu_2 \end{bmatrix}' \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma'_{12} & \Sigma_{22} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{Y}_1 - \mu_1 \\ \mathbf{Y}_2 - \mu_2 \end{bmatrix}$$

For now, let's just write the inverse in partitioned form without solving it out, thus

$$\begin{bmatrix} \mathbf{Y}_1 - \mu_1 \\ \mathbf{Y}_2 - \mu_2 \end{bmatrix}' \begin{bmatrix} \Sigma^{11} & \Sigma^{12} \\ \Sigma^{12'} & \Sigma^{22} \end{bmatrix} \begin{bmatrix} \mathbf{Y}_1 - \mu_1 \\ \mathbf{Y}_2 - \mu_2 \end{bmatrix}$$

This expands to

$$(\mathbf{Y}_1 - \mu_1)' \Sigma^{11} (\mathbf{Y}_1 - \mu_1) + 2(\mathbf{Y}_1 - \mu_1)' \Sigma^{12} (\mathbf{Y}_2 - \mu_2) + (\mathbf{Y}_2 - \mu_2)' \Sigma^{22} (\mathbf{Y}_2 - \mu_2)$$

where the cross terms are scalars which are transposes of each other, and hence equal, hence the 2 multiplier. If we now want to look at  $\mathbf{Y}_1|\mathbf{Y}_2$ , we get immediately that this has covariance matrix

$$(\Sigma^{11})^{-1}$$

and mean

$$\mu_1 - (\Sigma^{11})^{-1} \Sigma^{12} (\mathbf{Y}_2 - \mu_2)$$

If we want to reduce this to a formula in the original matrices, we can use partitioned inverse formulas to get

$$(\Sigma^{11})^{-1} = \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21}$$

and

$$(\Sigma^{11})^{-1} \Sigma^{12} = -\Sigma_{12} \Sigma_{22}^{-1}$$

thus

$$\mathbf{Y}_1|\mathbf{Y}_2 \sim N(\mu_1 + \Sigma_{12} \Sigma_{22}^{-1} (\mathbf{Y}_2 - \mu_2), \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21})$$

Note that the covariance matrix of the conditional distribution satisfies  $\Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21} \leq \Sigma_{11}$  and that it doesn't depend upon the actual data observed, even if those data are seriously in conflict with the assumed distribution.

## Non-Standard Matrix Calculations

Non-standard matrix inversion was introduced into the statistics literature by Koopman (1997).<sup>1</sup> The goal is to compute an exact (limit) inverse of an input (symmetric) matrix of the form<sup>2</sup>

$$\mathbf{A} + \kappa \mathbf{B} \text{ as } \kappa \rightarrow \infty$$

The result will be the matrix of the form

$$\mathbf{C} + \mathbf{D}\kappa^{-1} + \mathbf{E}\kappa^{-2} \quad (\text{D.1})$$

Note that the  $\kappa^{-2}$  term isn't needed in many applications. With the help of this expansion, it's possible to compute exact limits as  $\kappa \rightarrow \infty$  for calculations like:

$$(\mathbf{F} + \mathbf{G}\kappa)(\mathbf{A} + \mathbf{B}\kappa)^{-1} \approx (\mathbf{F} + \mathbf{G}\kappa)(\mathbf{C} + \mathbf{D}\kappa^{-1}) \rightarrow \mathbf{F}\mathbf{C} + \mathbf{G}\mathbf{D}$$

You might think that you could just use a guess matrix of the form (D.1) multiply, match terms and be done quickly. Unfortunately, because matrices generally don't commute, it's very easy to get a left inverse or a right inverse, but not a true inverse. The matching terms idea is correct, but it has to be done carefully. We will start with a case that's more manageable. Note that this derivation is simpler than that in Koopman's paper. Here, the "infinite" matrix is isolated into an identity block, and we look only at the expansion through  $\kappa^{-1}$ .

$$\left( \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}'_{12} & \mathbf{A}_{22} \end{bmatrix} + \kappa \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & 0 \end{bmatrix} \right) \times \quad (\text{D.2})$$

$$\left( \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}'_{12} & \mathbf{C}_{22} \end{bmatrix} + \kappa^{-1} \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}'_{12} & \mathbf{D}_{22} \end{bmatrix} \right) = \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & \mathbf{I}_{n-r} \end{bmatrix} + \kappa^{-1} \text{Rem}$$

Since the  $\kappa$  term in the product has to zero out,  $\mathbf{C}_{11} = 0$  and  $\mathbf{C}_{12} = 0$ . Using that and collecting the  $O(1)$  terms, we get

$$\left( \begin{bmatrix} \mathbf{D}_{11} & \mathbf{A}_{12}\mathbf{C}_{22} + \mathbf{D}_{12} \\ 0 & \mathbf{A}_{22}\mathbf{C}_{22} \end{bmatrix} \right) = \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & \mathbf{I}_{n-r} \end{bmatrix}$$

<sup>1</sup>*Non-standard analysis* was introduced into mathematics in the 1970's using results from "model theory" to embed the real numbers within a framework that includes "infinite" and "infinitesimal" numbers. It was hoped that this would allow simpler descriptions of limit calculations in real analysis, but never really caught on. See Blass (1978) for a brief survey of the ideas behind this.

<sup>2</sup>We're using conventional choices of  $\mathbf{A}$ ,  $\mathbf{B}$  for the names of the matrices. These are not intended to match with the use of those names elsewhere in the paper.

So we get  $\mathbf{D}_{11} = \mathbf{I}_r$ ,  $\mathbf{C}_{22} = \mathbf{A}_{22}^{-1}$  and  $\mathbf{D}_{12} = -\mathbf{A}_{12}\mathbf{A}_{22}^{-1}$ .  $\mathbf{D}_{22}$  is arbitrary; it just ends up in the remainder, so for simplicity we can make it zero. If we check the reverse multiplication

$$\left( \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{C}_{22} \end{bmatrix} + \kappa^{-1} \begin{bmatrix} \mathbf{I}_r & \mathbf{D}_{12} \\ \mathbf{D}'_{12} & 0 \end{bmatrix} \right) \left( \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}'_{12} & \mathbf{A}_{22} \end{bmatrix} + \kappa \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & 0 \end{bmatrix} \right)$$

we can verify that it also will be the identity + a term in  $\kappa^{-1}$ .

In general, we won't have an input matrix with the nice form in (D.2). However, for a p.s.d. symmetric matrix  $\mathbf{B}$ , we can always find a non-singular matrix  $\mathbf{T}$  such that  $\mathbf{T}\mathbf{B}\mathbf{T}'$  has that form.<sup>3</sup> So, in general, we can compute the inverse with:

$$(\mathbf{A} + \kappa\mathbf{B})^{-1} = \mathbf{T}'(\mathbf{T}\mathbf{A}\mathbf{T}' + \kappa\mathbf{T}\mathbf{B}\mathbf{T}')^{-1}\mathbf{T} \quad (\text{D.3})$$

For an input matrix pair  $\{\mathbf{A}, \mathbf{B}\}$ , this will produce an output pair  $\{\mathbf{C}, \mathbf{D}\}$ .

To derive the more accurate expansion in the case with the simpler form of  $\mathbf{B}$ , our test inverse is

$$\left( \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}'_{12} & \mathbf{C}_{22} \end{bmatrix} + \kappa^{-1} \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}'_{12} & \mathbf{D}_{22} \end{bmatrix} + \kappa^{-2} \begin{bmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} \\ \mathbf{E}'_{12} & \mathbf{E}_{22} \end{bmatrix} \right)$$

Everything from above goes through, except we now can't make  $\mathbf{D}_{22}$  arbitrary. When we multiply out, the  $\kappa^{-1}$  terms are

$$\begin{aligned} & \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}'_{12} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}'_{12} & \mathbf{D}_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{I}_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} \\ \mathbf{E}'_{12} & \mathbf{E}_{22} \end{bmatrix} \text{ or} \\ & \begin{bmatrix} \mathbf{A}_{11}\mathbf{D}_{11} + \mathbf{A}_{12}\mathbf{D}'_{12} & \mathbf{A}_{11}\mathbf{D}_{12} + \mathbf{A}_{12}\mathbf{D}_{22} \\ \mathbf{A}'_{12}\mathbf{D}_{11} + \mathbf{A}_{22}\mathbf{D}'_{12} & \mathbf{A}'_{12}\mathbf{D}_{12} + \mathbf{A}_{22}\mathbf{D}_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} \\ 0 & 0 \end{bmatrix} \end{aligned}$$

The bottom left element in the AD matrix is zero because of the first order solution. Since  $\mathbf{D}_{22}$  was arbitrary from before, we can now solve for it as

$$\mathbf{D}_{22} = -\mathbf{A}_{22}^{-1}\mathbf{A}'_{12}\mathbf{D}_{12} = \mathbf{A}_{22}^{-1}\mathbf{A}'_{12}\mathbf{A}_{12}\mathbf{A}_{22}^{-1}.$$

With that, we also get

$$\mathbf{E}_{11} = -\mathbf{A}_{11}\mathbf{D}_{11} - \mathbf{A}_{12}\mathbf{D}'_{12} = -\mathbf{A}_{11} + \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12}$$

and

$$\mathbf{E}_{12} = -\mathbf{A}_{11}\mathbf{D}_{12} - \mathbf{A}_{12}\mathbf{D}_{22} = (\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12})\mathbf{A}_{12}\mathbf{A}_{22}^{-1}$$

$\mathbf{E}_{22}$  is now arbitrary. In terms of the input matrix, this is:

$$\begin{aligned} & \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{A}_{22}^{-1} \end{bmatrix} + \kappa^{-1} \begin{bmatrix} \mathbf{I} & \mathbf{A}_{12}\mathbf{A}_{22}^{-1} \\ \mathbf{A}_{22}^{-1}\mathbf{A}'_{12} & \mathbf{A}_{22}^{-1}\mathbf{A}'_{12}\mathbf{A}_{12}\mathbf{A}_{22}^{-1} \end{bmatrix} + \\ & \kappa^{-2} \begin{bmatrix} -\mathbf{A}_{11} + \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12} & (\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12})\mathbf{A}_{12}\mathbf{A}_{22}^{-1} \\ \mathbf{A}_{22}^{-1}\mathbf{A}'_{12}(\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}'_{12}) & 0 \end{bmatrix} \quad (\text{D.4}) \end{aligned}$$

<sup>3</sup>The simplest to compute is based upon a modified version of the Cholesky factorization.

The extension of this to the more general  $\mathbf{B}$  matrix is as before. In the typical situation, the transforming  $\mathbf{T}$  matrix will take the form

$$\begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{bmatrix}$$

where  $\mathbf{T}_1$  is  $r \times n$  and  $\mathbf{T}_2$  is  $(n-r) \times n$ . If that's the case, then the component matrices in the formula (D.4) are  $\mathbf{A}_{11} = \mathbf{T}_1 \mathbf{A} \mathbf{T}'_1$ ,  $\mathbf{A}_{12} = \mathbf{T}_1 \mathbf{A} \mathbf{T}'$  and  $\mathbf{A}_{22} = \mathbf{T}_2 \mathbf{A} \mathbf{T}'_2$ . The expanded inverse will be (D.4) premultiplied by  $\mathbf{T}'$  and postmultiplied by  $\mathbf{T}$ .



---

## Bibliography

---

- Blanchard O & Quah D 1989 *American Economic Review* **79**(4), 655–673.
- Blass A 1978 *Bulletin of the American Mathematical Society* **84**(1), 34–41.
- De Jong P & Shepard N 1995 *Biometrika* **82**(2), 339–350.
- Durbin J & Koopman S 2012 *Time Series Analysis by State Space Methods* 2nd edn Oxford: Oxford University Press.
- Hamilton J 1994 *Time Series Analysis* Princeton: Princeton University Press.
- Koop G 2003 *Bayesian Econometrics* West Sussex: Wiley.
- Koopman S 1997 *Journal of American Statistical Association* **92**(6), 1630–1638.
- Leamer E 1974 *Specification Searches* New York: Wiley.
- Sims C & Zha T 1999 *Econometrica* **67**(5), 1113–1156.
- Waggoner D F & Zha T 2003 *Journal of Economic Dynamics and Control* **28**(2), 349–366.

---

# Index

---

||, 23

Acceptance method, 72

Antithetic acceleration, 94, 118

**AR1** instruction, 71

Bayes factor, 39

Bayesian Information Criterion, 40

Bernoulli distribution, 183

Beta distribution, 178

%BICDF function, 184

Blanchard, O., 96

**BOXJENK** instruction, 174

Burn-in, 34

@BVARBUILDPRIOR procedure, 115

@BVARBUILDPRIORMN procedure, 115

@BVARFinishPrior procedure, 115

CD measure, 38

%CDF function, 176

Chi-squared distribution, 179, 180

%CLOCK function, 116

%COLS function, 24

Complete Class Theorem, 2

Conjugate prior, 12

**CVMODEL** instruction, 174

DeJong, P., 162

%DENSITY function, 176

**DENSITY** instruction, 79

**DLM** instruction, 174

Durbin, J., 159

%EQNPRJ function, 78

%EQNREGLABELS function, 78

%EQNSETCOEFFS function, 78

%EQNSIZE function, 78

%EQNXVECTOR function, 78

**ESTIMATE** instruction, 92

**FIND** instruction, 58

Fixed effects, 137, 138, 148

Function wizard, 23

Gamma distribution, 181, 182

**GARCH** instruction, 174

Geweke CD measure, 38

Gibbs sampling, 33

Hamilton, J., 159

Hodrick-Prescott filter, 161

Hyperparameter, 69, 109, 165

Importance sampling, 122

Independence Chain Metropolis, 52

**INFOBOX** instruction, 78

Inliers, 70

Introduction, 174

Inverse Wishart distribution, 187

%INVNORMAL function, 176

%INVTcdf function, 177

%INVTTEST function, 20, 177

Koopman, S., 159

Koopman, S. J., 192

Leamer, E., 74

Likelihood principle, 1

Local trend model, 160

%LOGDENSITY function, 176, 184

%LOGL variable, 174

%LOGTDENSITY function, 13, 177, 185

%LOGTDENSITYSTD function, 13, 177, 185

**LOOP** instruction, 72

LSDV, 137

Marginal likelihood, 1

Markov Chain Monte Carlo, 33

**MAXIMIZE** instruction, 174

MCMC, *see* Markov Chain Monte Carlo

@MCMCPostProc procedure, 38

**MCOV** instruction, 43

METHOD=EVALUATE option, 174

Metropolis-Hastings, 51

Minnesota prior, 108

- Mixed estimation, 115
- MODEL data type, 92
- %MODELGETCOEFFS function, 116
- %MODELLABEL function, 116
- %MODELLAGSUMS function, 96, 116
- %MODELSETCOEFFS function, 116
- @MONTEVAR procedure, 93
- Multivariate Normal distribution, 184, 189
- Multivariate t distribution, 185
- NLIP model, 50
- NLLS instruction, 174
- Normal distribution, 176
  - multivariate, 184, 189
- Overflow, 5
- PARMSET, 57
- %PARMSPEEK function, 57
- %PARMSPOKE function, 57
- Posterior mode, 58
- Precision, 190
- PREGRESS instruction, 138
- Probability distributions
  - Bernoulli, 183
  - beta, 178
  - chi-squared, 179
  - gamma, 181
  - inverse chi-squared, 180
  - inverse gamma, 182
  - inverse Wishart, 187
  - multivariate normal, 184, 189
  - multivariate t, 185
  - normal, 176
  - t, 177
  - uniform, 175
  - Wishart, 186
- Progress bar, 78
- Proposal density, 51
- Quah, D., 96
- %RAN function, 176
- %RANBRANCH function, 183
- %RANCHISQR function, 35
- Random coefficients model, 140, 153
- Random effects, 137, 139, 150
- Random Walk Metropolis, 52, 59, 125
- %RANLOGKERNEL function, 56, 174
- %RANMAT function, 176, 184
- %RANMVKRON function, 91
- %RANMVNORMAL function, 21, 184
- %RANMVPOSTCMOM function, 36
- %RANMVT function, 21, 185
- %RANT function, 177, 185
- %RANTRUNCATE function, 143
- %RANWISHART function, 75, 186
- %RANWISHARTF function, 141, 186
- %RANWISHARTI function, 91, 92, 141, 187
- Rejection method, 72
- REPORT instruction, 24
- %RESHAPE function, 146
- %ROWS function, 24
- %RSSCMOM function, 18
- Schwarz Bayesian Criterion, 40
- SEED instruction, 7
- Seemingly unrelated regression, 73
- Serial correlation, 71
- Shephard, N., 162
- Sims, C., 96
- SSTATS instruction, 7
- Sufficient statistics, 17, 50
- SUR, 73
- @SURGIBBSDDATAINFO procedure, 77
- @SURGIBBSSETUP procedure, 77
- SURGibbsSigma function, 77
- SWEEP instruction, 117
- SYSTEM instruction, 92
- t distribution, 177
- %TCDF function, 20, 177
- %TDENSITY function, 177
- %TTEST function, 177
- Underflow, 4
- Uniform distribution, 175
- %UNIFORM function, 175
- %UNIFORMPARMS function, 175

- `%VARLAGSUMS` matrix, 96
- `%VEC` function, 146
- `%VECTORECT` function, 146
- `%VECTOSYMM` function, 146
- Wishart distribution, 75, 186
  - inverse, 187
- Wizards
  - function, 23
  - Kernel Density Estimation, 79
  - VAR (Setup/Estimate), 97
- `%XCOL` function, 24
- `%XDIAG` function, 24
- `%XROW` function, 24
- `%XSUBMAT` function, 24
- `%XSUBVEC` function, 24
- Zha, T., 96
- `%ZTEST` function, 176